Introduction to Machine Learning- report
Tuuli Kauppala 014082616

August 2, 2019

Task 1

In task 1) we implemented the perceptron algoritm to train a linear classsifier that we applied to the MNIST dataset of handwritten digits. Perceptron algorithm is a linear classifier. This means that a linear relationship between features and labels is assumed, and the prediction is obtained by a linear function that has a weight combination for the feature vector. Perceptron algorithm is a binary classifier with features having either of two possible labels. In our case, the labels are +1 and -1.

First, we implemented the algorithm described in Introduction to Machine learning slides, and obtained an algorithm that we tested with small (N=4) but linearly separable, and small (N=4) but linearly unseparable two-class dataset in two dimensions. If the separator works, it will separate the linearly separable datasets, and if it doesn't work, it will fail to separate the datasets.

Our algorithm is called **korjaus()**. The argument it takes is the matrix containing the dataset with the last column being the labels. We tested its output by a method called **toimiiko_uusi()**. **Korjaus()** gives the updated linear boundary between the two classes, and if the algorithm fails to find such boundary within n iterations, then korjaus() returns message telling that the conversion is not reached.

Here are the 5 datasets and their perceptron output: the first two colums give dots on the x-y axis and the third column represents their labels.

Data1 (linearly separable)

```
     [,1] [,2] [,3]
[1,]    1    1   -1
[2,]    2    2   -1
[3,]   -1   -1    1
[4,]   -1    2    1

> print(korjaus(data1))
[1] 0
[1] -1 -5 1
> toimiiko_uusi(data1)
[1] 0
[1] "ennuste onnistui!"
```

We see that the error sum is 0, the weight vector is (-1,5,1), and that the prediction given by the algorithm corresponds to the given labels.

Data2 (linearly separable)

```
> data2
     [,1] [,2] [,3]
[1,]    1   -1    1
[2,]    2   -1    1
[3,]    1    1   -1
[4,]    1    2   -1

> print(korjaus(data2))
[1] 0
[1]  1  1 -3

> toimiiko_uusi(data2)
[1] 0
[1] "ennuste onnistui!"
```

Data3 (linearly separable)

```
> data3
     [,1] [,2] [,3]
[1,]   -1    2    1
[2,]    2    2    1
[3,]   -1   -1   -1
[4,]    0    0   -1

> print(korjaus(data3))
[1] 0
[1] -1 1  1
> toimiiko_uusi(data3)
[1] 0
[1] "ennuste onnistui!"
```

Data4 (linearly separable)

```
> data4
     [,1] [,2] [,3]
[1,]   -1    1    1
[2,]    2    2    1
[3,]   -1   -1   -1
[4,]    1    1   -1
> print(korjaus(data4))
[1] 0
[1] -3 -1 3
> toimiiko_uusi(data4)
[1] 0
[1] "ennuste onnistui!"
>
```

Figure 1 shows a representation of the separator hyperplane dividing the data points of data4.

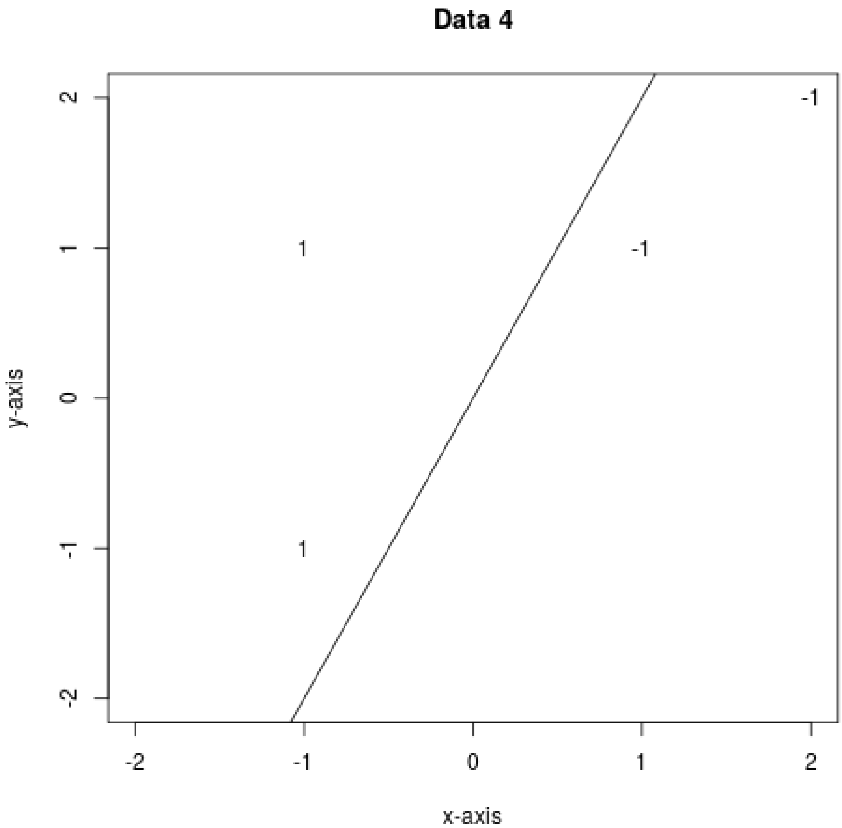

Figure 1: Example illustration of a small linearly separable dataset (Data 4). The line represents the result of the separation.

Data5 (linearly unseparable)

```
> data5
     [,1] [,2] [,3]
[1,]   -1   -1    1
[2,]   -1    1   -1
[3,]   -1    0    1
[4,]   -1   -2   -1
```

Plot of the data 5 is shown in figure 2.

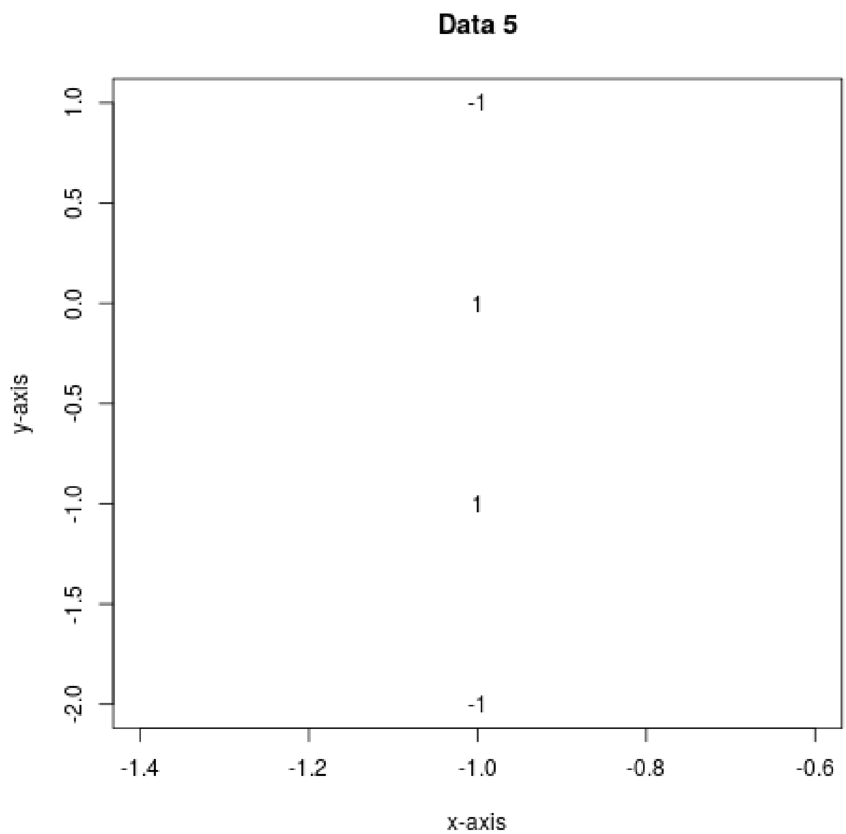

Figure 2: Example illustration of a small linearly unseparable dataset (Data 5)

```
> korjaus(data5)
[1] "not reched conversion in 1000 epochs! The weight vector is:"
[2] "-1"
[3] "3"
[4] "3"
```

Data6 (linearly unseparable)

```
> data6
     [,1] [,2] [,3]
[1,]   -1   -1    1
[2,]    0    0   -1
[3,]    1    1    1
[4,]    2    2   -1

> korjaus(data6)
[1] "not reched conversion in 1000 epochs! The weight vector is:"
[2] "1"
[3] "-1"
[4] "-1"
```

Next, we loaded first 5000 MNIST digits, and we took the first 2500 digits as the training set, and the rest as a test set. We selected only digits in the trainings set that are either zeros or ones, then we implemented our perceptron algorithm to the data. The **korjaus()**-method was slightly changed to print 1) the output vector 2) n. of epochs for this task.

The output of **mnist_painot-korjaus(mnist_digits)**:

```
> mnist_painot
[[1]]
  [1]    7    1    1    1    1    1    1    1    1    1    1    1
          1    1    1    1    1    1
 [19]    1    1    1    1    1    1    1    1    1    1    1    1
         1    1    1    1    1    1
 [37]    1    1    1    1    1    1    1    1    1    1    1    1
         1    1    1    1    1    1
 [55]    1    1    1    1    1    1    1    1    1    1    1    1
         1    1    1    1    1    1
 [73]    1    1    1    1    1    1    1    1    1    1    1    1
         1    1    1    1    1    1
 [91]    1    1    1    1    1    1   59  257  311  385  841  545
     111    1    1    1    1    1
[109]    1    1    1    1    1    1    1    1    1    1    1    1
        3   17   35  223  447  543
[127]  -17 -827 -265  247  677  373  419 -429 -195  -83    1    1
        1    1    1    1    1    1
[145]    1    1    1    1   23  411  457  431  295   87  381  229
     109 -125   57  203   17 -359
[163] -657 -283    1    1    1    1    1    1    1    1    1    1
        1    1   21  493  509   37
[181]  -69   13  191 -321  361 -159 -607  555  191  -31 -355  -99
     11    1    1    1    1    1
```

```
[199]    1    1    1    1    1    1    1   97  -55 -455 -523   11
    -241 -311  147  -99 -911  679
[217]   -7   -1 -123 -145   41    1    1    1    1    1    1    1
       1    1    1    1    1 -213
[235] -503 -499 -579 -239  257 -137 -149 -785 -971  479  -13    7
    -745 -423    1    1    1    1
[253]    1    1    1    1    1    1    1    1    1 -339 -413 -167
    -405  211  271  369  149 -451
[271] -285  321   29 -831 -1405 -737    1    1    1    1    1    1
       1    1    1    1    1    1
[289]  -97 -195 -103 -397 -417 -473   35  587 1099  951  443  405
    -383 -1505 -1845 -599    1    1
[307]  211    1    1    1    1    1    1    1    1   37   75  125
     -23  249 -625 -1135  125  327
[325] 1889 1625  481 -375 -735 -1571 -1889 -797  -37    1  309    1
       1    1    1    1    1    1
[343]   13  253  459    7 -309 -125 -561 -819  419  851 2427 2045
      -3  -279  -563 -1821 -1677 -893
[361] -359  -25    1    1    1    1    1    1    1    1  129  339
    -249 -891  -867 -1035 -1511 -483
[379] 1153 2777 2775 1587 -639 -437 -563 -1409 -1169 -893 -503 -223
       1    1    1    1    1    1
[397]    1    1  217 -257 -871 -985 -651 -1611 -1125 -137 1537 3425
    2593  365 -1335 -991 -919 -1059
[415] -831 -897 -505 -223    1    1    1    1    1    1    1    1
     -75  -619  -995  -855 -1295 -1705
[433] -957  273 2327 3555 2049 -117 -1477 -375 -663 -845 -673 -799
    -503 -315    1    1    1    1
[451]    1    1    1    1 -225 -673 -1039 -947 -1127 -1927 -577 757
    3367 3177 1163  -625 -1529 -437
[469] -779 -733 -409 -441 -503 -503    1    1    1    1    1    1
       1    1 -413 -705 -1347 -885
[487] -337 -1233 -285 1057 2511 1949  519 -519 -1793 -1515 -1013 -507
    -161 -405 -503 -471    1    1
[505]    1    1    1    1    1    1 -505 -641 -1149 -705 -425 -129
     143  385 2547 1421  277 -755
[523] -1637 -1339 -289 661  345 -143 -223 -147    1    1    1    1
       1    1    1    1 -227 -345
[541] -1171 -1169 101  789  315  351 2275 1767  417 -899  -29  475
     817  861  465    3   29  325
[559]   59    1    1    1    1    1    1    1 -149 -497 -925 -685
    1095  697  307  211  295  865
[577] -375 -585  659 1077  529  293   91  -21   59  495  109    1
       1    1    1    1    1    1
[595] -249 -189 -169  381  995    5 -535 -979 -881   19   87  575
     667   45 -285 -365 -309 -311
[613]  -37   99    5    1    1    1    1    1    1    1 -401 -691
       1  637  791 -193 -739 -2131
[631] -1875 -727  169  857  717 -383 -503 -449 -141    1    1    1
       1    1    1    1    1    1
```

```
[649]    1     1  -627  -987  -611  -163    83  -553  -917 -1691 -1419 -351
    275   983   759  -165  -223     1
[667]    1     1     1     1     1     1     1     1     1     1     1   -57
   -509  -509  -509  -461  -123    -3
[685]  -11  -281   337   501   435   185   521    79     1     1     1     1
      1     1     1     1     1     1
[703]    1     1     1     1     1     1     1     1     1     1     1     1
      1     1     1     1     1     1
[721]    1     1     1     1     1     1     1     1     1     1     1     1
      1     1     1     1     1     1
[739]    1     1     1     1     1     1     1     1     1     1     1     1
      1     1     1     1     1     1
[757]    1     1     1     1     1     1     1     1     1     1     1     1
      1     1     1     1     1     1
[775]    1     1     1     1     1     1     1     1     1     1     1

[[2]]
[1] 5
```

We see that the algorithm converged in 5 epochs.

We tested the algorithm's output by using **toimiiko_uusi()** - method:

```
> toimiiko_uusi(testdata_labels_01, mnist_painot[[1]])
[1] "laskuri on:" "1"
```

From the print above we can conclude that only one of 2500 predictions did not succeed. The rate seems surprisingly low, maybe something went wrong here. At the same time, if the data is easily enough separable, why would the separator not find the boundary quickly?

Finally, we plotted the pixel weights as an image (figure 3).

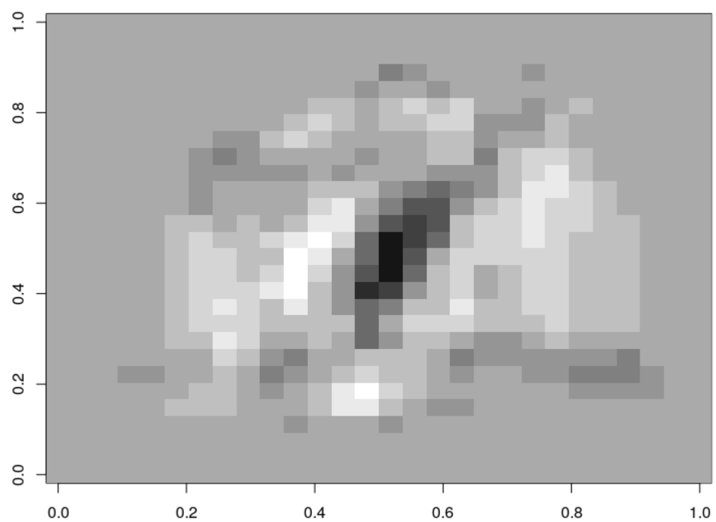

Figure 3: Pixel weights image

The pixel weights image resembles more 0 than 1, which is probably because in the middle of the "hole" there are positive, "black" values and around the "hole" in a "circle" there are negative values. The positive values probably may contribute to the recognition on "1", and the negative values could contribute to the recognition of "0". Around the center there is the gray area with the default values, as the periphery does not affect the digit recognition.

Task 2

Here our dataset is a 20 newsgroup dataset that contains information on different documents, their words and the newsgroup they belong to. The dataset also tells the amounts each word appears in the document, but we will not pay attention to that.

Our task is to divide our data into train data and test data, then teach a Naive Bayes classifier to classify the documents in the train data, and later apply the classifier to the test data. In order to achieve that we first divided the data in a following manner: 90Next, we made our Naive Bayes model that contained following information as the output: 1) priors for each newsgroup, 2) posterior probabilities for each word, given the newsgroup. The method in the code is called "nb()". We checked that our model made sense by looking at the words with highest probabilities in each group and found out that top 10 words are always “stopwords”: the, and, or, from, to... Below are all the unique top 10 most probable words of all 20 newsgroups:

```
> unique(unlist(top_words))
 [1] "this" "on" "it"  "for" "is"  "and" "in"  "of"
"to"  "have" "you" "that" "be" "with"

> unique(unlist(top_words))
 [1] "the"    "of"     "in"     "to"     "and"    "that"   "is"
  "writes" "it"     "you"    "for"    "on"     "windows" "with"
[15] "this"   "or"     "have"   "edu"    "not"    "be"
```

As our model made sense, we did the Naive Bayes inference. We made the “Naive Bayes” assumption, where all the document’s words are conditionally independent from the newsgroup , which is, as the name tells, a dubious assumption. Naive Bayes method is told to be quite robust even when the assumptions are violated . We are able to test this by using our Naive Bayes model of 20 newsgroup data to find the posterior probabilities for each newsgroup, given the document.

Here we assume that a document is a collection of words it contains as well as words it does not contain. Hence, document’s probability to belong to each newsgroup is the intersection of all probabilities of words given the newsgroup times the newsgroup’s prior: P(N—D)=P(D—N)*P(N)=P(W1 AND W2 NOT W3 . . . . — N)*P(N)=P(W1—N)*P(W2—N)*(1-P(W3 —N))*P(N). Here N=newsgroup, W=word and D=document.

In our program, the method “ennusta()” gives predictions for each group, when document is given. The method valitsemaksimi() gives us the group for which the prediction likelihood based on the document ID was the biggest. We ran the code first with the whole training data (lasted for 10 hours). Below is the confusion matrix: CONFUSION MATRIX BELOW.

```
> train_results
       traininglabels
maksimit           1           2           3           4           5
      1  0.972222222 0.000000000 0.001941748 0.000000000 0.000000000
      2  0.000000000 0.904397706 0.005825243 0.005681818 0.005791506
      3  0.000000000 0.019120459 0.937864078 0.001893939 0.001930502
      4  0.000000000 0.030592734 0.031067961 0.969696970 0.007722008
      5  0.000000000 0.001912046 0.001941748 0.003787879 0.961389961
      6  0.000000000 0.015296367 0.007766990 0.003787879 0.000000000
      7  0.000000000 0.005736138 0.001941748 0.005681818 0.003861004
      8  0.000000000 0.001912046 0.000000000 0.000000000 0.000000000
      9  0.000000000 0.000000000 0.000000000 0.000000000 0.001930502
      10 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
      11 0.000000000 0.001912046 0.000000000 0.000000000 0.000000000
      12 0.002314815 0.007648184 0.003883495 0.000000000 0.003861004
      13 0.000000000 0.001912046 0.001941748 0.001893939 0.000000000
      14 0.000000000 0.000000000 0.000000000 0.000000000 0.003861004
```

```
      15 0.000000000 0.007648184 0.001941748 0.001893939 0.001930502
      16 0.023148148 0.001912046 0.003883495 0.001893939 0.005791506
      17 0.000000000 0.000000000 0.000000000 0.003787879 0.000000000
      18 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
      19 0.002314815 0.000000000 0.000000000 0.000000000 0.001930502
      20 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
        traininglabels
maksimit           6           7           8           9          10
      1  0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
      2  0.003752345 0.003816794 0.000000000 0.001862197 0.001869159
      3  0.016885553 0.007633588 0.000000000 0.000000000 0.000000000
      4  0.001876173 0.053435115 0.001876173 0.000000000 0.001869159
      5  0.003752345 0.009541985 0.000000000 0.003724395 0.000000000
      6  0.960600375 0.000000000 0.005628518 0.000000000 0.001869159
      7  0.000000000 0.816793893 0.001876173 0.007448790 0.001869159
      8  0.000000000 0.024809160 0.975609756 0.001862197 0.003738318
      9  0.001876173 0.001908397 0.000000000 0.973929236 0.000000000
      10 0.000000000 0.003816794 0.001876173 0.000000000 0.977570093
      11 0.000000000 0.005725191 0.000000000 0.000000000 0.009345794
      12 0.001876173 0.020992366 0.001876173 0.000000000 0.000000000
      13 0.000000000 0.022900763 0.003752345 0.000000000 0.000000000
      14 0.000000000 0.001908397 0.000000000 0.003724395 0.000000000
      15 0.003752345 0.001908397 0.001876173 0.000000000 0.000000000
      16 0.003752345 0.007633588 0.000000000 0.003724395 0.001869159
      17 0.001876173 0.011450382 0.005628518 0.003724395 0.000000000
      18 0.000000000 0.001908397 0.000000000 0.000000000 0.000000000
      19 0.000000000 0.003816794 0.000000000 0.000000000 0.000000000
      20 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
        traininglabels
maksimit          11          12          13          14          15
      1  0.000000000 0.000000000 0.000000000 0.000000000 0.001872659
      2  0.000000000 0.001869159 0.005639098 0.003738318 0.001872659
      3  0.001858736 0.000000000 0.001879699 0.000000000 0.000000000
      4  0.003717472 0.000000000 0.026315789 0.000000000 0.003745318
      5  0.000000000 0.000000000 0.001879699 0.000000000 0.000000000
      6  0.000000000 0.000000000 0.000000000 0.001869159 0.001872659
      7  0.000000000 0.000000000 0.003759398 0.000000000 0.000000000
      8  0.001858736 0.000000000 0.001879699 0.001869159 0.001872659
      9  0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
      10 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
      11 0.979553903 0.000000000 0.000000000 0.000000000 0.000000000
      12 0.001858736 0.990654206 0.005639098 0.000000000 0.000000000
      13 0.001858736 0.000000000 0.939849624 0.003738318 0.001872659
      14 0.000000000 0.001869159 0.000000000 0.979439252 0.000000000
      15 0.000000000 0.000000000 0.001879699 0.000000000 0.985018727
      16 0.001858736 0.000000000 0.005639098 0.005607477 0.001872659
      17 0.001858736 0.001869159 0.005639098 0.003738318 0.000000000
      18 0.000000000 0.001869159 0.000000000 0.000000000 0.000000000
      19 0.005576208 0.001869159 0.000000000 0.000000000 0.000000000
      20 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
```

```
       traininglabels
maksimit          16          17          18          19          20
      1  0.000000000 0.000000000 0.000000000 0.004784689 0.076696165
      2  0.000000000 0.000000000 0.003937008 0.004784689 0.002949853
      3  0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
      4  0.003710575 0.000000000 0.000000000 0.002392344 0.000000000
      5  0.001855288 0.000000000 0.000000000 0.000000000 0.000000000
      6  0.001855288 0.000000000 0.000000000 0.000000000 0.002949853
      7  0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
      8  0.000000000 0.002036660 0.001968504 0.002392344 0.000000000
      9  0.000000000 0.000000000 0.000000000 0.000000000 0.002949853
      10 0.000000000 0.000000000 0.001968504 0.002392344 0.000000000
      11 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
      12 0.000000000 0.002036660 0.000000000 0.004784689 0.000000000
      13 0.000000000 0.002036660 0.000000000 0.000000000 0.002949853
      14 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
      15 0.000000000 0.000000000 0.000000000 0.000000000 0.005899705
      16 0.990723562 0.002036660 0.011811024 0.011961722 0.150442478
      17 0.000000000 0.991853360 0.000000000 0.007177033 0.026548673
      18 0.001855288 0.000000000 0.978346457 0.007177033 0.005899705
      19 0.000000000 0.000000000 0.001968504 0.952153110 0.002949853
      20 0.000000000 0.000000000 0.000000000 0.000000000 0.719764012
```

As we can see, most of our training data gets classified correctly over 90% of the time (except for newsgroups 7 and 20 that are classified correctly only 81% of times and 71% of times, respectively, get mixed up with neighbouring newsgroups). The error rate is the complement of the accuracy rate, meaning that our error rate is between 0 and 0.1, except for newsgroup 7.

Next, we ran our classifier with the test data. Below the confusion matrix:

```
> test_results
            testlabels
maksimit_test          1          2          3          4          5
           1  0.93750000 0.00000000 0.00000000 0.00000000 0.00000000
           2  0.00000000 0.93103448 0.01754386 0.00000000 0.00000000
           3  0.00000000 0.00000000 0.92982456 0.00000000 0.01754386
           4  0.00000000 0.03448276 0.00000000 0.93220339 0.01754386
           5  0.00000000 0.00000000 0.00000000 0.01694915 0.96491228
           6  0.00000000 0.01724138 0.03508772 0.00000000 0.00000000
           7  0.00000000 0.00000000 0.00000000 0.03389831 0.00000000
           8  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
           9  0.00000000 0.01724138 0.00000000 0.00000000 0.00000000
           10 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
           11 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
           12 0.00000000 0.00000000 0.01754386 0.01694915 0.00000000
           13 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
           14 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
           15 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
           16 0.06250000 0.00000000 0.00000000 0.00000000 0.00000000
```

```
          17 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          18 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          19 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          20 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            testlabels
maksimit_test          6          7          8          9         10
          1  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          2  0.03389831 0.00000000 0.00000000 0.00000000 0.00000000
          3  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          4  0.00000000 0.01724138 0.00000000 0.00000000 0.00000000
          5  0.00000000 0.01724138 0.01694915 0.00000000 0.00000000
          6  0.94915254 0.01724138 0.00000000 0.00000000 0.00000000
          7  0.00000000 0.86206897 0.03389831 0.00000000 0.00000000
          8  0.00000000 0.05172414 0.94915254 0.00000000 0.00000000
          9  0.00000000 0.00000000 0.00000000 1.00000000 0.00000000
          10 0.00000000 0.00000000 0.00000000 0.00000000 0.98305085
          11 0.00000000 0.01724138 0.00000000 0.00000000 0.00000000
          12 0.01694915 0.00000000 0.00000000 0.00000000 0.00000000
          13 0.00000000 0.01724138 0.00000000 0.00000000 0.00000000
          14 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          15 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          16 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          17 0.00000000 0.00000000 0.00000000 0.00000000 0.01694915
          18 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          19 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          20 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            testlabels
maksimit_test         11         12         13         14         15
          1  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          2  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          3  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          4  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          5  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          6  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          7  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          8  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          9  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          10 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          11 0.98333333 0.00000000 0.00000000 0.00000000 0.00000000
          12 0.00000000 0.96610169 0.01694915 0.01694915 0.00000000
          13 0.00000000 0.00000000 0.96610169 0.00000000 0.00000000
          14 0.00000000 0.00000000 0.00000000 0.98305085 0.00000000
          15 0.00000000 0.00000000 0.01694915 0.00000000 1.00000000
          16 0.01666667 0.01694915 0.00000000 0.00000000 0.00000000
          17 0.00000000 0.01694915 0.00000000 0.00000000 0.00000000
          18 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          19 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
          20 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            testlabels
maksimit_test         16         17         18         19         20
```

```
            1  0.00000000 0.00000000 0.00000000 0.00000000 0.02702703
            2  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            3  0.00000000 0.00000000 0.00000000 0.02173913 0.00000000
            4  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            5  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            6  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            7  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            8  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            9  0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            10 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            11 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            12 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            13 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            14 0.00000000 0.00000000 0.01785714 0.00000000 0.00000000
            15 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
            16 0.96666667 0.00000000 0.01785714 0.02173913 0.18918919
            17 0.01666667 1.00000000 0.00000000 0.00000000 0.16216216
            18 0.01666667 0.00000000 0.96428571 0.00000000 0.00000000
            19 0.00000000 0.00000000 0.00000000 0.95652174 0.00000000
            20 0.00000000 0.00000000 0.00000000 0.00000000 0.62162162
>
```

We can conclude that the classifier works surprisingly well on the test data, which means that the train data and the test data were homogeneous enough for the classifier to work, and the model is robust enough for all the data. The error rate varies between 0 and 0.1, except for newsgroups 7 and 20, where the error rates are 0.14 and 0.39, respectively. From the newsgroups mappings we find out that newsgroup 7 is "misc.forsale", and it seems to get mixed up with computer and motor vehicles-related articles. The reason is obvious – for the classifier, it is hard to distinguish between sales texts and articles that contain similar vocabulary – one sells cars and another one analyzer cars, for example. Newsgroup 20 is "talk.religion.misc" and gets mixed up especially with groups 16 "soc.religion.christian", probably the vocabulary in both articles is similar in a way that "talk.religion.misc" includes many words that "soc.religion.christian" does, but not the opposite, as christian religion is a subsection of all religions. It was interesting that different computer-related newsgroups do get mixed up, but a little. One would expect lots of similar words in those.

Task 3

In this exercise we tried some dimensionality reduction techniques, where the dimensionality of the data is reduced in order to help any further analysis of the data. Our dataset is Fashion mnist that contains images of different clothing types (28x28 pixel grayscale). First, we applied Principal Component Analysis (PCA) on the data. We loaded the dataset but used only the items in classes "sandal", "sneaker" and "angle boot". In order to save computational time and space, we took a small subset of the data, which in our case was the ready-made test set with 10 000 digits items. We centered and normalized the data for further computation (the normalized and centered data is stored in the variable "X_centered_SD"). The idea of PCA is to find such linearly uncorrelated principal components that will maintain maximally the variance of the data.

We projected the data then on the two first principal components. In order to achieve that, we calculated the eigenvalues and eigenvectors of the obtained empirical covariance matrix: XtX=t(X)*X. Next, we took the first two principal components and projected our data on them by matrix multiplication .The results figure 4 is attached below: it can be seen that the data separated somehow, although the separation between items "6"(=sneakers) and "8"(=angle boots) is not too successful with plenty of overlap in the highly dense area.

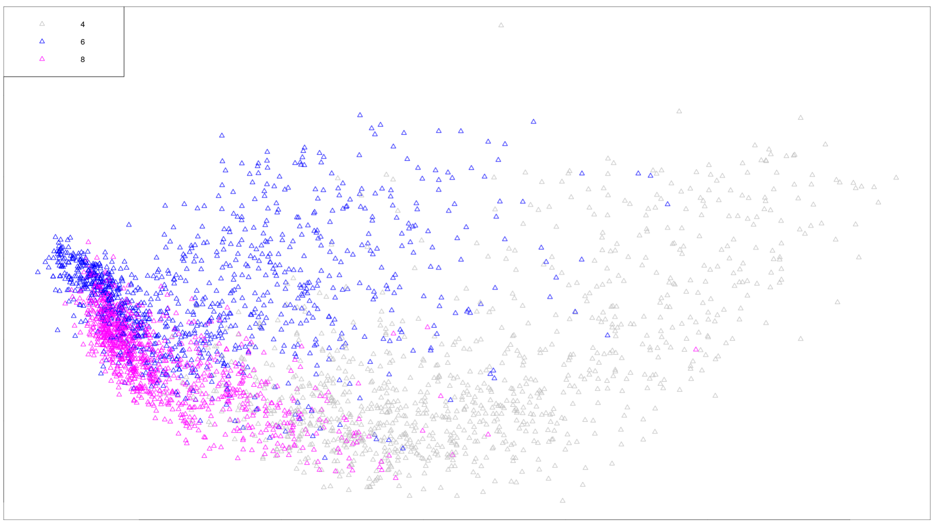

Figure 4: Data projection on the first two principal components. Blue color represents sneakers, grey represents sandals, magenta represents angle boots

Finally, we found an implementation of t-distributed stochastic neighbor embedding (t-SNE). One of the differences between PCA and t-SNE is that t-SNE is a non-linear dimensionality reduction technique. Thus, t-SNE is likely to be a more sensitive technique that could possibly achieve better separation in our case. We used a ready-made R package to calculate the results (method: tsne()). We obtained a two-dimensional representation of our data and produced a following plot in torms of the two first principal components. Results are displayed in figure 5:

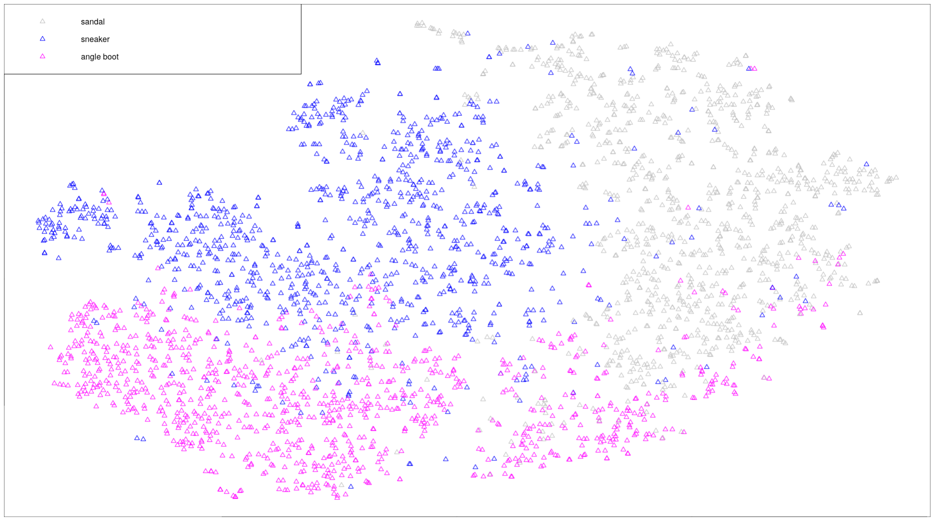

Figure 5: TSNE two-dimesional projection

The tsne-separator obviously worked better, as the groups are now well separated and overlap especially between sneakers and angle boots is smaller.