Niko Krasowski
niko.krasowski@iwr.uni-heidelberg.de

Exercise 2

Deadline: 03.11.2014

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

1 Data Preparation (1 point)

As in exercise 1 we will try to distinguish digits from the digits 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 try to distinguish 1 from 7. Please filter the dataset such that only these two digits are left. The filtered dataset is supposed to have 359 examples. Split this filtered dataset again in a training- and a testset (#train/#test = 3/2).

1.1 Dimension Reduction (2 points)

To make your life harder (and to simplify the visualization of the featurespace) you are supposed to restrict yourself to 2 feature-dimensions. The decision on how to choose the features is completely up to you. You can for example do this by choosing two pixels, that seem to have a big influence. Therefore you may want to have a look at the accumulated images of each class or their difference image. You can also come up with a clever combination of the 64 features that results in 2 features ($\hat{f}_i$). E.g.: $\hat{f}_1 = 0.3 f_{23} + 42\frac{f_{13}}{f_{64}}$ and $\hat{f}_2 = f_{33} - f_{62}$. For the following exercises on this sheet you are supposed to work only with your two features. Note that the quality of your features determines the intrinsic error and therefore is a limiting factor for the quality of the predictions. Your dimension reduction procedure should be callable through a function `dr`:
`xn = dr(x)`
where `x` is a $\#samples \times 64$ matrix and `xn` has the shape $\#samples \times 2$

1.2 Scatterplot 1 (1 point)

To decide whether your two new features are suitable for the classification task please inspect the distribution of the samples in the now two dimensional featurespace visually. Therefore draw one point per trainingsample in the featurespace. The coordinates of the points correspond to their featurevalues. You can produce such a scatterplot via:

```
import matplotlib.pyplot as plt
import numpy as np

x1 = np.random.randn(200)
y1 = np.random.randn(200)
x2 = np.random.randn(200)
y2 = np.random.randn(200)

plt.title('Scatter Plot')
plt.xlabel('X axis')
plt.ylabel('Y axis')

size = 20
# b: blue,    g: green,  r: red,   c: cyan,
# m: magenta, y: yellow, k: black, w: white
plt.scatter(x1,y1, marker="x", c="r", s=size)
plt.scatter(x2,y2, marker="o", c="b", s=size)
plt.show()
```

Distinguish the different classes by different symbols (x and o). If the samples of the different classes do overlap too much you might want to think about your features again.

2 Nearest Mean (2 points)

Implement a nearest mean classifier. Therefore find the mean of the feature vectors of each class in the training set and assign each testpoint the label of its nearest mean. The signature of the function is supposed to look like this:
`testy = nm(trainingx, trainingy, testx)`

3 QDA

3.1 Implementing QDA - Training (4 points)

As a first step, implement a python function
`mu0, mu1, covmat0, covmat1, p0, p1 = compute_qda(trainingy, trainingx)`
that accepts a d dimensional vector trainingy of input labels (that must be either 0 or 1) and a $n_1 \times d$ matrix trainingx of features. The output should contain the d dimensional mean vectors, the $d \times d$ covariance matrices and the priors p_0 and p_1 as scalars.

3.2 Implementing QDA - Test (3 points)

Now, using the output from the previous section implement a function
`qda_prediction = perform_qda(mu0, mu1, covmat0, covmat1, p0,p1, testx)`
which predicts the labels (a n_2 dimensional vector) of a $n2 \times d$ dataset testx using QDA.

3.3 Scatterplot 2 (1 point)

Apply the QDA prediction to the **training** data (the one that you constructed in 1 and 1.1 that has two feature dimensions and only labels 1 and 7), compute the correct classification rate and visualize the results: A good way is to make a 2D plot of the data with regard to the two scores again. Mark true predictions of class (1,7) with different colors and as is 1.2 distinguish the classes (estimated by qda) with different symbols.

3.4 Visualize the decision boundary (2 points)

It can be quite insightful to obtain an image of the decision boundary. Therefore, create a sensible grid of input values for the two scores (since you created the features on your own you will have to adapt the grid such that it covers your whole feature-domain) and perform the QDA for each combination of inputs. A 100×100 grid provides a sufficient resolution. Make a plot to visualize the decision boundary. Using this plot and your results from the previous section to inspect the quality of the QDA results on the **training** data. You should observe some misclassifications. Where do misclassifications on training data using QDA stem from (compared for example with NN, that doesn't have any misclassifications on training data)?

3.5 Prediction (2 points)

Now we want to get an idea of the generalization to unseen data. Therefore apply the QDA to the **test** data and compute the correct classification rate (we are still working on the 1 against 7 task). Make a similar 2D plot as in 3.3.

4 LDA (2 points)

Modify the training of your classifier in 3.1 in a way that it results in LDA:
`mu0, mu1, covmat0, covmat1, p0, p1 = compute_lda(trainingy, trainingx)`
How does the prediction quality suffer? You should then be able to analyze this classifier similarly to the analyses of QDA in 3.2, 3.3, 3.4, 3.5. Reuse your functions from the QDA case.

Regulations

Please hand in the python code, figures and explanations (describing clearly which belongs to which). Non-trivial sections of your code should be explained with short comments, and variables should have self-explanatory names. Plots should have informative axis labels, legends and captions. Please enclose all results into a single .pdf document and hand in the .py files that created the results. Please email the solutions to `niko.krasowski@hci.iwr.uni-heidelberg.de` before the deadline. You may hand in the exercises in teams of maximally three people, which must be clearly named on the solution sheet (one email is sufficient). Discussions between different teams about the exercises are encouraged, but the code must not be copied verbatim (the same holds for any implementations which may be available on the WWW). Please respect particularly this rule, otherwise we cannot give you a passing grade. Solutions are due by email at the beginning of the next exercise. For each exercise there will be maximally 20 points assigned. If you have 50% or more points in the end of the semester you will be allowed to take the exam.