# Report

## Practical assignment

## Introduction

This paper reports the result of our analysis of a data set of handwritten digits. We have divided the paper into paragraphs, where each paragraph describes a step in our analysis. All the experiments have been conducted in R language. To allow the reader to replicate the experiments, we have included the code in the “Appendix” paragraphs. All the code runs in Rstudio.

## Exploratory analysis of the data

The data are composed of 784 features, which correspond to the 784 pixels an image has. We have in total 42000 images where the corresponding class (the digit that is written in the image) has been hand labelled in the data set as first column. The values of the feature represent the colour of the pixel on the grayscale, where 0 represents completely white, and 255 represents completely black. For each of them, we extract the minimum value it takes in the data set, the maximum, the average, the number of times the pixel takes value 0 (it’s “white”) and the number of times it takes value 255 (it’s “black”).

Then we compute the mean of all these vectors.

|  |  |  |
| --- | --- | --- |
| Mean.minValue | Mean.maxValue | Mean.avgValue |
| 0 | 217.676 | 33.40891 |

|  |  |
| --- | --- |
| Mean.nwhite | Mean.nblack |
| 33955.76 | 284.8342 |

After that, we compute:

* The number of pixels which take value 0 at least once: 784 (all of them)
* The number of pixels which take value 255 at least once: 591

All these results suggest that we are dealing with a sparse matrix, where there are some pixels which always take value 0. In some sense they are “useless”, i.e. they do not bring any information we can use to build our predictive model. Moreover, we define as useless all the pixels who always take the same value in the data set.

We do so some analysis to spot those useless pixels. They are the following:

"pixel0" "pixel1" "pixel2"

"pixel3" "pixel4" "pixel5"

"pixel6" "pixel7" "pixel8"

"pixel9" "pixel10" "pixel11"

"pixel16" "pixel17" "pixel18"

"pixel19" "pixel20" "pixel21"

"pixel22" "pixel23" "pixel24"

"pixel25" "pixel26" "pixel27"

"pixel28" "pixel29" "pixel30"

"pixel31" "pixel52" "pixel53"

"pixel54" "pixel55" "pixel56"

"pixel57" "pixel82" "pixel83"

"pixel84" "pixel85" "pixel111"

"pixel112" "pixel139" "pixel140"

"pixel141" "pixel168" "pixel196"

"pixel392" "pixel420" "pixel421"

"pixel448" "pixel476" "pixel532"

"pixel560" "pixel644" "pixel645"

"pixel671" "pixel672" "pixel673"

"pixel699" "pixel700" "pixel701"

"pixel727" "pixel728" "pixel729"

"pixel730" "pixel731" "pixel754"

"pixel755" "pixel756" "pixel757"

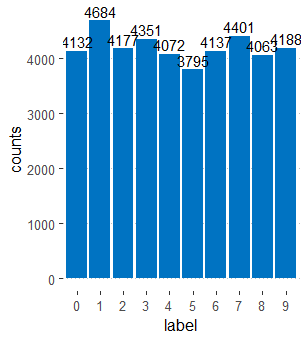
"pixel758" "pixel759" "pixel760"

"pixel780" "pixel781" "pixel782"

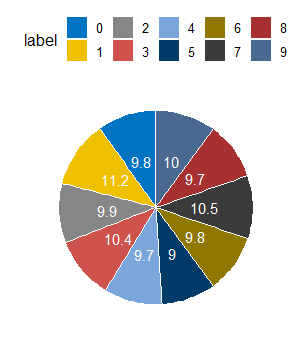
"pixel783"

They are 76 in total. All of them take always value 0. By, inspection, we find that there are no pixels which take always the same non-zero value.

Then we analyse the class distribution: we have in total 9 classes, each corresponding to a digit. We plot the following graph, with the classes on the x-axis and the counts of images with that digit on the y-axis.

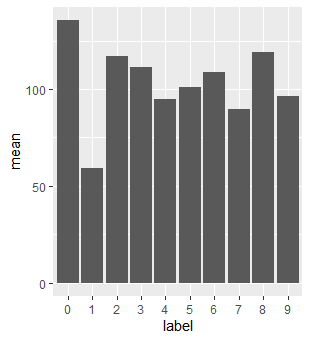


We plot a pie chart as well, where “label” stands for “class”



The majority class is the class corresponding to the digit “1”: if we predicted only that class, we would have a percentage of correct classifications of 11,2%.

Moreover, we do some analysis taking into consideration the images in the data set and aggregating the results per each class of belonging (i.e., per row instead of per column, as we have always done before). We define “dark pixels” the pixels whit a value greater than 128. Per each image, we compute the number of “dark pixels” and then compute the mean per each class. The results are in the following graph. There are few (consider that they each image has 784 pixels).



We feel that dark pixels are useful to spot the skeleton of a handwritten digit, eliminating all the remaining ink which has simply spread in the paper where the digit was written. As an example, we compare the image number 380 in the data set and the same image where we have shrunk to 0 out all the pixel which take values between 0 and 128. Note: in these images white and black are inverted.



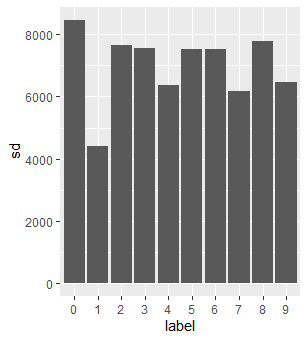
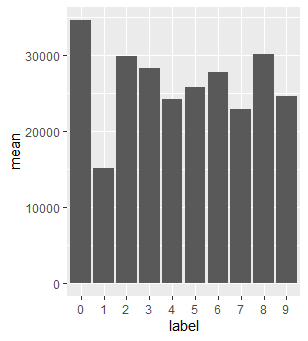
Figure : filteredl image



Figure 2: actual image

## How much ink

To measure “how much ink” a digit costs, we sum all the feature values per each element of the data we have (i.d., per each row of the dataframe). The results can be then in the interval [0; 255\*784 = 199920]. Then we aggregated the results per each digit, plotting the mean and the standard deviation. We called this feature “density”. As before, “label” means “class”.



label mean

1 0 34632.41

2 1 15188.47

3 2 29871.10

4 3 28320.19

5 4 24232.72

6 5 25835.92

7 6 27734.92

8 7 22931.24

9 8 30184.15

10 9 24553.75

Label sd

1 0 8462.916

2 1 4409.932

3 2 7653.922

4 3 7574.975

5 4 6375.416

6 5 7527.595

7 6 7531.413

8 7 6169.042

9 8 7778.354

10 9 6466.003

This feature allows us to distinguish correctly between four groups: {class 1} [which has a mean below 20000], {class 4, class 7, class 9} [which have a mean between 20000 and 25000], {class 2, class 3, class 5, class 6, class 8} [which have a mean between 25000 and 3000 and more or less the same sd], {class 0} [which has a mean over 30000]. Moreover, we feel that only class 1 can be correctly distinguished from the other ones. However, big values of standard deviations ( at least 20% for all the classes) among with the fact that all the mean values are comparable ( per each class, almost 7 of the other are within the standard deviation) suggest that the feature is not very relevant to spot a class.

We add to our data set a column with density values as a new feature. We scale it. Then we train a multinomial logistic regression model with only the density of all the 42000 images as a feature. We test it using the same density values.

We get the following confusion matrix, where the rows represent the prediction of the model, and the columns the actual values. As expected, the model has the best performances on class 1, while it never predicts classes 4,5,6,8. Class 0 is predicted very often: that’s why it’s predicted correctly many times (it has high recall but small precision)

The correct predictions are highlighted in yellow.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | sums |
| 0 | 2420 | 10 | 1496 | 1246 | 440 | 728 | 1057 | 325 | 1430 | 484 | 9636 |
| 1 | 83 | 3823 | 280 | 408 | 829 | 671 | 450 | 1190 | 192 | 763 | 8689 |
| 2 | 322 | 5 | 326 | 335 | 196 | 197 | 296 | 149 | 343 | 197 | 2366 |
| 3 | 805 | 101 | 1039 | 1037 | 886 | 846 | 982 | 819 | 1047 | 869 | 8431 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 384 | 722 | 874 | 1141 | 1496 | 1190 | 1145 | 1700 | 879 | 1651 | 11182 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 118 | 23 | 162 | 184 | 225 | 163 | 207 | 218 | 172 | 224 | 1696 |
| sums | 4132 | 4684 | 4177 | 4351 | 4072 | 3795 | 4137 | 4401 | 4063 | 4188 | 42000 |

The overall accuracy of the model is 0.2269048.

As an example, this the confusion matrix of the class 1 against all the others.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Image is of class 1** | |
|  |  | True | False |
| **Model predicts** | yes | 3823 | 4866 |
| **class 1.** | no | 861 | 32450 |

|  |  |  |
| --- | --- | --- |
| Accuracy | Precision | Recall |
| 0.8636 | 0.4400 | 0,8162 |

## New feature

To elaborate a new feature given an image, we deal with pixels as if they were points in a cartesian space. We try to find out a way of measuring different distributions of points, since each distribution corresponds to a different digit. Firstly, we associate to each pixel two cartesian coordinates which are, for simplicity sake, the row and column numbers, since the distribution doesn’t depend on the reference system, but only or the reference point around which we are measuring the distribution. Then, we consider only dark points, according to the definition previously given. Then, we select as a reference point the first dark point we encounter in the image’s pixels, i.e. the one with smallest row and column values. Then we compute the mean square distance between this point and all the other dark points. We repeat this procedure for each image, and then we have a new feature we can use in our multinomial model. By picking up this new feature, we must make some decisions between different possibilities, besides the general intuition of considering. We discuss them hereunder.

The reason for filtering out non dark points has already been explained in the section “Exploration of the data”. We hope in this way to filter out some noise in the input data.

The selection of the reference point is crucial in our model. There are some other possible choiches:

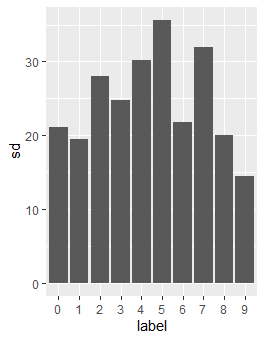
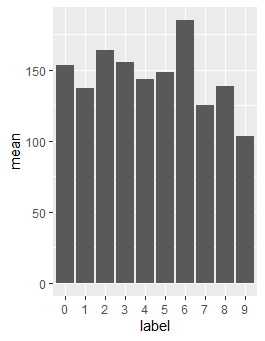
* The point (0,0). We discarde this option since we feel that sometimes the same digit is written in different positions in the cell on the paper sheet, i.e. it could be written more to the left or more to the right, near to the top border or to the bottom border. This would affect the measure of the distance and create problems to our model.
* The point (mean.row, mean.column), where mean row and mean column are computed per each image as the mean on the dark points. This allows to treat in the correct way the digits which are not at the centre of the grid of pixels. Unfortunately, this does not prevent some points from having the same distribution with respect to this reference point,because of some kind of symmetry. For example, the digit “3” and “8” would have the same distribution.
* The first dark point, on the other side, is always located in the same relative position with respect to the others, so it permits a correct measure.

Taking the sum of the squared distances instead of the squared mean values would allow to differentiate more between different classes. However, we avoid that because the number of dark points per each class is not equal (see the exploration of the data). This affects the sum of the distances (if we have more points, we have more addends in the sum), and it’s correlated to the amount of ink each digit costs. Since we already have a measure of the amount of ink (the density), we would have two correlated variables, which is not desirable, especially considering the next section, where we put together the two features.

We square the distance to avoid some sort of “netralization” between two points with high distance and low distance and two points with medium distance.

After having taken the feature, we draw some statistics of this new feature, as done for the density. A positive aspect of this feature is the small values of standard deviation (with respect to the mean, around 10% for all the classes, except for class 5), which is better than the density model ( which was more or less the double).

Nonetheless, class 0, class 3 and class 5 appear to have the same feature, as well as class 1 ad 8.



label mean

1 0 153.2546

2 1 136.9092

3 2 164.0622

4 3 155.5612

5 4 143.2583

6 5 148.1774

7 6 184.8624

8 7 125.0244

9 8 138.2670

10 9 103.5335

label sd

1 0 21.09303

2 1 19.46680

3 2 28.01959

4 3 24.74468

5 4 30.20623

6 5 35.61966

7 6 21.78778

8 7 31.90288

9 8 20.01939

10 9 14.47188

As done before, we scale these values, we train a multinomial logistic model on the data set and we test it on the same data set. We have the following confusion matrix, which unfortunately confirms our negative expectations: classes 0, 4, 5, 8 are never predicted. Classes 6 and 9, which have a distinctive feature value and low sd, are predicted very well. Class 1 is predicted correctly many times, but it’s predicted many times as well (it has relatively high recall, 0,4468, but low precision, 0,2002). Maybe that’s because it has included the predictions for class 0 and 8.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | sums |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1177 | 2093 | 621 | 1319 | 1111 | 1025 | 115 | 720 | 2039 | 235 | 10455 |
| 2 | 697 | 204 | 829 | 591 | 316 | 381 | 775 | 157 | 163 | 8 | 4121 |
| 3 | 1342 | 523 | 931 | 1194 | 524 | 698 | 417 | 289 | 542 | 34 | 6494 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 508 | 248 | 1415 | 811 | 667 | 760 | 2814 | 391 | 183 | 2 | 7799 |
| 7 | 254 | 1361 | 185 | 316 | 883 | 385 | 10 | 923 | 970 | 631 | 5918 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 154 | 255 | 196 | 120 | 571 | 546 | 6 | 1921 | 166 | 3278 | 7213 |
| sums | 4132 | 4684 | 4177 | 4351 | 4072 | 3795 | 4137 | 4401 | 4063 | 4188 | 42000 |

The overall accuracy is 0.2650238.

## Both features

We then repeat the same training and test set using both features and obtain the following results. It improves the performances, but still never predicts class 5, which was never predicted also in the two previous separated models.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | sums |
| 0 | 1911 | 7 | 947 | 818 | 193 | 420 | 438 | 96 | 807 | 76 | 5713 |
| 1 | 79 | 3781 | 230 | 434 | 707 | 486 | 389 | 732 | 184 | 194 | 7216 |
| 2 | 381 | 9 | 478 | 325 | 135 | 184 | 384 | 45 | 94 | 3 | 2038 |
| 3 | 499 | 44 | 496 | 639 | 286 | 368 | 247 | 155 | 315 | 22 | 3071 |
| 4 | 217 | 211 | 273 | 540 | 443 | 388 | 163 | 232 | 349 | 38 | 2854 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 337 | 148 | 1181 | 690 | 634 | 690 | 2470 | 375 | 161 | 3 | 6689 |
| 7 | 97 | 436 | 145 | 340 | 812 | 398 | 10 | 857 | 711 | 484 | 4290 |
| 8 | 448 | 26 | 233 | 450 | 371 | 364 | 33 | 277 | 1227 | 259 | 3688 |
| 9 | 163 | 22 | 194 | 115 | 491 | 497 | 3 | 1632 | 215 | 3109 | 6441 |
| sums | 4132 | 4684 | 4177 | 4351 | 4072 | 3795 | 4137 | 4401 | 4063 | 4188 | 42000 |

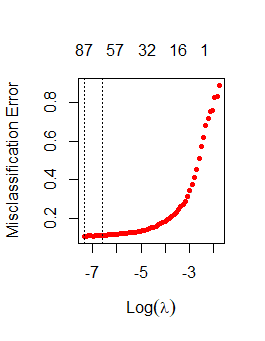
The overall accuracy is 0.355119, which is anyway better.

## Using the pixels as features themselves

In this part we use the pixels as features. First of all, we reduce the number of pixel (and then the level of detail of the image) scaling the image from a grid of 28 x 28 pixels to a grid of 14 x 14 pixels. We recompute the useless pixels in these images and remove them to have faster computation. They are the numbers 2, 3, 4, 5, 9, 10, 12, 14, 15, 16, 29, 30, 44, 58, 72, 114, 128, 142, 156, 170, 184, 185, 197. Unlike the previous experiments, we select a sample of 5000 elements (without replacement) as training set and use the rest as test set. We use the same training set and test set for all the experiments.

### Regularized multinomial logit model (using the LASSO penalty)

We apply 10 folds cross validation on the training set to find out the best lambda value to use in the LASSO penalty. Lambda is simply a coefficient which indicates how much we want to penalize big weights (and so overfitted models) in the multinomial model. Lambda is the only complexity parameter of this model. We obtain the value 0.0006389703. For completeness, this is the plot of lambda log values against classification errors. As expected, increasing lambda improves the performances since it avoids overfitting. This is valid until a critical value, above which the model is too simple and starts performing terribly bad. In our case, this critical value, which is the best value indeed, is very small.



We then test the model with the test set, obtaining the following confusion matrix (where the rows are the predictions of our model, the columns are the actual values).

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | sums |
| 0 | 3455 | 0 | 11 | 8 | 11 | 34 | 38 | 20 | 33 | 32 | 3642 |
| 1 | 2 | 3979 | 45 | 27 | 25 | 42 | 20 | 41 | 113 | 40 | 4334 |
| 2 | 30 | 22 | 3198 | 114 | 26 | 46 | 34 | 79 | 56 | 7 | 3612 |
| 3 | 23 | 22 | 78 | 3342 | 3 | 151 | 2 | 14 | 154 | 62 | 3851 |
| 4 | 8 | 4 | 37 | 8 | 3235 | 58 | 52 | 69 | 30 | 150 | 3651 |
| 5 | 57 | 24 | 21 | 149 | 10 | 2747 | 43 | 6 | 111 | 23 | 3191 |
| 6 | 41 | 8 | 66 | 21 | 36 | 87 | 3426 | 4 | 31 | 7 | 3727 |
| 7 | 6 | 15 | 72 | 59 | 13 | 29 | 13 | 3470 | 18 | 134 | 3829 |
| 8 | 28 | 38 | 76 | 73 | 32 | 119 | 24 | 8 | 2942 | 48 | 3388 |
| 9 | 2 | 3 | 39 | 47 | 184 | 40 | 1 | 182 | 87 | 3190 | 3775 |
| sums | 3652 | 4115 | 3643 | 3848 | 3575 | 3353 | 3653 | 3893 | 3575 | 3693 | 37000 |

The overall accuracy is 0.8914595

### Support Vector Machine

The complexity parameters to determine in this model are gamma and Cost. Gamma is a parameter for nonlinear hyperplanes (the free parameter for a Gaussian radial basis kernel function), and it indicates how the model shapes its decision boundary. Higher values of gamma lead to overfitting. C is the cost function for misclassification error for points which are on the wrong side of the decision boundary. We try different values of gamma and C and pick the best ones to build the model: for gamma all the powers of 10 between -5 and -1, for cost between -3 and 1. We get the following matrix.

gamma cost error dispersion

1 1e-05 1e-03 0.8862 0.01245258

2 1e-04 1e-03 0.8862 0.01245258

3 1e-03 1e-03 0.8862 0.01245258

4 1e-02 1e-03 0.8862 0.01245258

5 1e-01 1e-03 0.8862 0.01245258

6 1e-05 1e-02 0.8862 0.01245258

7 1e-04 1e-02 0.8862 0.01245258

8 1e-03 1e-02 0.8862 0.01245258

9 1e-02 1e-02 0.8862 0.01245258

10 1e-01 1e-02 0.8862 0.01245258

11 1e-05 1e-01 0.8240 0.01720465

12 1e-04 1e-01 0.8862 0.01245258

13 1e-03 1e-01 0.8862 0.01245258

14 1e-02 1e-01 0.8862 0.01245258

15 1e-01 1e-01 0.8862 0.01245258

16 1e-05 1e+00 0.5068 0.04096828

17 1e-04 1e+00 0.8818 0.02294825

18 1e-03 1e+00 0.8862 0.01245258

19 1e-02 1e+00 0.8862 0.01245258

20 1e-01 1e+00 0.8862 0.01245258

21 1e-05 1e+01 0.4590 0.03380007

22 1e-04 1e+01 0.8816 0.02350981

23 1e-03 1e+01 0.8862 0.01245258

24 1e-02 1e+01 0.8862 0.01245258

25 1e-01 1e+01 0.8862 0.01245258

The best one is the 21st. We pick then gamma = 1e-05 and C = 1e+01. We find anyway other useless pixels (they could have become useless due to the sampling of 5000 images in the training set, so that they do not take always value zero in the test set, but they do in the training set. We remove them only for this experiment (we need them for the cross validation of the following experiment). They are the numbers 6, 8, 13, 17, 18, 31, 86, 100, 196. We train a new model with these values. We predict on the test set. We get the following confusion matrix.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | sums |
| 0 | 3252 | 0 | 17 | 16 | 4 | 32 | 62 | 9 | 32 | 26 | 3450 |
| 1 | 3 | 4034 | 164 | 147 | 74 | 202 | 153 | 235 | 263 | 132 | 5407 |
| 2 | 27 | 13 | 2965 | 119 | 15 | 32 | 26 | 18 | 50 | 7 | 3272 |
| 3 | 50 | 16 | 84 | 3249 | 0 | 579 | 8 | 13 | 246 | 53 | 4298 |
| 4 | 22 | 3 | 108 | 12 | 2940 | 84 | 66 | 102 | 30 | 196 | 3563 |
| 5 | 123 | 5 | 5 | 30 | 20 | 2109 | 72 | 1 | 108 | 5 | 2478 |
| 6 | 95 | 7 | 98 | 13 | 19 | 56 | 3258 | 2 | 15 | 0 | 3563 |
| 7 | 2 | 2 | 54 | 36 | 1 | 10 | 1 | 3115 | 10 | 178 | 3409 |
| 8 | 61 | 34 | 116 | 146 | 22 | 110 | 7 | 71 | 2657 | 68 | 3292 |
| 9 | 17 | 1 | 32 | 80 | 480 | 139 | 0 | 327 | 164 | 3028 | 4268 |
| sums | 3652 | 4115 | 3643 | 3848 | 3575 | 3353 | 3653 | 3893 | 3575 | 3693 | 37000 |

The overall accuracy is 0.8272162.

### Feed forward neural networks

The two parameters of this model are size and decay. Size is the number of units in hidden layer and decay is the regularization parameter to avoid over-fitting. We do 10 folds cross validation to compute the right values of these parameters. We try values from 1 to 10 for size and from 0.1 to 0.5 for decay. We find that size = 10 and decay = 0.4 are the best models.

We get the following confusion matrix.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | sums |
| 0 | 3438 | 0 | 19 | 17 | 9 | 46 | 26 | 13 | 21 | 27 | 3616 |
| 1 | 0 | 3979 | 56 | 34 | 26 | 15 | 12 | 46 | 86 | 17 | 4271 |
| 2 | 39 | 29 | 3228 | 101 | 19 | 41 | 35 | 56 | 44 | 7 | 3599 |
| 3 | 21 | 17 | 97 | 3378 | 2 | 145 | 1 | 14 | 113 | 40 | 3828 |
| 4 | 14 | 3 | 50 | 3 | 3296 | 40 | 88 | 77 | 30 | 169 | 3770 |
| 5 | 61 | 19 | 24 | 139 | 14 | 2795 | 48 | 11 | 89 | 31 | 3231 |
| 6 | 41 | 17 | 39 | 17 | 59 | 91 | 3407 | 2 | 39 | 12 | 3724 |
| 7 | 11 | 10 | 67 | 45 | 5 | 39 | 1 | 3533 | 13 | 83 | 3807 |
| 8 | 20 | 37 | 48 | 63 | 25 | 84 | 35 | 17 | 3083 | 44 | 3456 |
| 9 | 7 | 4 | 15 | 51 | 120 | 57 | 0 | 124 | 57 | 3263 | 3698 |
| sums | 3652 | 4115 | 3643 | 3848 | 3575 | 3353 | 3653 | 3893 | 3575 | 3693 | 37000 |

The overall accuracy is 0.9027027.

## Discussion of the differences

The feed forward neural network is the best model, according to the computed accuracies.

To find out in a rigorous way whether the differences between the three models are statistically significant a McNemar test has been performed between the three models, splitting the predictions in correct ones and incorrect ones. We order the models according to their accuracy and we test each model against the ones with better accuracy. The McNemar test performs a chi-squared test to indicate if there is evidence that marginal probabilities of the table are the same (the null ipothesis)

### Svm vs Multinomial

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Multinomial |  |
|  |  | Correct | Incorrect |
| Svm | Correct | 29581 | 1026 |
|  | Incorrect | 3403 | 2990 |

McNemar's chi-squared = 1274.6, df = 1, p-value < 2.2e-16

We refuse the null ipothesis (the differences are not significant) with an error probability near 0. We conclude that the multinomial model is significantly better.

### Svm vs Neural Network

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Neural | Networks |
|  |  | Correct | Incorrect |
| svm | Correct | 29475 | 1132 |
|  | Incorrect | 3925 | 2468 |

McNemar's chi-squared = 1541.5, df = 1, p-value < 2.2e-16

We refuse the null ipothesis (the differences are not significant) with an error probability near o. We conclude that the neural network model is significantly better.

### Multinomial vs Neural Networks

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Neural | Netwokrs |
|  |  | Correct | Incorrect |
| Multinomial | Correct | 31894 | 1090 |
|  | Incorrect | 1506 | 2510 |

McNemar's chi-squared = 66.342, df = 1, p-value = 3.79e-16

We refuse the null ipothesis (the differences are not significant) with an error probability near 0. We conclude that the neural network model is significantly better ( even if the accuracies are comparable)

## Appendix 1: Set up code

This code installs all the packages needed for the experiments and calls all the libraries. Then creates the dataframe in the environment, and changes the class values as factors, to handle them #-----------------install packages----------------------#

install.packages("OpenImageR")

install.packages("ggplot2")

install.packages("ggpubr")

install.packages("dplyr")

install.packages("nnet")

install.packages("raster")

install.packages("e1071")

install.packages("caTools")

install.packages("glmnet")

install.packages("caret")

#-----------------libraries----------------------#

library( OpenImageR)

library(ggplot2)

library(ggpubr)

library(dplyr)

theme\_set(theme\_pubr())

library(nnet)

library(raster)

library(e1071)

library(caTools)

library(glmnet)

library(caret)

#----------------set up code) ----------------------#

mnist.dat <- read.csv("C:/Users/ponti/Documents/GitHub/DataMining/pattern recognition/mnist.csv")

mnist.dat$label<-factor(mnist.dat$label) #Converting label to factor

## Appendix 2: Code for the exploration of the data

This code is used to draw the graphs and extract the results contained in the section Exploration of the data.

#-----------------class distribution----------------------#

ggplot(data=mnist.dat, aes(label)) +geom\_bar(fill = "#0073C2FF") +theme\_pubclean()

frequency.dataframe <- mnist.dat %>%

group\_by(label) %>%

summarise(counts = n())

frequency.dataframe

ggplot(frequency.dataframe, aes(x = label, y = counts)) +

geom\_bar(fill = "#0073C2FF", stat = "identity") +

geom\_text(aes(label = counts), vjust = -0.3) +

theme\_pubclean() # histrogram with quantity of the bars

frequency.dataframe <- frequency.dataframe %>%

arrange(desc(label)) %>%

mutate(prop = round(counts\*100/sum(counts), 1),

lab.ypos = cumsum(prop) - 0.5\*prop)

frequency.dataframe

ggpie(frequency.dataframe, x = "prop", label = "prop",

lab.pos = "in", lab.font = list(color = "white"),

fill = "label", color = "white",

palette = "jco"

) #pie chart

#-----------------summary statistics----------------------#

pixel.minValue <- apply(mnist.dat[,-1], 2, min) #2 means by column

pixel.maxValue <- apply(mnist.dat[,-1], 2, max) #1 means by row

pixel.avgValue <- apply(mnist.dat[,-1], 2, mean)

pixel.nwhite <- apply(mnist.dat[,-1], 2, function(pixel){

return (length(pixel[pixel ==0]))

})

pixel.nblack <- apply(mnist.dat[,-1], 2, function(pixel){

return (length(pixel[pixel ==255]))

})

mean.minValue <- mean(pixel.minValue)

mean.maxValue <- mean( pixel.maxValue)

mean.avgValue <- mean(pixel.avgValue)

mean.nwhite <- mean(pixel.nwhite)

mean.nblack <- mean(pixel.nblack)

summary <- cbind(pixel.minValue, pixel.maxValue, pixel.avgValue, pixel.nwhite, pixel.nblack)

summary <- as.data.frame(summary)

useless.pixels <- summary[summary$pixel.minValue == summary$pixel.maxValue, ]

useless.pixels.name <- row.names(useless.pixels)

bottom.pixels <- summary[summary$pixel.minValue == 0, ]

top.pixels <- summary[summary$pixel.maxValue == 255, ]

num.bottom.pix <- nrow(bottom.pixels)

num.top.pix <- nrow(top.pixels)

## Appendix 3: code for black points

#-----------------analysis of the dark points----------------------#

mnist.dat$darks <- apply(mnist.dat[,c(2:785)], 1, function( sample){

return (length(sample[sample > 128]))

} )

blacks.mean <- aggregate(mnist.dat$blacks,by=list(mnist.dat$label),FUN=mean)

colnames(blacks.mean) <- c("label", "mean")

ggplot(data=blacks.mean, aes(x=label,y=mean))+geom\_bar(stat="identity") #Depicting average number of black points.

imageShow(matrix(as.numeric(mnist.dat[380,-c(1,786)]),nrow=28,ncol=28,byrow=T)) #image with no filter

#verifying the filter

sample.image <- as.numeric(mnist.dat[380,-c(1,786)])

sample.image <- sapply(sample.image, function(pixel){

if (pixel > 128){

return (pixel)

}

else return (0)

})

imageShow(matrix(sample.image,nrow=28,ncol=28,byrow=T)) #image with no filter

## Appendix 4:code for “how much ink”

#-----------------how much ink ("density") ( mean and standard deviation)----------------------#

mnist.dat$density <- apply(mnist.dat[,c(2:785)], 1, sum)

density.mean <- aggregate(mnist.dat$density,by=list(mnist.dat$label),FUN=mean)

colnames(density.mean) <- c("label", "mean")

print(density.mean)

ggplot(data=density.mean, aes(x=label,y=mean))+geom\_bar(stat="identity") #Depicting average intensity.

#standard deviation

density.sd <- aggregate(mnist.dat$density,by=list(mnist.dat$label),FUN=sd)

print(density.sd)

colnames(density.sd) <- c("label", "sd")

ggplot(data=density.sd, aes(x=label,y=sd))+geom\_bar(stat="identity") #Depicting standard deviation.

#-----------------simple multinomial model ("density")----------------------#

mnist.dat$density <- as.vector(scale(mnist.dat$density))

input <- as.data.frame(mnist.dat$density, row.names = NULL, optional = TRUE)

colnames(input) <- "density"

multinom.model<-multinom(label~density,data = mnist.dat, maxit = 1000)

summary(multinom.model)

multinom.pred <- predict(multinom.model, input, type = "class")

multinom.conf.mat <- table(multinom.pred,mnist.dat$label) #confusion matrix

multinom.accuracy <- sum(diag(multinom.conf.mat))/sum(multinom.conf.mat)

print (multinom.conf.mat)

print(multinom.accuracy)

## Appendix 5: code for new feature

#-----------------new feature multinomial model ( distance of the dark points)----------------------#

mnist.dat$mean.distance <- apply(mnist.dat[,c(2:785)],1, function(current.example){

current.example <- unlist(current.example)

current.example <- as.vector(current.example)

index.row <- lapply(c(0:27), function(row){

rep(row, 28)

})

index.row <- unlist(index.row, recursive = FALSE)

index.column <- lapply(c(0:27), function(column){

c(0:27)

})

index.column <- unlist(index.column, recursive = FALSE)

current.example.with.indexes <- cbind(current.example, index.row, index.column)

current.example.points <- current.example.with.indexes[current.example.with.indexes[,1] > 128, ]

mean.row <- current.example.points[1,2]

mean.column <- current.example.points[1,3]

point.distances <- lapply(c(1:nrow(current.example.points)), function(point){

example <- current.example.points[point,]

return ( ((pointDistance(current.example.points[point,-1], cbind(mean.row,mean.column), lonlat = FALSE))^2))

})

point.distances <- unlist (point.distances, recursive = FALSE)

return (mean(point.distances))

})

#average value

mean.distance.mean <- aggregate(mnist.dat$mean.distance,by=list(mnist.dat$label),FUN=mean)

colnames(mean.distance.mean) <- c("label", "mean")

print(mean.distance.mean)

ggplot(mean.distance.mean, aes(x=label,y=mean))+geom\_bar(stat="identity") #Depicting average distance to the center.

#standard deviation

mean.distance.sd <- aggregate(mnist.dat$mean.distance,by=list(mnist.dat$label),FUN=sd)

colnames(mean.distance.sd) <- c("label", "sd")

print(mean.distance.sd)

ggplot(data=mean.distance.sd, aes(x=label,y=sd))+geom\_bar(stat="identity") #Depicting standard deviation.

#-----------------simple multinomial model ( with the second feature)----------------------#

mnist.dat$mean.distance <- as.vector(scale(mnist.dat$mean.distance))

input <- as.data.frame(mnist.dat$mean.distance, row.names = NULL, optional = TRUE)

colnames(input) <- "mean.distance"

multinom.model<-multinom(label~mean.distance,data = mnist.dat, maxit = 1000)

summary(multinom.model)

multinom.pred <- predict(multinom.model, input, type = "class")

multinom.conf.mat <- table(multinom.pred,mnist.dat$label) #confusion matrix

multinom.accuracy <- sum(diag(multinom.conf.mat))/sum(multinom.conf.mat)

print (multinom.conf.mat)

print(multinom.accuracy)

## Appendix 6: code for both features

This code is used to train a multinomial logistic model and test it on the same dataset used. It prints on the console a summary of the model, the confusion matrix between the predicted values and the actual values, and the overall accuracy.

#-----------------multinomial model with both features----------------------#

data.training <- mnist.dat[, cbind("label", "mean.distance", "density")]

multinom.model<-multinom(label~.,data = data.training, maxit = 1000)

summary(multinom.model)

multinom.pred <- predict(multinom.model, data.training[,-1], type = "class")

multinom.conf.mat <- table(multinom.pred,data.training$label) #confusion matrix

multinom.accuracy <- sum(diag(multinom.conf.mat))/sum(multinom.conf.mat)

print (multinom.conf.mat)

print(multinom.accuracy)

## Appendix 7: set up code for the last 3 models

#reducing the dimensions of the image by half

mnist.dat.reduced<- apply(mnist.dat[ , -1], 1, function(image){

image.matrix <- matrix(image,nrow = 28, ncol = 28, byrow = TRUE)

image.reduced <- down\_sample\_image(image.matrix, 2, gaussian\_blur = FALSE)

return (image.reduced)

})

mnist.dat.reduced <- t(mnist.dat.reduced)

mnist.dat.reduced <- cbind(mnist.dat[,1], mnist.dat.reduced)

mnist.dat.reduced <- as.data.frame(mnist.dat.reduced)

mnist.dat.reduced[,1]<-factor(mnist.dat.reduced[,1])

pixel.minValue <- apply(mnist.dat.reduced[,-1], 2, min) #2 means by column

pixel.maxValue <- apply(mnist.dat.reduced[,-1], 2, max) #1 means by row

summary <- cbind(pixel.minValue, pixel.maxValue)

summary <- as.data.frame(summary)

useless.pixels <- summary[summary$pixel.minValue == summary$pixel.maxValue, ]

mnist.dat.reduced <- mnist.dat.reduced[,-c(2,3,4,5,9,10,12,14,15,16,29,30,44,58,72,114,128,142,156,170,184,185,197)]

#drawing 500 random samples

training.rows<-sample(nrow(mnist.dat.reduced),5000)

training.set<-mnist.dat.reduced[training.rows,]

#using the remaining samples as a test set

test.set <-mnist.dat.reduced[-training.rows,]

## Appendix 8: code for multinomial model

#-----------------the regularized multinomial logit model (using the LASSO penalty)----------------------#

lasso.model<-cv.glmnet(as.matrix(training.set[,-1]),training.set[,1],family="multinomial",type.measure ="class")

print(lasso.model$lambda.min) #Best lambda value 0.001246766

plot(lasso.model)

lasso.pred<-predict(lasso.model,as.matrix(test.set[,-1]),s="lambda.min", type="class")

lasso.confmat<- table(lasso.pred,test.set[,1])

lasso.accuracy <- sum(diag(lasso.confmat))/sum(lasso.confmat)

print(lasso.confmat)

print(lasso.accuracy)

## Appendix 9: code for support vector machine model

#---------------------------------------svm--------------------------------------------------

tuned.parameters <- tune.svm(training.set[,-1],training.set[,1],gamma = 10^(-5:-1), cost = 10^(-3:1))

tuned.parameters$performances

summary (tuned.parameters)

#training.set <- training.set[, -c(6,8,13,17,18,31,86,100,196)]

#test.set <- test.set[, -c(6,8,13,17,18,31,86,100,196)]

training.set <- training.set[, -c(2,4,6,7,8,19,71,85,174)]

test.set <- test.set[, -c(2,4,6,7,8,19,71,85,174)]

mnist.svm.tuned <- svm(training.set[, -1],training.set[,1],cost = 1e+01, gamma = 1e-05)

mnist.svm.pred <- predict(mnist.svm.tuned,test.set[,-1])

mnist.svm.confmat <- table(mnist.svm.pred, test.set[,1])

mnist.svm.accuracy <- sum(diag(mnist.svm.confmat)/sum(mnist.svm.confmat))

## Appendix 10: code for neural network model

#-----------------feed forward neural network----------------------#

x.train\_labels<-mnist.dat.reduced[training.rows,1]

x.test\_labels<-mnist.dat.reduced[-training.rows,1]

x.train\_labels<-as.factor(x.train\_labels)

x.test\_labels<-as.factor(x.test\_labels)

train\_norm <- as.data.frame(lapply(mnist.dat.reduced[training.rows, -1], function(x) {

return(x / 255)

}))

test\_norm <- as.data.frame(lapply(mnist.dat.reduced[-training.rows,-1], function(x) {

return(x / 255)

}))

train\_labels\_matrix = class.ind(x.train\_labels)

head(x.train\_labels)

TrainingParameters <- trainControl(method = "cv", number = 10)

grid\_nn <- expand.grid(size = seq(from = 1, to = 10, by = 1),

decay = seq(from = 0.1, to = 0.5, by = 0.1))

nn <- train(train\_norm, x.train\_labels,

trControl= TrainingParameters,

method = "nnet",

tuneGrid = grid\_nn,

MaxNWts = 20000

)

nn

NNPredictions <-predict(nn, test\_norm)

mnist.dat.nnet.confmat<- table(NNPredictions,x.test\_labels)

sum(diag(mnist.dat.nnet.confmat))/sum(mnist.dat.nnet.confmat)

## Appendix 11: code for McNemar test

#-----------------Mc Nemar test----------------------#

NNPredictions.is.correct <- sapply(c(1:length(NNPredictions)),function(index){

if(NNPredictions[index] == x.test\_labels[index]){

return (1)

}

else (return (0))

})

lasso.pred.is.correct <- sapply(c(1:length(lasso.pred)),function(index){

if(lasso.pred[index] == x.test\_labels[index]){

return (1)

}

else (return (0))

})

mnist.svm.pred.is.correct <- sapply(c(1:length(mnist.svm.pred)),function(index){

if(mnist.svm.pred[index] == x.test\_labels[index]){

return (1)

}

else (return (0))

})

svm.vs.multinomial.conf.matrix <- table(mnist.svm.pred.is.correct, lasso.pred.is.correct)

mcnemar.test(svm.vs.multinomial.conf.matrix)

svm.vs.nn.conf.matrix <- table(mnist.svm.pred.is.correct, NNPredictions.is.correct)

mcnemar.test(svm.vs.nn.conf.matrix)

multinomial.vs.nn.conf.matrix <- table(lasso.pred.is.correct, NNPredictions.is.correct)

mcnemar.test(multinomial.vs.nn.conf.matrix)