Machine Intelligence 2 **Exercise Sheet 2**
SoSe 2021, Obermayer/Kashef due: 2020-05-03 23:55

PCA: batch preprocessing and online-PCA

Exercise T2.1: Batch vs. online-PCA (tutorial)

(a) Identify disadvantages of batch PCA that can be solved using an online PCA method.

(b) How do you find the first PC using Hebbian Learning?

(c) Why do we need Oja's rule?

(d) How do we find all other PCs?

(e) Identify different ways for constructing a novelty filter?

Exercise 2.1: Preprocessing (1 point)

(a) Load the dataset `pca2.csv`. Compute the Principal Components PC1 and PC2 and plot the data in the coordinate system PC1 vs. PC2. – What do you observe?

(b) Remove observations 17 and 157 (subtract 1 for zero-based indexing) and redo the above. – What is the difference?

Exercise 2.2: Whitening (3 points)

(a) Load the dataset `pca4.csv` and visually inspect for outliers in the individual variables. Come up with a simple heuristic to remove the outliers from the data and proceed with using the "cleaned" data in all the next steps.

(b) Perform PCA on a reasonable[1] subset of this data. Use a scree plot to determine how many PCs represent the data well.

(c) "Whiten" the data, i.e. transform the data into 4 new *uncorrelated-unit-variance* variables. Each new variable should have *mean 0* and a *standard deviation equal to 1*. This can be done e.g. using the transformation

$$\underline{\mathbf{V}} = \underline{\mathbf{\Lambda}}^{-1/2}\underline{\mathbf{M}}^{\top}\underline{\mathbf{X}},$$

where,

the new variables v_i form the rows of $\underline{\mathbf{V}}$,

$\underline{\mathbf{M}}$ is a matrix containing in its columns the normalized eigenvectors of the covariance matrix $\underline{\mathbf{C}}$ of the centered data $\underline{\mathbf{X}} \in \mathbb{R}^{N\times p}$

[1] The reason for leaving this choice up to you is for you to find a good tradeoff between speed (small subset) vs. robust/stable results (large subset but slow). Hint: Start small then double the size of the subset, if results don't change, i.e. are stable, the subset is reasonably large.

and $\underline{\mathbf{\Lambda}}$ is a diagonal matrix containing the corresponding eigenvalues[2].

(d) Generate 3 heat plots of the

(i) 4×4 covariance matrix $\underline{\mathbf{C}}$,

(ii) the covariance matrix of the data projected onto PC1-PC4 (i.e. $\underline{\mathbf{M}}^\top \underline{\mathbf{X}}$), and

(iii) of the whitened variables v_i.

[2]If your data is stored in a $p \times N$matrix, you can use $\underline{\mathbf{V}} = \underline{\mathbf{M}}^\top \underline{\mathbf{X}}\underline{\mathbf{M}}\underline{\mathbf{\Lambda}}^{-1/2}$, which will give you a $p \times 4$ matrix for $\underline{\mathbf{V}}$.

Exercise 2.3: Oja's Rule: Derivation (2 points)

Consider a linear connectionist neuron whose output $y = y(t)$ at time t is an inner product of the N-dim input vector $\underline{\mathbf{x}} = \underline{\mathbf{x}}(t)$ with the N-dim weight vector $\underline{\mathbf{w}}$:

$$y = \underline{\mathbf{w}}^\top \underline{\mathbf{x}}.$$

The Hebbian update rule for learning the weights can be written as

$$w_i(t+1) = w_i(t) + \varepsilon y(t) x_i(t), \quad i = 1, 2, ..., N$$

where ε is the learning rate and t is the iteration step. As was shown in the lecture, the Hebbian learning rule leads to a divergence of the length of the weight vector. Therefore, the following *explicit normalization* was introduced by Oja:

$$w_i(t+1) = \frac{w_i(t) + \varepsilon y(t) x_i(t)}{\left(\sum_{j=1}^{N} \left[w_j(t) + \varepsilon y(t) x_j(t) \right]^2 \right)^{\frac{1}{2}}} \qquad (*)$$

Task: Derive an approximation to this update rule for a small value of the learning rate parameter ε by Taylor-expanding the right hand side of the equation (*) with respect to ε around $\varepsilon = 0$. Show that neglecting terms of second or higher order in ε gives *Oja's rule*:

$$w_i(t+1) = w_i(t) + \varepsilon y(t) \left[x_i(t) - y(t) w_i(t) \right].$$

Exercise 2.4: Oja's Rule: Application (4 points)

The file `data-onlinePCA.txt` contains observations from an artificial experiment run over an interval of time (i.e. the first datapoint was observed at $t_0 = 0$ and the last at $t_T = 10s$).

(a) Produce a scatter plot of the data and indicate the time index by the color of the datapoints. You can break the full dataset into 10 blocks corresponding to 1 second length each and use 10 different colors, 1 color for each block..

(b) Determine the principal components (using batch PCA) for *each of the 10 blocks* separately. Plot the first PC ofr each block (e.g. as an arrow or the endpoint of it) together with the original data.

(c) Implement Oja's rule and apply it with a learning rate parameter $\varepsilon \in \{0.002, 0.04, 0.45\}$ to the dataset. In each iteration, take the next[3] data point and apply the learning step. Plot the weights at each timestep (as points whose x vs. y coordinates are given by the weight for x and y) in the same plot as the original data (use the colors from (a) to indicate the time index for each plotted weight).
Interpret your results.

Total 10 points.

[3] In contrast to the algorithm of the lecture where datapoints are sampled randomly here we do not want to lose the temporal relation between the data points and therefore process the data in the original order.