IN 5520 2018

Classification 1

Exercises related to the lecture on 9.10.18

A light exercise set to get you started on classification. You will need to implement your own Gaussian classifier for mandatory exercise 2.

All images/files are under undervisningsmateriale/week9. Start by downloading it.

Exercise 1.

Matlab exercise for classification based on a univariate Gaussian classifier.

Step 1: Implement a Gaussian classifier using a single feature at a time.

For the algorithm, see last lecture foils

The image tm1.png to tm6.png contain the 6 original bands of a Landsat satellite image of Kjeller (corresponding to different wavlengths).

For images = tm1.png, tm2.png,… tm6.png do

Step 2: Train the classifier

Train the classifier on the image tm$.png (replace with the band number $) (band (found at undervisningsmateriale/week6) using the 4 classesdefined in the mask file tm_train.png. You do this by computing the mean vector and variance for each class k for based on the pixels with class label k in the training mask file.

Classify all pixels in the image and save the estimated class labels in an image. Display this image.

Compute the classification error by counting the number of misclassified pixels in the test mask

tm_test.png (contains the true labels for the test pixels).

(We have one satellite image, and only the pixels in the training mask are used to estimate the parameters mu and sigma. Only the pixels in the test mask are used to compute the classification accuracy. We can still display the entire classified image to look at the result).

Step 3: Find the best feature for classification using single features.

Compute the percentage of correctly classified pixels for each of the features alone, and find the

accuracy for the best single feature.

Exercise 2: From feature extraction to classification

In this exercise we work with the MNIST dataset for handwritten digit recognition. We compare classifying the original images to using HOG-features, and study their robustness.

The MNIST data set consists of 60000 images for training/validation and a separate dataset for test.

The following exercise is in matlab.

First download and uncompress the images and the matlab scripts to read MNIST.

1. Load the training data using the function loadMNISTLabels and loadMNISTImages:

   mnisttrainlab = loadMNISTLabels('train-labels.idx-ubyte');
   mnisttrainim = loadMNISTImages('train-images.idx3-ubyte');

   Each sample consists of ONE IMAGE, and we will classify ONE IMAGE as one sample.

2. Use the first 30000 samples as training and the remaining 30000 samples as validation data to test different models on.
3. The MNIST images are small images of size 28x28, but organized as vectors of length 784. Display the first 10 images using imshow (Hint: use reshape(…,28,28) to convert back to 2D images).
4. On this dataset, using a classifier based on variance will give trouble – why?
   Hint: compute the variance of e.g. pixel 1 in the dataset. Will using this give numerical problems?

5. First, we will classify the images by considering each pixel as a feature, so the feature vector will have dimension 784 (28x28 features). We will use a simple minimum Euclidean distance classifier (equivalent to assuming that all the classes have common unit variance). Minimum distance classifiers means that each image should be compared to the class mean using Euclidean distance in the 784-dimensional feature space. Classify each image to the class that has the smallest distance from the sample to the class mean.
   - To do this:
   - Find all samples in the training data set for each class.
   - Compute the class means as vectors of length 784.
   - For each sample in the validation data:
     - Compute the Euclidean distance to all class means.
     - Assign the sample to the class that has the smallest distance.
   - Compute the number of correctly classified images in the validation data set.

     Try to understand why we get a quite low classification accuracy. Look at the mean images (the mean vectors can be converted to images), and think of what the minimum distance classifier compares each image to.

6. Extract HOG-features using the function extractHOGFeatures. See the matlab documentation for examples.
   [featureVector,hogVisualization] = extractHOGFeatures(currimage, ‘CellSize’, [cellsize cellsize]);
   Experiment wth cellsize for a couple of training images, and use plot(hogVisualization) to find a cell size that you think can be used to discriminate between the numbers. Look at visualizations for images for different classes.

7. When you have found a good cell size, repeat the classification of the validation data set with HOG-features and compare the accuracy to using the original pixels as features as we did above. You need to train on HOG-features too.

   Now you should see a large increase in classification accuracy on the validation data set!

   Are both the original images and the HOG-features equally sensitive to translation?

8. Let us now try some other data set of handwritten digits. Load the file imarr.mat. This file contains 100 small handwritten test images, image 1-10 has label 1, 11-20 label 2,…. And 91-100 label 0.
   Display some of these images.

9. Select some of the images, display them, classify them with orginal pixel values and HOG features. Try to understand why HOG features work better than the original pixels. Try also to understand why the classifier make a lot of errors.

10. IF TIME (Running KNN takes longer time): Compare the accuracy to using the KnearestNeighbor (KNN) classifier with K=3 using the HOG-features. Note that for the KNN-classifier, there is no training, as classfication is done to computing the Euclidean distance to all samples in the training data set, finding the 3 closest samples, and selecting the class as the majority of these 3 classes. To do this you can if you want use the functionf fitcknn and predict. Check how they work in the matlab documentation.

Notice how slow the knn-classifier is compared to the minimum distance classifier for HOG features on the validation data set.

Compute the number of correctly classified images in the validation data set with the original pixels and using HOG-features. Do you still see the same improvement in accuracy?