TTT4175 Estimation, Detection and Classification

Project descriptions: Classification 1 & 2

All students will be organized into groups of two. **The Iris task has to be done by all groups that choose this classification task**. The groups shall in addition choose to do either the "handwritten numbers" (mnist) or "vowels" task. All experiments should be implemented in MATLAB or Python, unless otherwise agreed upon.

1 The Iris Task

The Iris flower has several variants, including three called Setosa, Versicolor, and Virginica, see Figure 1. The flower has both large (Sepal) and small (Petal) leaves, see Figure 2. The three variants can be discriminated by the different lengths and widths of the petal and sepal; i.e., these four measurements are a logical choice for input features.

A database (often called "Fisher Iris data") is produced, consisting of 50 examples of each of the three variants/classes. We refer to `https://en.wikipedia.org/wiki/Iris_flower_data_set` for a more detailed description of the database.

The Iris task is one of a few practical problems that is close to linearly separable. Thus, an error-free linear classifier can be designed for the database. This first part of the project therefore focuses on the design and evaluation of a linear classifier. Further, one should analyze the relative importance of each of the four features with respect to linear separability.

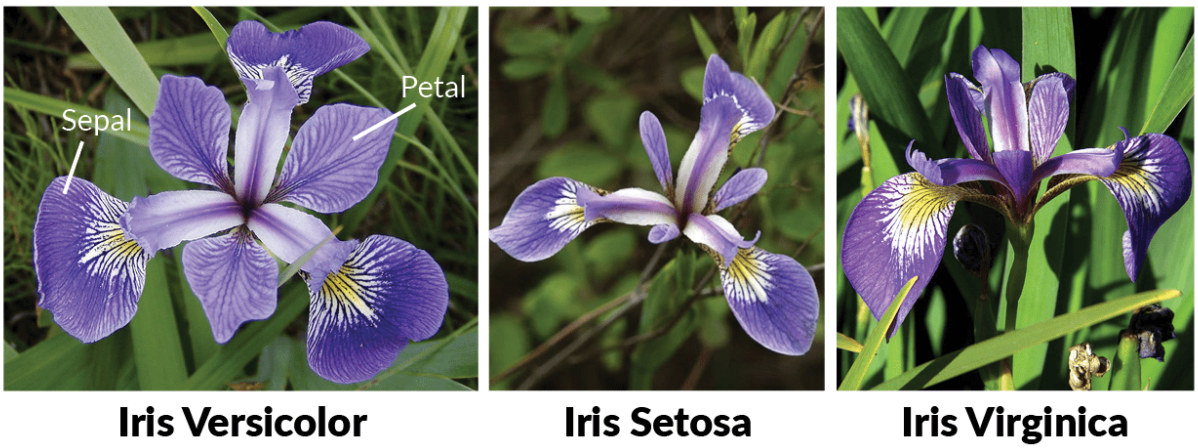


Figure 1: The three Iris variants.

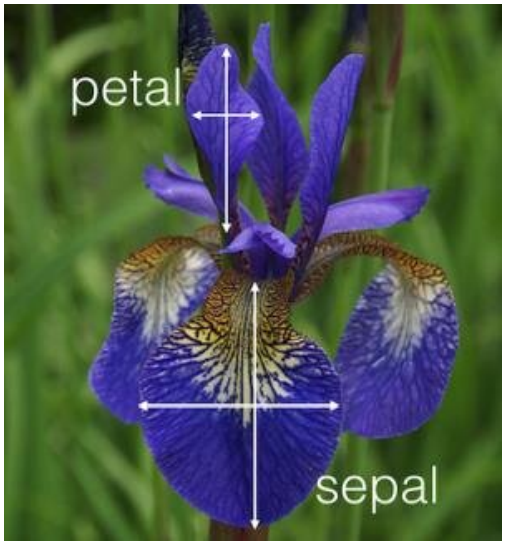


Figure 2: Length and width for petal and sepal.

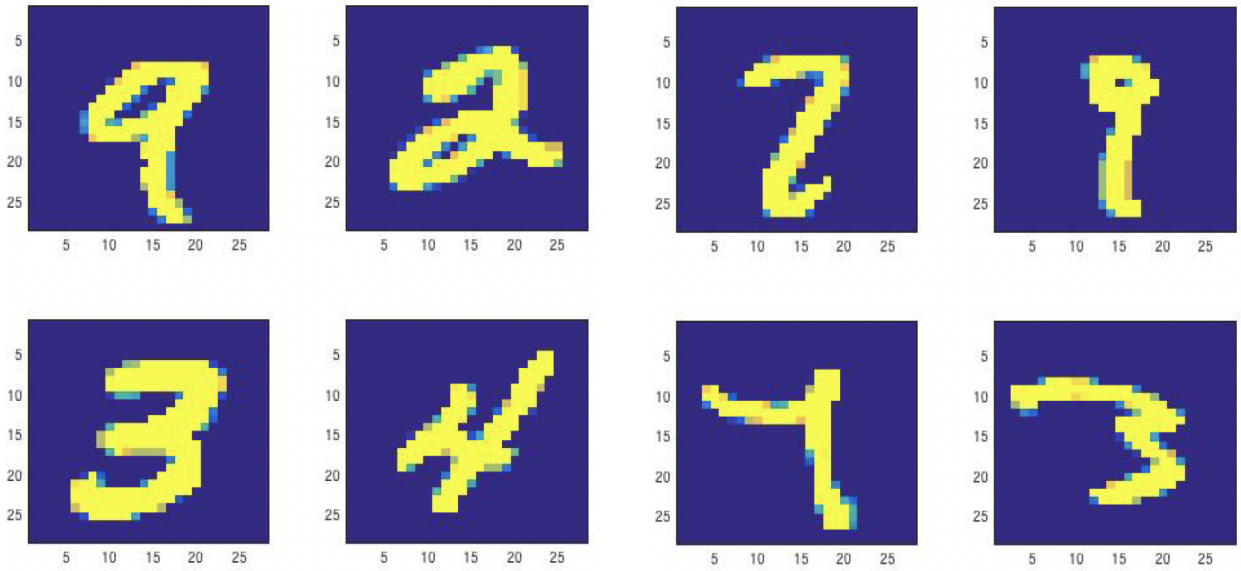
1.1 Task Description

The task consists of two parts:

1. The first part focuses on design/training and generalization.
   (a) Choose the first 30 samples for training and the last 20 samples for testing.
   (b) Train a linear classifier as described in chapters 2.4 and 3.2 of the "Compendium - Part III - Classification" on blackboard. Tune the step factor α in Equation 23, with MSE defined as in Equation 19, until the training converges.
   (c) Find the confusion matrix and the error rate for both the training and test sets.
   (d) Now use the last 30 samples for training and the first 20 samples for testing. Repeat the training and test phases for this case.
   (e) Compare the results for the two cases and comment.
2. The second part focuses on features and linear separability. In this part, the first 30 samples are used for training and the last 20 samples for testing.
   (a) Produce histograms for each feature and class. Remove the feature that shows the most overlap between the classes. Train and test a classifier with the remaining three features.
   (b) Repeat the experiment above with two and one features.
   (c) Compare the confusion matrices and the error rates for the four experiments. Comment on the properties of the features with respect to linear separability for the three separate classes.

2 Classification of Handwritten Numbers 0-9

NIST is the USA variant of the Norges Forskningsråd (NFR). NIST has designed a database called MNIST (`yann.lecun.com/exdb/mnist`), with pictures of handwritten numbers 0-9. The images have dimensions of 28×28 pixels and are in 8-bit grayscale (i.e. pixel values between 0-255). For practical purpose one should note that the the images have been "preprocessed"; i.e. centered and scaled to prepare them for classification. Figure 3a shows four clear examples, while Figure 3b shows examples which are harder to classify correctly. The database consists of 60000 training examples written by 250 different persons and 10000 test examples written by 250 other persons. A large amount of classifiers have been designed for this case, resulting in error rates between 1 and 10 %. The state-of-the-art is, of course, a deep neural network.



(a) Numbers 9, 2, 3, and 4.

(b) Nmbers 2, 9, 7, and 3.

Figure 3: Examples of (a) easier and (b) dubious handwritten numbers in MNIST.

2.1 Task Description

The task consists of two parts using different variants of the nearest-neighborhood classifier:

1. In the first part part **the whole training set** shall be used as templates.

   (a) Design a nearest neighborhood (NN)-based classifier using Euclidean distance. Find the confusion matrix and the error rate for the test set. *Note: To improve efficiency, datasets should ideally be divided into chunks of images (e.g., 1,000 images per chunk). This helps prevent overly large distance matrices and reduces computation time, compared to classifying images individually.*

(b) Plot some misclassified images.
Some useful Matlab commands for this are :

i. `x = zeros(28,28); x(:) = testv(i,:);` converts the picture vector (number i) to a 28×28 matrix.

ii. `image(x);` plots the matrix `x`.

iii. `dist(template, test);` calculates the Euclidean distance between a set of templates and a set of test vectors, both in matrix form.

Some useful Python commands for this are:

i. `import numpy as np` then `x = testv[i, :].reshape((28, 28))` converts the image vector (number i) to a 28×28 matrix.

ii. `import matplotlib.pyplot as plt` then
`plt.imshow(x, cmap='gray') plt.show()` plots the matrix `x`,.

iii. `distance = np.linalg.norm(template - test)` calculates the Euclidean distance between two vectors. For multiple vectors, you can use `cdist(template, test, metric='euclidean')` from the `scipy.spatial.distance`.

(c) Also, plot some correctly classified images and analyze them. Do you as a human disagree with the classifier for some of the correct/incorrect plots?

2. In the second part you shall use clustering to produce a small(er) set of templates for each class. The Matlab function `[id`$_{x_i}$`,C`$_i$`] = kmeans(train`$_{v_i}$`,M);` will cluster training vectors from class ω_i into M templates given by the matrix C_i. A suitable Python counterpart is `sklearn.cluster.KMeans`, which can be used as follows:

```
kmeans = KMeans(n_clusters=M, random_state=42)
id_xi = kmeans.fit_predict(train_vi)
Ci = kmeans.cluster_centers_
```

(a) Perform clustering on the 6000 training vectors for each class into $M = 64$ clusters.

(b) Find the confusion matrix and the error rate for the NN classifier using these 64 templates per class. Comment on the processing time and the performance relatively to using all training vectors as templates.

(c) Now design a KNN classifier with $K = 7$. Find the confusion matrix and the error rate and compare to the two other systems.

3 Classification of Pronounced Vowels

In this task we will try to classify a full set of twelve vowel classes. The data set consists of 139 recordings of each vowel class. The first 70 samples shall be used for training while the remaining 69 samples shall be used for testing. From each recording several features, so called formants, are extracted which represent the frequencies where the sound have peaks in the frequency spectrum. All formants are stored in the ".dat" files but we shall only use the first three in this project. Typical formant frequencies differ from vowel to vowel, however, they also show a significant variation within the classes. Thus, we have a classical non-separable problem. The waveform files has five letter names with extension ".wav" where the first letter indicates man/female/boy/girl, the next two person number and the two last the vowel name. In classification exercise 2 a subset of three classes were analyzed.

In Figure 4 we see two waveform examples from each of the vowels "AE" and "IH". Note that the vowels (large energy) are surrounded by consonants (i.e. CVC). It is not easy to decide upon which examples belong to which class just by inspecting the waveforms.

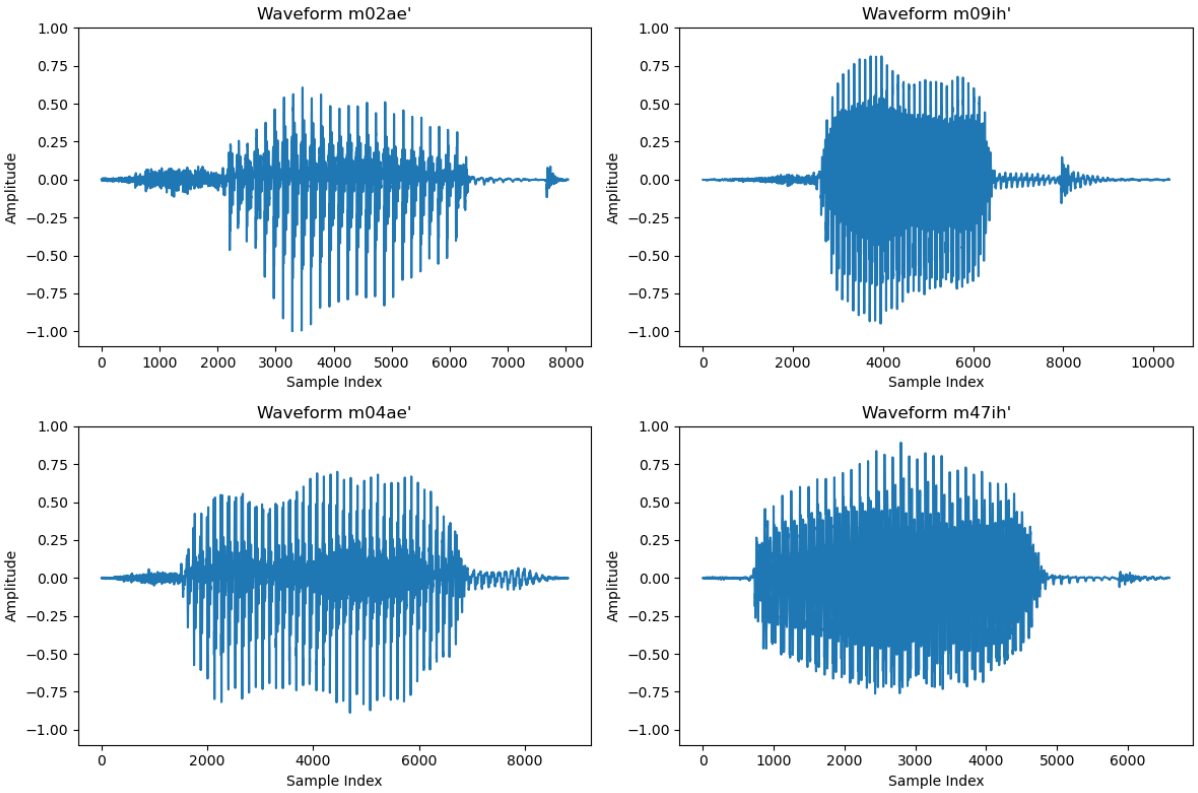


Figure 4: Examples of waveform signals for vowels "AE" and "IH".

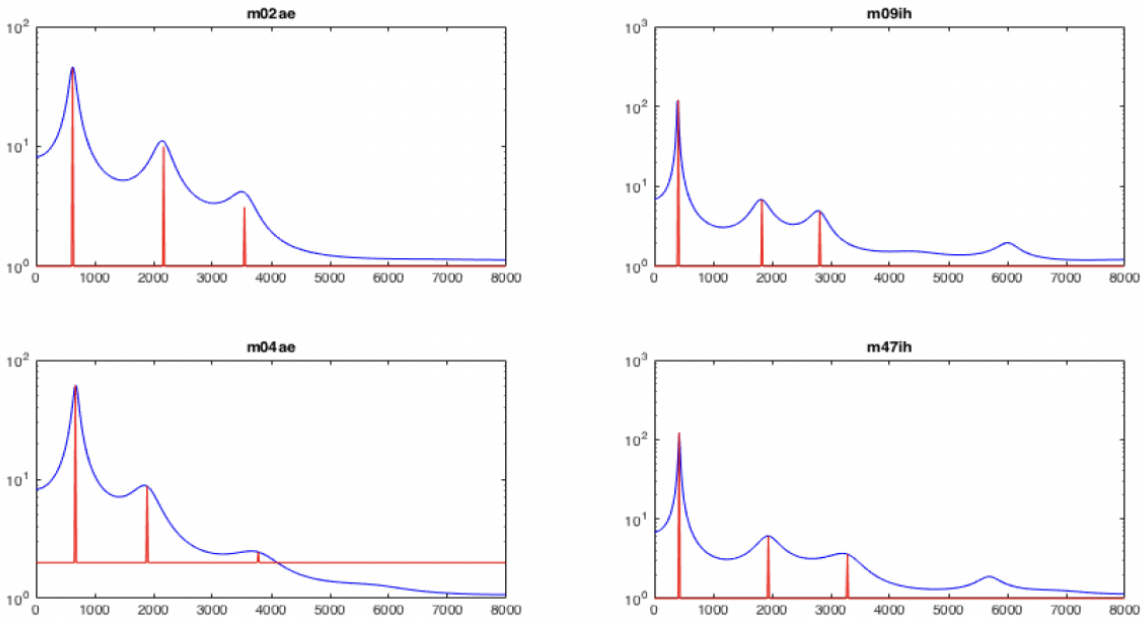


Figure 5: Examples of spectrograms and formants for vowels "AE" and "IH".

Figure 5 shows (estimates of) the frequency spectra of the vowel examples in Figure 5. We see that the spectra have clear peaks (formants), where the three first/dominant formants are indicated by red. We further see that the formants are quite similar for the same class while the first and third formant differ significantly between the classes. Thus it should be possible to use the formants to discriminate between the classes.

In Figure 6 formant histograms are plotted for the vowels "AE"and "UW" and the mean over all twelve classes. The first two formants differ especially much between the two vowels "AW" and "UW".

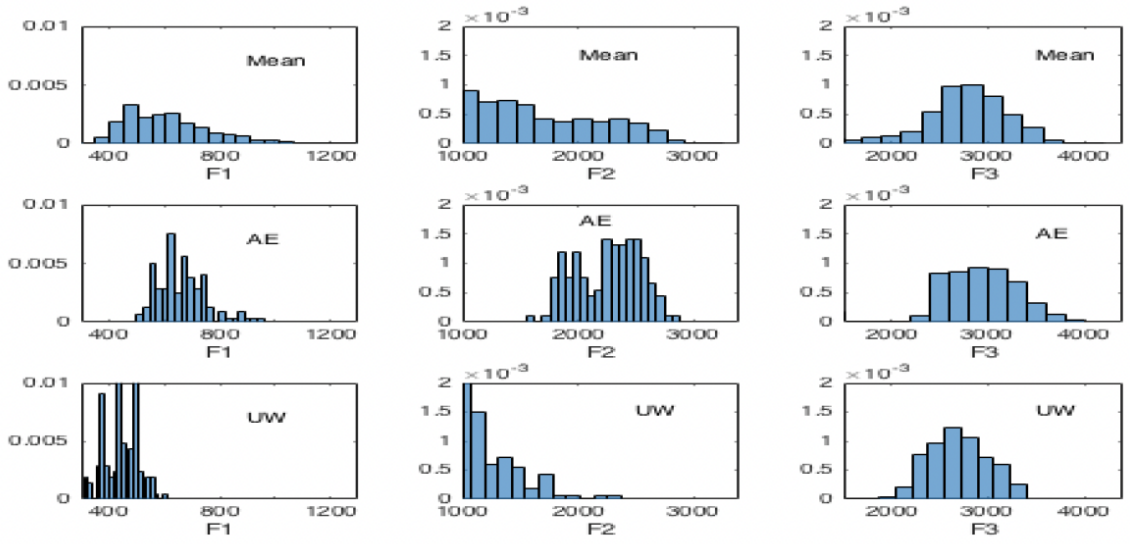


Figure 6: Formant histograms for vowels "AE" and "UW".

3.1 Task Description

The task consists of two parts which both use Gaussians as models:

1. In the first part part a single Gaussian class model with respectively a full and a diagonal covariance matrix is to be analyzed.
   (a) Find the sample mean and the sample covariance matrix for each class.
   (b) Design a classifier assuming full covariance PDFs and compute the confusion matrix and error rate for the test set.
   (c) Repeat the above with diagonal covariance matrices (i.e. set off-diagonal matrix values to zero). Compare the confusion matrices and the error rates.
2. In the second part you shall use a mixture of 2 and 3 Gaussians for each class. The Matlab function `GMM`$_i$ `= fitgmdist(trainv`$_i$`,M);` will put M mixtures into GMM$_i$ using the training vectors $trainv_i$ from class ω_i. The function `mvnpdf` will calculate the likelihoods for a test set. Another option is to use a class of function `gmmdistribution.fit` to do the training and the function `pdf` in the test phase. In corresponding python approach is to do

   ```
   from sklearn.mixture import GaussianMixture
   gmm_i = GaussianMixture(n_components=M, covariance_type='full', random_state=42)
   gmm_i.fit(trainv_i) likelihoods = gmm_i.score_samples(test_set)
   ```

   (a) Find GMM models with diagonal covariance matrices for both 2 and 3 mixtures for each class.
   (b) Modify the classifier in task 1b to deal with mixtures of Gaussians.
   (c) Find the confusion matrices and the error rates for the GMM classifier using respectively $M = 2$ and $M = 3$ Gaussians per class.
   (d) Compare the performances for all four model types (tasks 1b, 1c and 2c). For which class(es) is the difference largest?