TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Datenverarbeitung
PD Dr. Martin Kleinsteuber

`Information Retrieval in High Dimensional Data`

`Lab #2`, 19.04.2018

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

a) Compute the marginal probability densities p_{X1}, p_{X2} and the conditional probability $P(X_2 = 0|X_1 = 0)$ as well as the expected value $\mathbb{E}[X]$ and the covariance matrix $\mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^\top]$.

b) 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.

c) Verify your results in a) by generating 10000 samples with `toyrnd` and computing the respective empirical values[1].

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

a) 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 (abitrary) images from the database and convert them to matrices `I1` and `I2` of `float64` values in the interval $[0, 1]$.

b) 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

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

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

where tr denotes the trace of a matrix. Compute the distance $\|\mathbf{I}_1 - \mathbf{I}_2\|_F$ between the images `I1` and `I2` 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*n_train)`. Provide an indicator vector `Y_train` of length `2*n_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*n_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), imageio

`imageio.imread(path)`	import image from path as uint8-array
`np.dot(x, y)`	computes matrix multiplication arrays x and y
`np.sqrt(x)`	computes square root of x
`np.trace(x)`	computes matrix trace of x
`np.sum(x, axis)`	sums entries of array over axis x
`np.argsort(x)`	returns indices required to sort array x by size
`np.zeros(shape)`	generates array of all zeros of a given shape
`np.ones(shape)`	generates array of all ones of a given shape
`np.random.rand(shape)`	generate array of random numbers
`np.reshape(x,shape)`	reshape array x to a given shape
`np.ravel(x)`	returns a flattened array
`np.expand_dims(x, axis)`	adds dimension to array
`np.concatenate((x,y))`	concatenates two arrays
`np.vstack((x,y))`	vertically stack two arrays