Image Analysis - FMAN20
Assignment 3

Least Squares and Total Least Squares
Machine Learning
OCR - Classifier Construction

Hicham Mohamad
hsmo@kth.se

November 8, 2018

1 Introduction

In this assignment we will work on line fitting problem using Least Squares methods, as well as on machine learning problems and continue our work on our own Optical Character Recognition system from the previous assignments.

2 Line fit

For this task, we need to edit the file `task1.m`. In the file `linjepunkter.mat` a set of data points are in the plane. Then, we shall fit `least squares` and `total least squares` lines to these data points.
In linear algebra, we can represent lines as

$$y = kx + m \tag{1}$$

Each point on the line is a point in the km-plane. k is the `slope` of the line, m is the `y-intercept`. we notice that in this line form `vertical` lines, i.e. undefined slope/gradient, cannot be represented.

2.1 Least squares (LS) approach

In least squares method, it is assumed that the errors are only in the y-direction. Assuming that the points (x_i, y_i) are measured, then

$$y_i = kx_i + m$$

for line parameters (k, m), we write this in matrix form adding the vector $\mathbf{n}$ representing an error

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \begin{pmatrix} k \\ m \end{pmatrix} + \mathbf{n} = Ap + \mathbf{n}$$

When the errors $\mathbf{n}$ are `independent` and `Gaussian` distributed, then we can solve $y = Ap$ in least squares sens, i.e. by minimizing $|y - Ap|$. The solution is

$$p = (A^T A)^{-1} A^T y$$

In matlab the least squares solution can be obtained using the `slash function` or 'slash'-operator

$$p = A \backslash y = \begin{pmatrix} 7.5687 \\ 5.3613 \end{pmatrix}$$

One problem with LS above lies in the two assumptions:

- It cannot handle vertical lines.
- For lines that are close to vertical the assumption that the errors are only in the y-direction gives sub-optimal estimates of the line.

It is better to `minimize the distance` between the point and the line. This idea is used in the total least squares (TLS) approach shown in the next section.

2.2 Total least squares (TLS) approach

From linear algebra we know that the distance between the point (x, y) and the line $ax + by + c = 0$ with line parameters $l = (a, b, c)$ is given by

$$\frac{|ax + by + c|}{\sqrt{a^2 + b^2}} \tag{2}$$

Now assuming that $a2 + b2 = 1$ then the distance is $d = |ax + by + c|$. The line $l = (a, b, c)$ that minimizes the `sum of squares` of the distance is given by

$$f(a, b, c) = \min_{a,b,c} \sum_i (ax_i + by_i + c)^2. \tag{3}$$

Then, we solve the `optimization problem` by using the `Lagrange function` is

$$L(a, b, c, \lambda) = \sum_i (ax_i + by_i + c)^2 + \lambda(1 - a^2 - b^2) \tag{4}$$

whose stationary points is given by

$$\begin{pmatrix} \bar{x^2} & \bar{xy} & \bar{x} \\ \bar{xy} & \bar{y^2} & \bar{y} \\ \bar{x} & \bar{y} & N \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \lambda \begin{pmatrix} a \\ b \\ 0 \end{pmatrix}$$

where the `'bar'` sign denotes the sum, for example $\bar{x^2} = \sum_i x_i^2$. We can solve for c using the last equation:

$$c = -\frac{1}{N}(a\bar{x} + b\bar{y})$$

Then we can substitute c in the equations above to obtain the following solution

$$\begin{pmatrix} \bar{x^2} - \frac{1}{N}\bar{x}\bar{x} & \bar{xy} - \frac{1}{N}\bar{x}\bar{y} \\ \bar{xy} - \frac{1}{N}\bar{x}\bar{y} & \bar{y^2} - \frac{1}{N}\bar{y}\bar{y} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 0,2470 & 1,8700 \\ 1.8700 & 16.3776 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \lambda \begin{pmatrix} a \\ b \end{pmatrix} \tag{5}$$

This is an `eigenvalue problem`, i.e. $Av = \lambda v$ and there are two solutions which are `orthogonal`. One maximizes the likelihood, the other minimizes it. It is straightforward to test which one of the two minimize $f(a, b, c)$. In this way, we find the values a,b and c that minimize the sum of squares of the distance. Using the Matlab function `eig()` as shown in Listing 1, we get the following eigenvectors:

$$W = \begin{pmatrix} -0.9935 & 0.1137 \\ 0.1137 & 0.9935 \end{pmatrix}$$

Plots with the data points and the two fitted lines using LS and TLS methods are illustrated in Figure 1.

Listing 1: The code for the respective line fittings and errors computing .

```
% Task 1: Fit least squares and total least squares lines to data points.

% Clear up
clc;
close all;
clearvars;

% Begin by loading data points from linjepunkter.mat
load linjepunkter

% Convenient to have x, y as column vectors
x = x';
```

```
13 y = y';
14 N = length(x); % number of data points
15
16 % Plot data
17 plot(x, y, '*'); hold on;
18 xlabel('x')
19 ylabel('y')
20 title('Line Fittings with LS and LTS methods') % OBS - CHANGE TITLE!
21 x_fine = [min(x)-0.05,max(x)+0.05]; % used when plotting the fitted lines
22
23 % Fitting line with Least Squares
24 % Fit a line to these data points with least squares
25 % Here you should write code to obtain the p_ls coefficients (assuming the
26 % line has the form y = p_ls(1) * x + p_ls(2)).
27
28 % XXXXXXXX TO DO
29 %p_ls = [rand(), 6]; % REMOVE AND REPLACE WITH LEAST SQUARES
30
31 A = [x ones(length(x),1)];
32 % least squares solution p = [(A'*A)^(-1)]*A'*y
33 % In matlab the least squares solution can be obtained using the
34 % slash function (operator).
35 p_ls = A\y
36
37 plot(x_fine, p_ls(1) * x_fine + p_ls(2));
38
39 % ######    Fitting line with Total Least Squares
40 % Fit a line to these data points with total least squares.
41 % Note that the total least squares line has the form
42 % ax + by + c = 0, but the plot command requires it to be of the form
43 % y = kx + m, so make sure to convert appropriately.
44
45
46 % XXXXXXXXXX TO DO
47 %p_tls = [rand(), 6]; % REMOVE AND REPLACE WITH TOTAL LEAST SQUARES
48 % here we implement the eigenvalue problem to get the parameters a, b and c
49 sumx2 = sum(x.^2);
50 sumxsumx = sum(x)*sum(x);
51 sumxy = sum(x.*y);
52 sumxsumy = sum(x)*sum(y);
53 sumy2 = sum(y.^2);     %*sum(y);    %XXXXXXX
54 sumysumy = sum(y)*sum(y);
55
56 A11 = sumx2 - 1/N*sumxsumx;
57 A12 = sumxy - 1/N*sumxsumy;
58 A21 = sumxy - 1/N*sumxsumy;
59 A22 = sumy2 - 1/N*sumysumy;
60
61 A_tls = [A11 A12; A21 A22];
62 %[~,~,W] = eig(Atls); % [V,D,W] = eig(A) also produces a full matrix W
       whose
63 % columns are the corresponding left eigenvectors so
64 % that W'*A = D*W'.
65 % here we find the values a,b and c that minimize the sum of squares of the
66 % distance
67 [W,D] = eig(A_tls);
68 V = W(:,1,:);
69 a = V(1);
70 b = V(2);
71 c = -1/N*(a*sum(x) + b*sum(y));
72
```

```
73 % conversion to y = kx + m line form
74 % ax + by + c = 0 => y = -a/bx + -c/b = y = kx + m
75 p_tls(1) = -a/b;
76 p_tls(2) = -c/b;
77
78 plot(x_fine, p_tls(1) * x_fine + p_tls(2), 'k--')
79
80 % Legend --> show which line corresponds to what (if you need to
81 % re-position the legend, you can modify rect below)
82 h=legend('data points', 'least-squares','total-least-squares');
83 rect = [0.20, 0.65, 0.25, 0.25];
84 set(h, 'Position', rect)
85
86 % Compute the 4 errors
87 % After having plotted both lines, it's time to compute errors for the
88 % respective lines. Specifically, for each line (the least squares and the
89 % total least squares line), compute the least square error and the total
90 % least square error. Note that the error is the sum of the individual
91 % errors for each data point! In total you should get 4 errors. Report
       these
92 % in your report, and comment on the results.
93 % OBS: Recall the distance formula between a point and a line from linear
94 % algebra, useful when computing orthogonal errors!
95
96 % WRITE CODE BELOW TO COMPUTE THE 4 ERRORS
97 % Line 1
98 % compute the least square error
99 n_ls = abs(y - A*p_ls);
100 e_ls = sum(n_ls)
101
102 % the total least square errors (orthogonal errors)  XXXXXX TO DO
103 a1 = p_ls(1); b1 = -1; c1 = p_ls(2);
104 n1_tls = abs((a1*x + b1*y + c1))/sqrt(a1^2 + b1^2);
105 e2_tls = sum(n1_tls)
106
107 % Line 2
108 % the total least square errors (orthogonal errors)
109 n2_tls = abs((a*x + b*y + c))/sqrt(a^2 + b^2);
110 e2_tls = sum(n2_tls)
111
112 % LS solution can be obtained by
113 n2_ls = abs(y - A*p_tls');
114 e2_ls = sum(n2_ls)
```

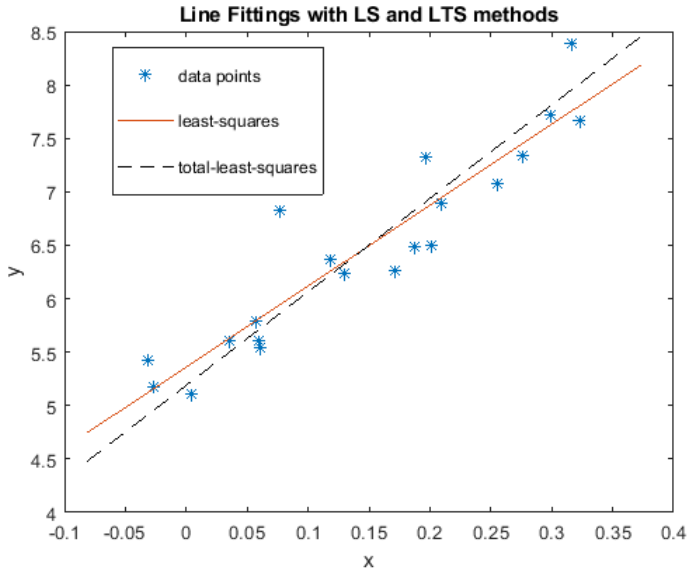


Figure 1: The plots with the data points and the two fitted lines

2.3 LS and TLS errors

We need to compute errors for the respective lines. Specifically, for each line, i.e. the line estimated with the Least Squares and that one estimated with the Total Least Squares.
The least square error is the sum of the `vertical errors`

$$\mathbf{n} = |y - Ap|$$

Using this formula we get

- Sum of the LS errors for the line fitted with TS methode is 5.0451.
- Sum of the LS errors for the line fitted with LTS method is 5.7998.

The total least squares line error is computed by using the distance formula between a point and a line. This error is the sum of the individual errors for each data point, i.e. the `orthogonal errors`

$$\mathbf{e} = \frac{|ax_i + by_i + c|}{\sqrt{a^2 + b^2}}$$

Using this formula we get

- Sum of errors for line fitted with the TLS approach is 0.6592.
- Sum of the TLS errors for the line estimated with LS approach is 0.6608.

Conclusion: Looking at these results, we can conclude that with Least Squares method we get a `sub-optimal` estimate of the line because of the assumption that the errors are only in the y-direction. In this case, it is better to minimize the distance between the point and the line using TLS method. If the errors n are independent and Gaussian distributed, then it is reasonable to solve $y = Ap$ in Least Squares sense, i.e. minimizing $|y - Ap|$, but it is not the case in our problem.

3 Your own classifier

In this task we need to code our own classifier to better understand machine learning. Machine Learning typically has two phases as illustrated in Figure 2

- *Phase 1 – Training*: A training dataset is used to estimate model parameters and store these parameters. Code usually assumes that input are vectors.

- *Phase 2 – Prediction*: Once the parameters have been estimated, we can use the model to classify future data.

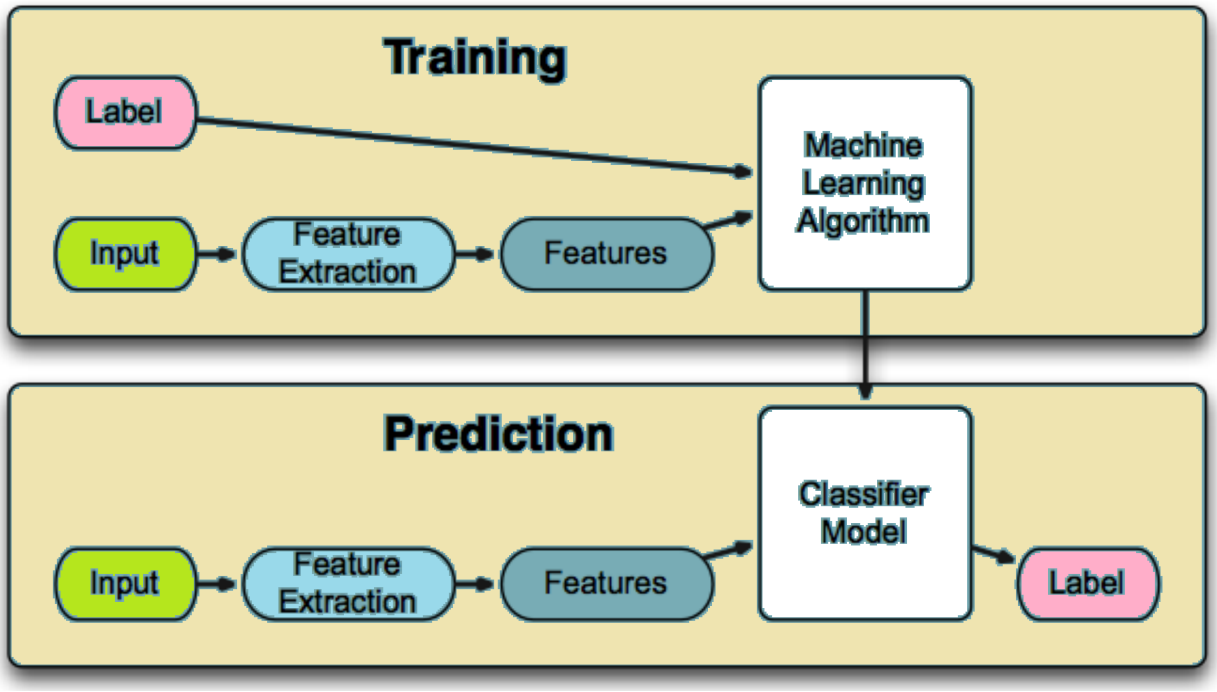


Figure 2: Machine learning typically has 2 phases: training and prediction. Phase 1 – A training dataset is used to estimate model parameters and store these parameters. Code usually assumes that input are vectors. Phase 2 – Prediction: Once the parameters have been estimated, we can use the model to classify future data

Therefore, most classifiers consists of two parts, a `training` part and a `classification` part. For the training part, one uses a training set consisting of example `feature vectors` x_i with corresponding `ground truth` y_i. The training set

$$T = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$$

is sometimes represented as the matrices

$$\mathbf{X} = (\mathbf{x}_1 \cdots \mathbf{x}_n)$$

and

$$\mathbf{Y} = (y_1 \cdots y_n).$$

Here each feature vector $\mathbf{x}_i$ typically contains several features, i.e. x_i is a column vector, whereas y_i is a scalar that denotes the ground truth label. we should mention that in our classifier, i.e. for the nearest neighbour method, there is no actual training phase, instead, the training data is simply stored in `classification_data`. An illustration for the code of the function `class_train` is shown in Listing 2.

3.1 A simple non-parametric classifier: K-nearest neighbors

As a classification method, the `K nearest neighbor (KNN)` classifier is chosen. It is a `non-parametric` classifier, that is, the model has a number of parameters which grow with the amount of training data. This simply classifies using the label of the nearest neighbour, i.e. it looks at the K points in the training set that are `nearest` to the test input x and counts how many members of each class are in this set. Then this will be classified in the class y with the most representatives among these K. Otherwise, it is classified as 'unknown'.

During classification one compares the `distance` between x with the ones in the training set $(x_1, ..., x_n)$, i.e. solves

$$k = \arg\min_i d(x_i, x) \quad \text{and put} \quad y = k$$

where the most common distance metric to use is Euclidean distance and where y denote the class index. Then, the object is simply assigned to the class of those nearest neighbors.

The method is a type of `instance-based learning`, or `memory-based learning` and can work quite well, provided it is given a good distance metric and has enough labeled training data. However, the main problem with KNN classifiers is that they don't work well with high dimensional inputs. This poor performance in high dimensional settings is due to the `curse of dimensionality`.

Normalizing: For classification algorithms like KNN, where we measure the `distances` between pairs of samples, these distances are influenced by the measurement units also. To avoid any misclassification problem, we should normalize the feature variables. Any algorithm where distance play a vital role for prediction or classification, we should normalize the variable as we do the same process also in PCA, i.e. Principal Component Analysis.

For implementing K-NN classification algorithm in matlab, the following main steps are used and the implementation is shown in Listing 3:

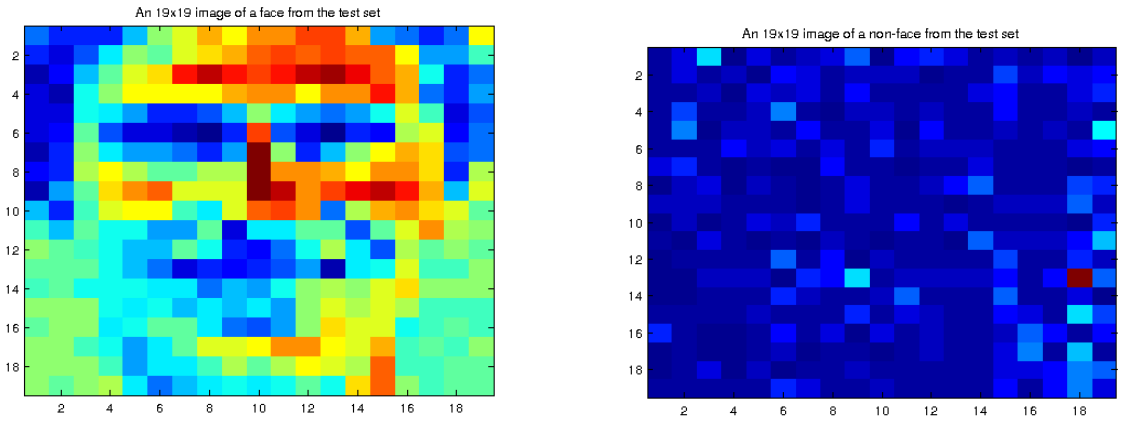
1. Load the training and test data
2. Choose the value of $K = 5$
3. For each point in `test data`:
   - find the Euclidean distance to all training data points
   - store the Euclidean distances in a list and sort it
   - choose the first K points
   - assign a class to the test point based on the majority of classes present in the chosen points
4. End

3.2 Cross validation

The dataset `FaceNonFace.mat` on which we need to try our classifier contains 200 feature vectors as columns in a matrix $\mathbf{X}$ and the corresponding ground truths in a vector $\mathbf{Y}$. Using the given code in `task2.m`, these 200 examples are patitioned into 80 % training set and 20 % testing or `validation set`, by using Matlab's built-in function `cvpartition`, i.e. using `cross validation (CV)`. In this way, we can fit the model on the training set and evaluate its performance on the validation set.

In this way, the dataset contains a matrix $\mathbf{X}$ of size 361×200. Each column consists of 361 pixels from 200 small 19×19 pixel images. In the matrix $\mathbf{Y}$ the ground truth is given for the 200 examples. These are coded so that '1' corresponds to face and '-1' to non-face.

After choosing two figures from the test set, One 19×19 image of a face and one 19×19 image of a non-face, we tried the implemented classifier on them and we verified that it could predict them correctly, as shown in Figure 3. For displaying these figures two Matlab functions `reshape` and `imagesc` are used.



(a) (b)

Figure 3: a) Plot of face image from the test set . b) Plot of an non-face image from the test set. This shows that the implemented K-NN classifier predict the image of the face to be a face and similar for the non-face.

3.3 Misclassification rate - Model selection

To study the perfermance of an classifier, we usually calculate the `misclassification rate`, i.e. overall, how often is it wrong? or how many aren't correctly classified?

$$ErrorRate = \frac{(FP + FN)}{total} = 1 - Accuracy$$

where

- Negatives that are classified as negatives = True Negatives (TN)
- Positives that are classified as positives = True Positives (TP)
- Negatives that are classified as positives = False Positives (FP)
- Positives that are classified as negatives = False Negatives (FN)

After running the given script `task2_and_3.m` with 100 trials, we get the following average error rates

- Average error rates on training data for KNN classifier: 0.05
- Average error rates on testing data for KNN classifier: 0.125

As mentioned in [4], when we have a variety of different complexity, i.e. KNN classifiers with different values of K, how should we pick the right one? A natural approach is to compute the misclassification rate on the training set.
The value of K can have a large effect on the behavior of this model. When $K = 1$, the method makes no errors on the training set since we just return the labels of the original training points, i.e. the model is just memorizing the data. But the resulting prediction surface becomes very wiggly, i.e. the method `overfits`. Thus, we need to be careful that we don't overfit the data, that is, we should avoid trying to model every minor variation in the input, since this is more likely to be `noise` than true signal.
On the other hand, increasing K increases our error rate on the training set, because we are over-smoothing, i.e. the method `underfits`. Therefore, an obvious way to choose K is to pick the value with the minimum error on the validation/test set. That's why $K = 5$ was a good choice in our problem .

Listing 2: The code for `class_train()` function .

```
1
2  function classification_data = class_train(X, Y)
3  % IMPLEMENT TRAINING OF YOUR CHOSEN MACHINE LEARNING MODEL HERE
4  %classification_data = 0; % REMOVE AND REPLACE WITH YOUR CODE
5  %{
6  Here classification_data is a variable that contains whatever information
       we have extracted during training and is needed during testing.
7  %}
8  [m,n] = size(X);
9  for i = 1:n
10    X(:,i) = normalize(X(:,i));
11 end
12
13 % store the training data in classification_data
14 classification_data = cell(1,2);
15 classification_data{1} = X;
16 classification_data{2} = Y;
17
18 end
```

Here in Listing 3, we can see the function `classify()` implemented in Matlab

Listing 3: The code for classify() function .

```
1
2  function y = classify(x, classification_data)
3  % IMPLEMENT YOUR CHOSEN MACHINE LEARNING CLASSIFIER HERE
```

```
4  %y = 0; % REMOVE AND REPLACE WITH YOUR CODE
5
6  % Using the K-Nearest Neighbours algorithm
7
8  % K represents the number of training data points lying in proximity to the
9  % test data point which we are going to use to find the class.
10 % choose the value of k
11 K = 5;
12
13 isFace = 0;
14 nonFace = 0;
15
16 % 0. preprocessing of data set
17 x = normalize(x);
18
19 % 1. Loading the training and test data\
20 X = classification_data{1};
21 Y = classification_data{2};
22
23 [m1,n1] = size(X);
24 [m2,n2] = size(Y);
25
26 faces = [];
27 nonfaces = [];
28
29 % 3. For each point in test data:
30 for i = 1:n1
31    % distance measurement to all training data points
32    d = norm(X(:,i) - x);
33
34    % store the distances in a list
35    if Y(i) == 1
36        faces = [faces d];
37    else
38        nonfaces = [nonfaces d];
39    end
40
41
42 end
43
44 % sort the Euclidean distances
45 faces = sort(faces);
46 nonfaces = sort(nonfaces);
47
48 i = 1;
49 % choose the first k points
50 while i < K
51     if faces(1) <= nonfaces(1)
52         isFace = isFace + 1;
53         faces = faces(2:end);
54
55     elseif nonfaces(1) < faces(1)
56         nonFace = nonFace + 1;
57         nonfaces = nonfaces(2:end);
58     end
59     i = i + 1;
60 end
61
62 % assign a class to the test point based on the majority of classes present
63 % in the chosen points
64 if isFace > nonFace
```

```
65        y = 1;
66    else
67        y = -1;
68    end
69
70    end
```

4 Use pre-coded machine learning techniques

There are many machine learning techniques. Most of them have a training part and a classifying/evaluating/testing part. In this exercise we will practice on understanding the inputs and outputs of the code of some machine learning algorithms.

In this task, we need to try our own classifier together with other three different machine learning techniques: regression tree classifier, support vector machine and nearest neighbour in `task2_and_3.m`. As in the last task, we select a random training subset using Matlab's built-in function `cvpartition`. Here are Matlab's corresponding built-in training routines:

```
1    tree = fitctree(X,Y); % Regression tree classifier
2    SVMModel = fitcsvm(X,Y); % Support Vector Machine
3    mdl = fitcknn(X,y); % Nearest Neighbour Classifier
```

Here in the table, we can show the Result of the average error rates for my own classifier KNN, and for the three built-in classifiers, i.e. 4 for training and 4 for testing data.

	my KNN	tree	svm	nn
Testing data	0.0930	0.1505	0.0415	0.1615
Training data	0.0581	0	0	0

Table 1: Average error rates for the four classifiers both on training and testing data.

Looking at the results in table, we notice that fitting the model on the training set does not result in any error rate for the built-in classifiers. For my classifier instead, the error exists just when $K > 1$, otherwise it should be zero as well.

However, evaluating the performance of these classifiers on the validation set results in error rate with all them. But we can see that the best performance is with the built-in svm classifier and then comes my own KNN classifier. These error rates on testing data can be derived from the fact that the features are not good enough or that the algorithm is not compatible with this type of problem we are working on.

5 OCR system construction and system testing

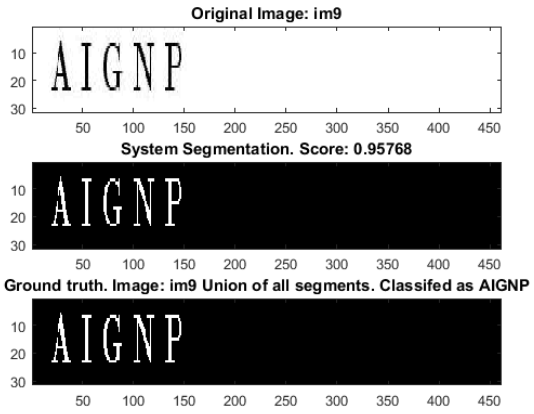
Here we will continue the work on developing our OCR system. From the two previous assignments, we need to use again the same functions implemented before, i.e.

`S = im2segment(Im)` - a segmentation algorithm that takes an image to a number of segments.

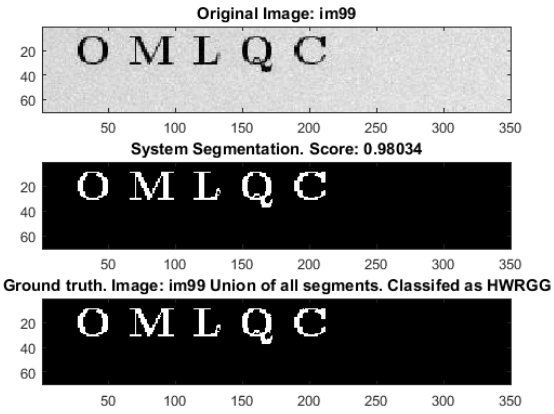
`x = segment2features(S)` - an algorithm for calculating a feature vector x from a segment S.

Now, using machine learning we construct a classifier that takes a feature vector $\mathbf{x}$ as input and returns a number y as output, i.e. a class. Here y is an integer between 1 and 26. The characters are coded from 1 to 26, where 1 corresponds to 'A', 2 corresponds to 'B' and so forth.

The variable `classification_data` is what we store from the training of the classifier. In our case this depends on K-NN algorithm, so we can use `class_train.m` that we coded before. Giving a Matlab file `ocrsegments.mat`,which it contains a number of segments in a cell array S and corresponding letter code y. Using the selected features in Assignment 2, we will go from segments in S to generate corresponding feature vectors in a matrix $\mathbf{X}$, where each column corresponds to features of a segment in S. In this way, we use KNN machine learning method to train the classifier and then we store the data from the training as a variable `classification_data`.



(a)



(b)

Figure 4: a) Output from running the benchmark test on the dataset short1. b) Output from running the benchmark test on the dataset home1.

5.1 Hitrates for the OCR-system

After running OCR-system on short1 and home1 datasets we get the following hitrates Results:

- Running the script on short1 results in $hitrate = 1$, without any errors from Matlab.
- When we run the benchmark test script on home1 dataset results in a Matlab error saying: `" Subscript indices must either be real positive integers or logicals."` Then, when I take away 2 of the 9 features on which the Matlab error is related, as shown in Listing 4, it works and I could run the script resulting in very low $hitrate = 0.0960$.

As expected in the assignment text, the error rate for the 'home1' isn't good because the examples in 'home1' are slightly more difficult. So this poor performance can be explained by the fact that the features extraction is not working very well. We will work more on improving this in Assignment 4, as mentioned in the text.

Listing 4: The code for features2class() function .

```
1
2  function y = features2class(x, classification_data)
3
4  % Here we can choose K to K-NN algorithm
5  K = 1;
6  % the stored training data which come from class_train function
7  feats_train = classification_data{1};
8  Y_train = classification_data{2};
9  N = size(feats_train,2);
10
11 % initialize a list for the distances to all training data points
12 D = zeros(1,N);
13
14 for i = 1:N
15     % calculate the Euclidean distance and store it in the list
16     D(i) = norm(feats_train(:,i) - x);
17 end
18
19 % initialize a voting vector
20 votes = zeros(1,K);
21
22 % voting loop
23 for i = 1:K
24    % get the index of the least distance
```

```
25      minimum = min(D);
26      index = min(find(D == minimum));
27
28      % assign the class of the K nearest neighbors
29      votes(i) = Y_train(index);
30      D(index) = Inf; % IEEE arithmetic representation for positive infinity
31  end
32
33  % Assign a class based on the majority of classes present in the chosen
34  % points, now we have only one element because K = 1
35  % mode() get the most frequently occurring value in votes
36  y = mode(votes);
37
38  end
```

References

[1] Szeliski, Computer Vision - Algorithms and Applications, Springer.

[2] Forsyth and Ponce, Computer Vision - A Modern Approach, Pearson Education, ISBN 0-13-191193-7

[3] Robert M. Haralick and Linda G. Shapiro. *Computer and Robot Vision*, volume 1 Addison-Wesley, 1991.

[4] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*, The MIT Press, Cambridge, Massachusetts.

[5] Matlab journal demos in Image Analysis. Available: `http://www.ctr.maths.lu.se/matematiklth/personal/kalle/imageanalysis/`.

[6] Linear Algebra. Available: `http://www.immersivemath.com`

[7] Computer Vision - CS6670, Lectures notes [Online]. https://www.cs.cornell.edu/courses/cs6670/2018fa/calendar-final.html

[8] Maskinsyn - UNIK4690, Lectures notes [Online]. https://www.uio.no/studier/emner/matnat/its/UNIK4690/v16/time

[9] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *Gradient-Based Learning Applied to Document Recognition.* PROC. OF THE IEEE, November 1998.

[10] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning.* Second Edition, Springer.