CSCI 567, Spring'18
Michael Shindler

Programming Set #5

Due: 11:59 pm, April 29, 2018

General instructions

Due date and time April 29th, 11:59pm

Starting point Your repository will have now a directory 'homework5/'. Please do not change the name of this repository or the names of any files we have added to it. Please perform a `git pull` to retrieve these files.

High Level Descriptions

0.1 Tasks You will be asked to implement (1) hidden Markov models' (HMM) inference procedures and (2) Principal Component Analysis (PCA). Specifically, you will

- **For Problem 1**, finish implementing the function `forward`, `backward`, `seqprob_forward`, `seqprob_backward`, and `viterbi`. Refer to `hmm.py` for more information.
- Run the scripts `hmm.sh`, after you finish your implementation. `hmm.sh` will output `hmm.txt`.
- Add, commit, and push `hmm.py` and `hmm.txt`.
- **For Problem 2**, finish implementing the functions `pca`, `decompress` and `reconstruction_error`. Refer to `pca.py` for more information.
- Run `pca.py` (i.e., do `python3 pca.py`), after you finish your implementation, it will generate `pca_output.txt`.
- Add, commit, and push `pca.py` and `pca_output.txt`.

0.2 Dataset In Problem 1, we will play with a small hidden Markov model. The parameters of the model are given in `hmm_model.json`.

In Problem 2, we will use a subset of MNIST dataset that you are already familiar with. As a reminder, MNIST is a dataset of handwritten digits and each entry is a 28x28 grayscale image of a digit. We will unroll those digits into one-dimensional arrays of size 784. Please download the data `mnist_subset.json` from Piazza (in the Homework section of Resources) and put it into the directory 'homework5/'. **Please DO NOT commit and push the data `mnist_subset.json`. Otherwise, you will get a (unspecified) deduction of your score**.

0.3 Cautions Please **DO NOT** import packages that are not listed in the provided code. Follow the instructions in each section strictly to code up your solutions. **Do not change the output format**. **Do not modify the code unless we instruct you to do so**. A homework solution that does not match the provided setup, such as format, name, initializations, etc., **will not** be graded. It is your responsibility to **make sure that your code runs with the provided commands and scripts on the VM**. Finally, make sure that you **git add, commit, and push all the required files**, including your code and generated output files.

0.4 Final Submission Add, commit, and push `hmm.py`, `pca.py` and the files `hmm.txt` and `pca_output.txt` that you have generated. **Please DO NOT commit and push the data `mnist_subset.json`. Otherwise, you will get a (unspecified) deduction of your score**.

Problem 1 Hidden Markov Models

In this problem, you are given parameters of a small hidden Markov model and you will implement three inference procedures, similar to what you will or have seen in Problem Set 5. In `hmm_model.json`, you will find the following model parameters:

- π: the initial probabilities, $\pi_i = P(Z_1 = s_i)$;
- A: the transition probabilities, with $a_{ij} = P(Z_t = s_j | Z_{t-1} = s_i)$;
- B: the observation probabilities, with $b_{ik} = P(X_t = o_k | Z_t = s_i)$.

Now we observe a sequence O of length L. Your task is to write code to compute probabilities and infer about the possible hidden states given this observation. In particular, first in **1.1** and **1.2**, we want to compute $P(x_1, \ldots, x_L = O)$, the probability of observing the sequence. You should use the forward algorithm and the backward algorithm to achieve that. Then in **1.3**, we infer the most likely state path. Note that your code might be tested against different parameter sets/observation sequences at grading time.

1.1 Please finish the implementation of the functions `forward()` and `backward()` in `hmm.py`:

- `forward(`π, A, B, O`)` takes the model parameters π, A, B and the observation sequence O as input and output a numpy array α, where $\alpha[j, t] = \alpha_t(j) = P(Z_t = s_j, x_{1:t})$.
- `backward(`π, A, B, O`)` takes the model parameters π, A, B and the observation sequence O as input and output a numpy array β, where $\beta[j, t] = \beta_t(j) = P(Z_t = s_j, x_{t+1:T})$.

1.2 Now we can calculate $P(x_1, \ldots, x_L = O)$ from the output of your forward and backward algorithms. Please finish the implementation of the function `seqprob_forward()` and `seqprob_backward()` in `hmm.py`. Both of them should return $P(x_1, \ldots, x_L = O)$.

1.3 We want to compute the most likely state path that corresponds to the observation O. Namely,

$$\boldsymbol{k}^* = (k_1^*, k_2^*, \cdots, k_L^*) = \arg\max_{\boldsymbol{k}} P(s_{k_1}, s_{k_2}, \cdots, s_{k_L} | x_1, x_2, \cdots, x_L = O).$$

Please implement the Viterbi algorithm in `viterbi()` in `hmm.py`. The function `viterbi(`π, A, B, O`)` takes the model parameters π, A, B and the observation sequence O as input and output a list `path` which contains the most likely state path $\boldsymbol{k}^*$ (in terms of the state index) you have found.

What to do and submit: After you finish each task in **1.1**, **1.2** and **1.3** in `hmm.py`, run the script `hmm.sh`. It will generate `hmm.txt`. Add, commit, and push both `hmm.py` and `hmm.txt` before the due date.

Problem 2 Principal Component Analysis

In this question we will implement Principal Component Analysis (PCA) and use it for image compression. We will use black and white images in this example for illustratory purposes, however, PCA can also be used for compression of color images too. For that you might need to perform compression on each channel separately and then combine them.

Let the data matrix X be $N \times P$, where N is the number of entries and P is the dimensionality of each sample in the dataset. As you remember from the lecture, the purpose of PCA will be mapping P-dimensional data into K dimensions, where $K < P$. This is achieved by finding a linear transformation of the variables to capture a maximum amount of spread in the points. You have seen in the lecture that this

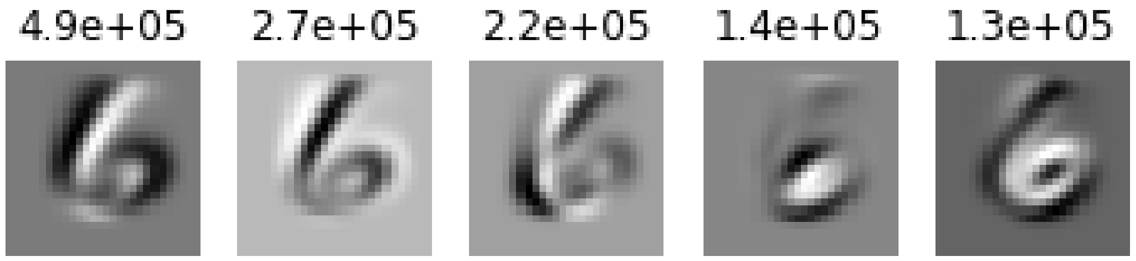

Figure 1: Eigenvectors together with their eigenvalues

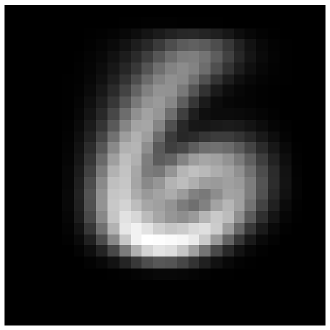

Figure 2: Mean image

transformation is given by eigenvectors and eigenvalues of the covariance matrix $X^T X$ (after subtracting the mean vector $\frac{1}{N} \times \mathbf{1}^T X$ from each row of X). The size of this covariance matrix is $P \times P$.

Because each of those eigenvectors has dimensionality P, it can be represented as an image of the same size as data points. For example, in Figure 1 we demonstrate top eigenvectors for every digit 6 in MNIST together with their eigenvalues. Figure 2 demonstrates the mean values of MNIST images that correspond to digit 6.

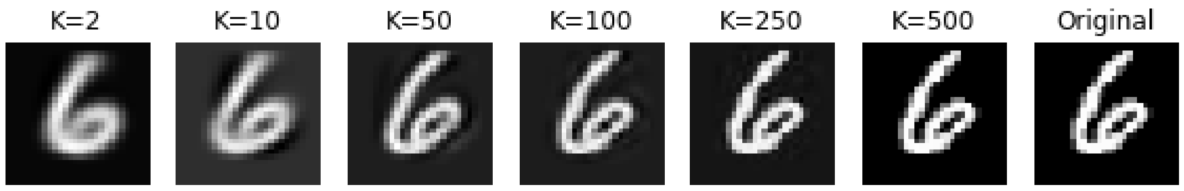

Figure 3: Original image (on the right) and decompressed versions with different compression rates.

Compression

PCA approximation to the data vector is the compressed version of that data vector, because instead of storing data matrix X of size $N \times P$, we store matrix Y of size $N \times K$, where $K < P$. The smaller the value of K, the greater the compression rate. **Please make sure that each eigenvector in M is ℓ_2 normalized.**

What to do: Please finish the implementation of *pca* function in `pca.py` file, which will return compressed representation Y and a matrix M that consists of the top K eigenvectors, represented as column vectors. Note, that we have subtracted the mean values from the data before passing it to *pca*, so you don't need to do it.
What to submit: Nothing

Decompression

We can restore the images by multiplying the compressed version of the data Y by the transpose of the eigenvectors matrix M (i.e., YM^T). This will lead to the decompressed version of our data $\hat{X}$. In Figure 3 we show different compression rates with the corresponding number of principal components used for each compression. It is easy to see that the more principal components we use, the better result visually is. To quantify the reconstruction error you will calculate pixel-wise MSE between decompressed image and the original image.
What to do and submit: Finish implementation of *decompress* and *reconstruction_error* function in `pca.py` and run it to generate `pca_output.txt`. Submit `pca.py` and `pca_output.txt`.