

Information Retrieval in High Dimensional Data

Lab #2, 26.10.2017

## Statistical Decision Making

Task 1. Consider the two-dimensional, discrete random variable  $X = [X_1 \ X_2]^\top$  subjected to the joint probability density  $p_X$  as described in the following table.

$p_X(X_1, X_2)$	$X_2 = 0$	$X_2 = 1$
$X_1 = 0$	0.4	0.3
$X_1 = 1$	0.2	0.1

- Compute the marginal probability densities  $p_{X_1}, p_{X_2}$  and the conditional probability  $P(X_2 = 0 | X_1 = 0)$  as well as the expected value  $\mathbb{E}[X]$  and the covariance matrix  $\mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^\top]$ .
- Write a PYTHON function `toyrnd` that expects the positive integer parameter `n` as its input and returns a matrix `X` of size `(2, n)`, containing `n` samples drawn independently from the distribution  $p_X$ , as its output.
- Verify your results in a) by generating 10000 samples with `toyrnd` and computing the respective empirical values<sup>1</sup>.

Task 2. The MNIST training set consists of handwritten digits from 0 to 9, stored as PNG files of size  $28 \times 28$  and indexed by label. Download the provided ZIP file from Moodle and make yourself familiar with the directory structure.

- Grayscale images are typically described as matrices of `uint8` values. For numerical calculations, it is more sensible to work with floating point numbers. Load two (arbitrary) images from the database and convert them to matrices `I1` and `I2` of `float64` values in the interval  $[0, 1]$ .
- The matrix equivalent of the euclidean norm  $\|\cdot\|_2$  is the *Frobenius* norm. For any matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , it is defined as

$$\|\mathbf{A}\|_F = \sqrt{\text{tr}(\mathbf{A}^\top \mathbf{A})}, \quad (1)$$

---

<sup>1</sup>Unless stated otherwise, we are working with the *biased* estimator  $\frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - (\frac{1}{n} \sum_{j=1}^n \mathbf{x}_j))(\mathbf{x}_i - (\frac{1}{n} \sum_{j=1}^n \mathbf{x}_j))^\top$  of the covariance

where  $\text{tr}$  denotes the trace of a matrix. Compute the distance  $\|\mathbf{I}_1 - \mathbf{I}_2\|_F$  between the images  $\mathbf{I}_1$  and  $\mathbf{I}_2$  by using three different procedures in PYTHON:

- Running the `numpy.linalg.norm` function with the 'fro' parameter
  - Directly applying formula (1)
  - Computing the euclidean norm between the vectorized images
- c) In the following, we want to solve a simple classification problem by applying *k-Nearest Neighbours*. To this end, choose two digit classes, e.g. 0 and 1, and load `n_train = 500` images from each class to the workspace. Convert them according to subtask a) and store them in vectorized form in the matrix `X_train` of size  $(784, 2 \cdot n_{\text{train}})$ . Provide an indicator vector `Y_train` of length  $2 \cdot n_{\text{train}}$  that assigns the respective digit class label to each column of `X_train`. From each of the two classes, choose another set of `n_test=10` images and create the according matrices `X_test` and `Y_test`. Now, for each sample in the test set, determine the `k = 20` training samples with the smallest Frobenius distance to it and store their indices in the  $(2 \cdot n_{\text{test}}, k)$  matrix `NN`. Generate a vector `Y_kNN` containing the respective estimated class labels by performing a majority vote on `NN`. Compare the result with `Y_test`.

## Helpful Numpy functions

Required packages: numpy (np), scipy

<code>scipy.misc.imread(path)</code>	import image from path as uint8-array
<code>np.array(x)</code>	turn x into a numpy array
<code>np.linalg.norm(x)</code>	norm of an array x, different norms possible
<code>np.dot(x, y)</code>	computes matrix multiplication arrays x and y
<code>np.sqrt(x)</code>	computes square root of x
<code>np.trace(x)</code>	computes matrix trace of x
<code>np.sum(x, axis)</code>	sums entries of array over axis x
<code>np.argsort(x)</code>	returns indices required to sort array x by size
<code>np.zeros(shape)</code>	generates array of all zeros of a given shape
<code>np.ones(shape)</code>	generates array of all ones of a given shape
<code>np.random.rand(shape)</code>	generate array of random numbers
<code>np.reshape(x, shape)</code>	reshape array x to a given shape
<code>np.ravel(x, 'F')</code>	returns a flattened array, stack columns
<code>np.expand_dims(x, axis)</code>	add dimension to array
<code>np.concatenate((x,y))</code>	concatenates two arrays
<code>np.vstack((x,y))</code>	vertically stack two arrays