© N. Krasowski, L. Cerrone, U. Köthe
ullrich.koethe@iwr.uni-heidelberg.de

Exercise 2

Deadline: 15.11.2017, 10:00

This time, we are focusing on the implementation and evaluation of linear/quadratic discriminant analysis (LDA/QDA) and useful visualization techniques. We will work on the digits dataset again.

Regulations

Please hand-in your solutions as a jupyter notebook `qda-lda.ipynb`. Also export your notebook into a PDF file named `qda-lda.pdf`. Zip all files into a single archive with naming convention `name1_name2_exercise2.zip` and upload it to Moodle before the given deadline. Remember that you have to reach 50% of the homework points to be admitted to the final mini-research project.

1 Data Preparation

As in exercise 1 we will try to distinguish digits from the dataset provided by sklearn. Reminder:

```
from sklearn.datasets import load_digits

digits = load_digits()

print digits.keys()

data          = digits["data"]
images        = digits["images"]
target        = digits["target"]
target_names  = digits["target_names"]
```

We will once again work with digits "1" and "7". Please filter the dataset such that only these two digits are left. The filtered dataset should have 361 instances. Split this filtered dataset in a training and a test set (#train/#test = 3/2).

1.1 Dimension Reduction (4 points)

To make classification harder (and to simplify the visualization of the feature space), you are supposed to restrict yourself to 2 feature dimensions. The decision on how to define these features is completely up to you. You can, for example, choose two pixels that seem to have a big influence. To identify suitable pixels, you may want to look at the average images for the two classes – pixels that tend to be bright in one class and dark in the other are good candidates. You can also come up with some clever combination of the 64 original pixel values into 2 features, for example: $\tilde{f}_1 = 0.3 f_{23} + 42 \frac{f_{13}}{f_{64}}$ and $\tilde{f}_2 = f_{33} - f_{62}$. You may only work with your two features in the subsequent tasks. Note that the quality of your features determines the intrinsic error and therefore is a limiting factor for the quality of your predictions. (But don't be too clever either: if all classes form circular clusters in your feature space, the rest of the exercise will be quite boring. :-) Your dimension reduction procedure should be callable through a function `reduce_dim`:

```
reduced_x = reduce_dim(x)
```

where `x` is a #instances × 64 matrix and `reduced_x` has shape #instances × 2.

1.2 Scatterplot (4 points)

To decide whether your two new features are suitable for the classification task, you should visually inspect the distribution of the instances in the new feature space by means of a *scatter plot*, which you can produce with the function `matplotlib.pyplot.scatter`. The plot axes are the two feature

dimensions, and each training instance is placed at the appropriate location. To distinguish the classes, use two different markers. Check that the classes don't overlap too much and search for better features if they do. A good plot must be informative and easy to understand. Optimize your plot by choosing good values for the scatter function parameters (marker, color, size etc., see `https://matplotlib.org` for more options).

2 Nearest Mean

2.1 Implement the nearest mean classifier (3 points)

Implement the nearest mean method: find the mean of the 2D feature vectors of each class in the training set and assign the label of its nearest mean to each test instance. The signature of the function is supposed to look like this:

```
predicted_labels = nearest_mean(training_features, training_labels, test_features)
```

where `training_features` and `test_features` are the outputs of `reduce_dim()` for training and test data respectively.

2.2 Visualize the decision regions (4 points)

An image of the decision regions can be quite helpful for analyzing classifier performance. The simplest way to do this is to create a grid (i.e. an image) where each pixel position represents a feature coordinate, and the pixel is colored with the corresponding predicted class label. Since you created the features on your own, you must find a sensible transformation (i.e translation and scaling) from feature coordinates to grid coordinates so that your image covers the interesting part of the feature domain. A 200×200 grid provides sufficient resolution. Plot this image to visualize the decision regions. Overlay the plot with two markers for the two class means and with a scatterplot of the test data, marked according to their ground truth labels as in task 1.2. Consult the matplotlib documentation to find out how several layers of information can be overlayed in the same plot.

3 QDA

3.1 Implement QDA Training (6 points)

Implement the function:

```
mu, covmat, p = fit_qda(training_features, training_labels)
```

that accepts a $N \times D$ matrix `training_features` and a N-dimensional vector `training_labels`, where N is the total number of training instances used. Due to dimensionality reduction via `reduce_dim()`, we have $D = 2$ here, but your code should work for arbitrary values of D. Note that the features x_i are the *rows* of matrix `training_features`, in contrast to the lecture, where we defined the x_i to be column vectors. The formulas from the lecture must be adapted accordingly. The label vector should use label 0 to indicate digit "1" and label 1 to indicate digit "7".

The output should be the $2 \times D$ matrix `mu` whose rows are the two class means, the $2 \times D \times D$ array `covmat` containing the two covariance matrices and the vector `p` containing the two priors. Apply the fit function to your training data from task 1.1.

3.2 Implement QDA Prediction (3 points)

Now, using the output of `fit_qda()` implement a function:

```
predicted_labels = predict_qda(mu, covmat, p, test_features)
```

which uses QDA to predict the labels (a M dimensional vector) for a $M \times D$ matrix `test_features` of test instances. Apply the function separately to your training and test data and compute the training and test error rates respectively.

3.3 Visualization (5 points)

Create a grid for the QDA decision regions and plot it as in task 2.2. Overlay this plot with a scatter plot of the *training data*. Then add another overlay that visualizes the cluster shape of the two Gaussian distributions in terms of representative isocontours (i.e. ellipses). Check out the documentation for matplotlib's `contour` function to learn how to do this.

Furthermore, perform an eigenvalue/eigenvector decomposition of the two covariance matrices. Recall that the eigenvectors indicate the directions of the principal cluster axes, and the eigenvalues' square roots are the corresponding standard deviations, i.e. they indicate the cluster radii along each axis. Add another overlay to your plot that draws these axes (centered at the cluster mean) in form of lines whose length equals the standard deviation.

Use this plot and your estimated training error to rate the quality of the QDA results on the training data. You'll probably observe some misclassifications. Where do training errors in QDA classification stem from (compared with nearest neighbor classification, which always has zero training error)?

3.4 Performance evaluation (3 points)

Now we want to get an idea of how well the QDA classifier generalizes to unseen data. In order to estimate the test error, apply 10-fold cross validation to the QDA classifier. Take advantage of sklearn's cross validation function (`http://scikit-learn.org/stable/modules/cross_validation.html`) for splitting your data.

4 LDA (8 points)

Repeat all tasks of assignment 3 for LDA, i.e. implement functions `fit_lda()` and `predict_lda()`, apply them to your data, visualize the decision boundary with overlays for training points and cluster shape, and perform performance evaluation. How does the prediction quality change relative to QDA and the nearest mean classifier?