Centre for Mathematical Sciences, Mathematics, LTH, September 2023

Image Analysis, Assignment 3

1 Your own classifier

For this task, go to the folder `task1_and_2` in Matlab and edit the files `task1_and_2.m`, `class_train.m` and `classify.m`.

To better understand machine learning it is useful to have coded your own classifier. 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 $\boldsymbol{x}_i$ with corresponding ground truth y_i. The training set

$$T = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$$

is sometimes represented as the matrices

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n \end{pmatrix}$$

and

$$\boldsymbol{Y} = \begin{pmatrix} y_1 & \cdots & y_n \end{pmatrix}.$$

Here each feature vector $\boldsymbol{x}_i$ typically contains several features, i.e. $\boldsymbol{x}_i$ is a column vector, whereas y_i is a scalar that denotes the ground truth label. *Note that in some systems the feature vectors are row vectors instead. In this case you need to transpose the matrix $\boldsymbol{X}$, but don't worry about this now (it is important in Task 3).*

Choose your favorite classification method (e.g. Bayes, Nearest Neighbour, or Support Vector Machine) and implement your own routine for training

```
function [classification_data] = class_train(X, Y)
```

and a routine for classification

```
function [y] = classify(x, classification_data)
```

Here `classification_data` is a variable that contains whatever information you have extracted during training and is needed during testing. Note that for the nearest neighbour method, there is no actual training phase; instead, the training data is simply stored in `classification_data`.

Some machine learning techniques are easier to implement than others. Of the aforementioned methods, nearest neighbor is the easiest, the Bayesian approach assuming Gaussian distributions[1] *is the second easiest, and the most difficult one is the support vector machine. Choose wisely!*

[1] In addition to assuming Gaussian distributions, it is often necessary to make additional model assumptions. In this case one such assumption could be to assume that individual features are independent of each other, in which case the problem is reduced to Naïve Bayes (see wiki).

The dataset `FaceNonFace.mat` contains 200 feature vectors as columns in a matrix $\boldsymbol{X}$ and the corresponding ground truths in a vector $\boldsymbol{Y}$. You are given code (see `task2.m`) which randomly partitions these 200 examples into 80% training and 20% testing, by using Matlab's built-in function `cvpartition`; make sure that you understand how this works.

Try your implemented method on the dataset `FaceNonFace.mat`. It contains a matrix $\boldsymbol{X}$ of size 361×200. Each column consists of 361 pixels from 200 small 19×19 pixel images. There are thus 200 such feature vectors. In the matrix $\boldsymbol{Y}$ the ground truth is given for the 200 examples. These are coded so that '1' corresponds to face and '-1' to non-face.

For the report: You should supply

- Relevant code (i.e. the code for your two functions classify and class_train).
- Average error rates (100 trials) for your chosen classifier, **both on training and testing data**, including **comments on these results**.
- **Two figures:** One 19×19 image of a face and one 19×19 image of a non-face from the **test set**. In the figure text for these two images, write whether your classifier predicted them correctly (e.g. did your method predict the image of the face to be a face, and similar for the non-face). Note that you have to write code for displaying these figures as well as selecting the face and non-face yourself. You may want to look up the functions *reshape* and *imagesc* in Matlab, and of course you may want to reuse some of the other code in this task.

2 Use pre-coded machine learning techniques

For this task, go to the same folder as the previous task, `task1_and_2` in Matlab and edit the file `task1_and_2.m`.

There are many machine learning techniques. Most of them have a training part and a classifying/evaluating/testing part. Some of these take considerable effort to code. It is important to be able to re-use code. One of the difficulties here lies in understanding the inputs and outputs of such code. In this exercise you will practice this.

In the file `task1_and_2.m`, which you worked with in Task 1, you should now also uncomment the code suggested in the file (for Task 1) and edit the code to make it work. **Don't do anything with the code that you already implemented in the last task, just leave everything as it is.**

Try the suggested three different machine learning techniques (regression tree classifier, support vector machine and nearest neighbour) in `task1_and_2.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:

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

NOTE: When using these routines you have to transpose the matrices, since they assume that each example is a row in the matrix.

See `help cvpartition`, `help fitctree`, `help fitcsvm`, `help fitcknn` and `help predict` (or, perhaps better, search for these keywords in your favourite search engine).

For the report: You should supply

- Average error rates (100 trials) for your own method, and for the three built-in classifiers, **both on training and testing data**. Hence it should be in total **8 errors** (4 for training and 4 for testing).
- **Comment on these results**, both between the three built-in methods, but also in relation to the results you obtained with your own classifier.

3 Testing a simple CNN model

For this task, go to the folder `task3` in Matlab
Here there are two prewritten routines `trainSimpleCNN.m` and `predictSimpleCNN.m`. These functions setup and train a simple CNN and predict using it respectively. Study the scripts and try to understand what they do. They are hardcoded for the classification task that you tested in `task1` and `task2`. Train and test this CNN model in the same way as in `task1` and `task2` (on the same data). Again, for each trial, divide examples into 80% training and 20% testing. You are welcome to change the training parameters and the model.

For the report: You should supply

- Average error rates (100 trials) for the predefined CNN classifier, **both on training and testing data**. Hence it should be in total **2 errors** (1 for training and 1 for testing).
- **Comment on the results**, in relation to the results you obtained in the previous tasks.

4 Line fit

For this task, go to the folder `task4` in Matlab and edit the file `task4.m`.

In the file `linedata.mat` are a set of data points in the plane. In this task you shall fit a line to these points according to

- the total least squares (TLS) approach, and
- the RANSAC approach (you can choose to count inliers based on either the least square error or the total least squares error). Here use RANSAC to find an initial solution and the inlier set. After you have found the inlier set, find the best line according to TLS on the inlier set.

See lecture notes for a derivation / discussion of the two approaches.

For the report: Provide your **code for the respective line fittings**. Provide plots with the data points and the two fitted lines in **one figure**. Write a few sentences on the result, including:

- How many points do you minimally need to sample, in order to estimate the line model in RANSAC?
- How is the line model estimated for the minimal dataset in the RANSAC approach?
- What is the difference between the two methods?
- What are the errors for the two lines? In particular, you should for each of the two lines compute the LS and TLS errors, where for RANSAC you calculate the errors on the inlier set only[a]. Include these **four errors** (two errors per line) in your report, and **comment on them**. Provide also the code for how you computed these 4 errors. In this case discuss which method is best (Total Least squares or RANSAC) and why (relating also to the errors).

[a]The LS error is the sum of the *squared* vertical errors, and the TLS error is the sum of the *squared* orthogonal errors. Also, the errors in this assignment refers to errors on the same data that you fit your lines to, i.e., to the 40 data points for TLS and on the inliers set for RANSAC, in linedata.mat

5 OCR system construction and system testing

For this task, go to the folder `task5/ocr_project/matlab` in Matlab and have a look at the files `first_steps_handin3_task5.m` and `inl3_test_and_benchmark.m`.

Here we will continue the work on our OCR system. From the two previous assignments you have constructed

- `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.

Put both of these Matlab files in the folder `task5/ocr_project/matlab`. Now, using machine learning you will construct a classifier that takes a feature vector $\boldsymbol{x}$ and calculates a class y. Write this as a function[2] `y = features2class(x,classification_data)`. The function takes a feature vector $\boldsymbol{x}$ as input and returns a number y as output. Here y is an integer between 1 and 10. (The numbers are coded from 1 to 10, where 1 corresponds to '0', 2 corresponds to '1' and so forth.)

The variable `classification_data` is whatever you need to store from the training of the classifier. What this is depends on what machine learning algorithm you are using. Again, if you successfully completed task 1, then you can use `class_train.m` that you coded

[2]Note that this function can be the same function for classification that you already implemented in task 2.

there. In the folder `task5/ocr_project/matlab`, there is a Matlab file `ocrsegments.mat`. It contains a number of segments in a cell array S and corresponding number code y. Again, these are coded from 1 to 10, where 1 corresponds to '0', 2 corresponds to '1' and so forth. Use your own set of favorite features (recall what you did in Assignment 2), to go from segments in S to corresponding feature vectors in a matrix $\boldsymbol{X}$, where each column corresponds to features of a segment in S. See `first_steps_handin3_task5.m`. Use your favorite machine learning method to train a classifier. Store the data from the training as a variable `classification_data` (most things are automatically done in the function `first_steps_handin3_task5.m`, so use that!).

Code the function `y = features2class(x,classification_data)`. Here you could use your own classifier from exercise 1 if you want to.

When you're done with `first_steps_handin3_task5.m`, i.e. you have created `classification_data.mat`, try running `inl3_test_and_benchmark`. In the script you will specify which routines to use for segmentation, feature extraction and classification, as well as specifying which classification data you got from the training of the classifier. You can also specify which dataset to try the system on. There are two datasets 'short1' (with 10 example images) and 'home1' (with 200 example images). **Try your system on both of these.**

Note that the examples in 'home1' are slightly more difficult, so one could expect the performance to be worse on it. We will work more on improving this in Assignment 4. In other words it is ok if the error rate for the 'home1' isn't very good.

It is important to test your image analysis system on real data with ground truth. This makes it possible to check your system and verify that it works. However, if the system does not work, a high error rate can be quite uninformative. It is difficult to know if it is the segmentation part that fails, if it is the features that are not good enough or if it the classification algorithms that fails. In the script `inl3_test_and_benchmark` there are several options for getting information about the performance of different parts of the system. Therefore, try setting `mode=2` in `inl3_test_and_benchmark` if your hitrates are too low (**you should get close to at least 0.50 as hitrate on 'short1'**).

For the report, provide **hitrates for your OCR-system on short1 and home1**, including **comments on these results**. You should get close to **at least 50% hitrate for short1; any hitrate will do for home1**. Also, as for all tasks, write a **comprehensive report** where you explain what you have done. For example, if you had to improve any parts of your system in order to get at least 50% hitrate on short1, write about this. Similarly, if you got very low hitrate on home1, try to suggest why that is.