Computervision Lab 5
Classification

David Van Hamme

1 Feature vectors

Many applications require observations to be assigned to categories or classes. For example, a doctor may look at a magnetic resonance image (MRI) of a patient's brain and classify the brain as healthy or pathological based on the prevalence or absence of certain distinctive qualities or features. In computer vision, such classification problems are very common. In order to automate the assignment of a class label to an observation, typically a series of numbers is first calculated that describe the presence of features in the observation. This set of descriptive numbers is called a *feature vector*. Features may relate to average color values, presence of edges, strength of texture, ... The feature vectors can be seen as data points in a multidimensional space.

An expert will then typically label part of the data, i.e. he will manually indicate the classes to which that data belongs. A multitude of machine learning algorithms exists to then define a set of rules which determine the most likely class label for every feature vector. Finding these rules is referred to as *training* the classifier.

2 Linear and Quadratic Discriminant Analysis

A naive way to classify multidimensional data is to determine the center of gravity of every class (the average feature vector for all members of this class) and then assign each data point to the closest class. This has the downside that it only works when the spread of the data is roughly uniform in every dimension. If one field in the feature vector is a number between 0 and 1, but another field is between 0 and 1000000, finding the closest class center will largely ignore the first field.

Linear Discriminant Analysis (LDA) overcomes this problem by calculating the distribution in each dimension, and possible correlations between variables. The data is then projected to a lower dimensional space spanned by new, independent features that are linear combinations of the original features. This space is then carved into classes by linear decision surfaces (e.g. planes in three dimensions, lines in two dimensions). An example is shown in Figure 1.

Quadratic Discriminant Analysis (QDA) is very similar, but with quadric decision surfaces. This is more powerful especially when some classes are very compact in the independent feature space while others have much larger variety in their feature vectors.

On Ufora you can find `classify_skypix.py`, a very basic pixel classifier that attempts to determine if a pixel belongs to the sky or not based on its red, green and blue values. Make sure you understand what happens:

- pixel data is labeled as white (sky), black (not sky) or do-not-care (anything else) in the masks provided for two images,
- a QDA classifier is trained on the labeled sky and non-sky pixels of these two images,
- the classifier is used to predict where the sky is in both the two training images and two non-training images.

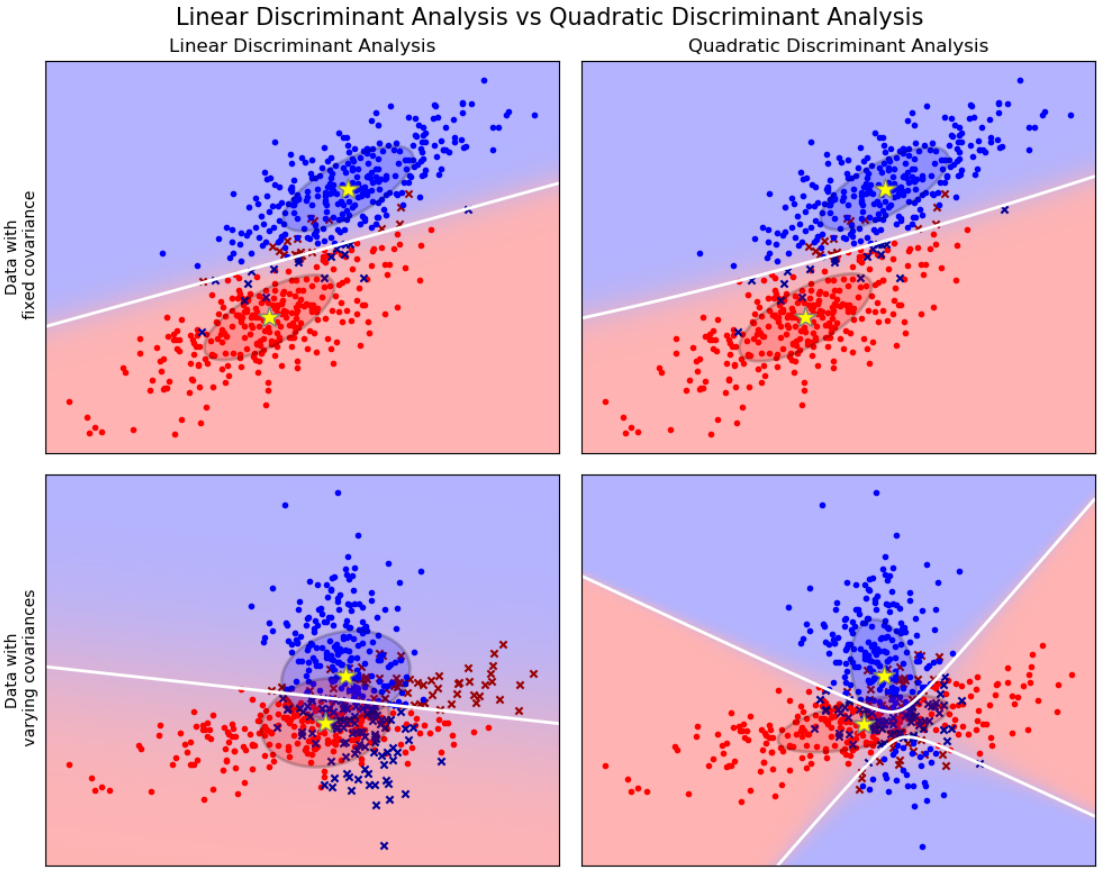

Figure 1: LDA and QDA (source: scikit-learn)

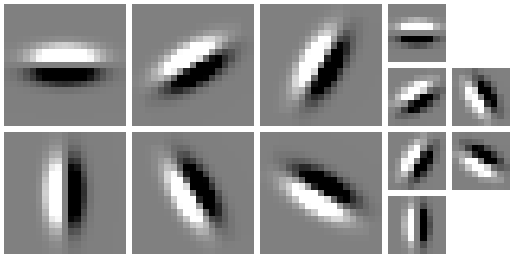

Figure 2: DoG filter bank with two scales and six orientations.

Exercise 1

The color based classifier makes many mistakes in the last two images, notably the white car and the road marks are mislabeled. Color by itself is not a powerful enough feature to segment the sky. In this exercise, you will add texture features to the classifier. Make a set of DoG (derivative of Gaussian) filters, using the code from Lab 5 exercise 10. You will use it to extend the three-dimensional feature vector in `classify_skypix.py`.

Assignment 1 Make a filterbank of DoG filters in 2 scales and 6 orientations (so 12 filters in total). Visualize the filters as in Figure 2.

Assignment 2

- Filter *road*.png* with each of the filters. This gives you 12 filter response images. Make sure they are floating point and contain negative values!
- Append the 12 filter response images to the blue, green and red channels to make a 15-channel image, from which you can extract a 15-dimensional feature vector for each pixel. If you imagine the 15 channels as a stack of images lying on top of each other, each pixel's feature vector is a vertical string of values from the stack of images.
- Train and test a new QDA classifier on the 15-dimensional feature vectors of all pixels of all four images.

Show the classification result in your report.

3 Random forest

A random forest classifier is a classifier that consists of a number of decision trees. Each decision tree makes a hierarchical set of decisions, each decision based on a simple criterion on only a few feature values, to arrive at a class label for the full feature vector. The random forest introduces random variations in the construction of the trees and takes a majority vote of their outcomes. This solves a major drawback of decision trees: their poor generalization to data slightly dissimilar from the training data. The main advantage of a random forest over a decision surface based classifier like QDA is that the class samples do not need to form a contiguous space. When there are groups of samples with the same label but scattered in different parts of the multi-dimensional space, the decision trees can carve out these parts. Discriminant analysis on the other hand will calculate the centroid of all the samples of the same label, and this centroid may be far removed from *all* the samples if the data is not contiguous!

Exercise 2

Assignment 3 Replace the QDA classifier you trained in Exercise 14 with a random forest classifier. Pay attention to the main parameters:

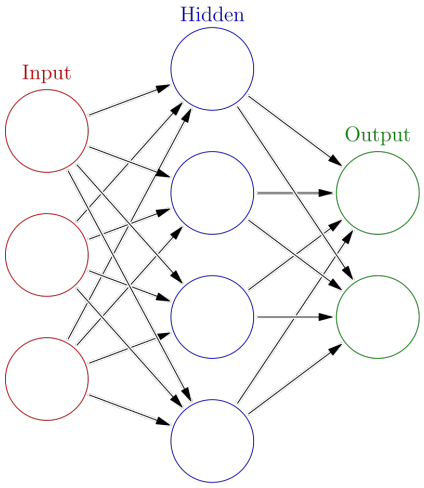

Figure 3: Basic artificial neural network structure, also called Multi-Layer Perceptron.

- the number of decision trees, controlling the overall complexity of the classifier (related to the complexity of the data),
- the minimum leaf size, which controls how many samples must be grouped together as a minimum at the end of each decision tree (usually specified as a fraction of the total data).

Show the classification result in your report.

Question 1 Does the RF classifier outperform the QDA classifier on the sky pixel classification problem?

4 Deep Learning

In the last decade, traditional machine learning classifiers have been gradually superceded by Artificial Neural Networks (ANNs or simply NNs) for complex classification problems. Neural networks are inspired by the human brain, which is an enormous collection of interconnected neurons which continuously adapt their connections to learn to solve new problems. In an ANN, each neuron has a number of inputs, and calculates its output as a weighted sum of these inputs. Several layers of neurons are typically used between the input features and the the output prediction, so the neural network is capable of computing intermediate representations (analogous to feature detection) and modeling relationships between the data at many levels (low-level features to high-level, more abstract semantic interpretations).

Neural networks need to be trained: the weights of all the interconnections must be adjusted so that the output approximates the desired result. This is achieved by a technique called backpropagation. Essentially, the gradient of each of the output nodes with respect to the final prediction is determined, i.e., the derivative of the node's output with respect to each of its input node connections. Then the gradients of each of those input nodes with respect to *their* input node connections are calculated, and this process is repeated until the gradients have been calculated up to the first layer. These gradients describe how the output changes with relation to each of the weights in the network. During each training step, the weights are adjusted with an amount proportional to this gradient so that the output evolves towards the desired prediction. To ensure the network does not get stuck into a local optimum, the steps are small, and different random samples are used to calculate the gradient in each step. This way, the network hopefully converges towards a global optimum.

4.1 Exercise 3

Assignment 4 Replace the classifier you made in Exercise 15 with a neural network. You can use the `MLPClassifier` from sklearn. Pay special attention to the following parameters:

- number of layers,
- number of neurons in each layer,
- learning rate of the optimizer,
- size of the training batches,
- number of training epochs.

Ensure that your network has converged to a global optimum by plotting the loss as the network trains and observing whether it has flattened out to a stable value. Not that the training process is in many ways stochastic, and multiple runs with identical parameters do not necessarily produce the same network! Show the classification result in your report.

Question 2 How does this classifier perform compared to the QDA classifier you made earlier? Do you see overfitting, i.e., good performance on the training data but poor performance on unseen data?

Question 3 Analyse the remaining errors in the prediction of your three classifiers. Where are the errors mostly located? Can you think of a simple extra feature that may help classification?