

Problem Set 2: PCA, Kernels, and Graphical Models

Due: 5PM Monday, Nov. 10, 2008

First, [download the datasets](#). All of the implementations should be done in FASTlib and submitted. Please be clear about which implementations correspond to which parts of the problem set.

For your submission, create a directory with the same name as your GT user account. Place into this directory all of the files necessary to build your code using FASTlib (including build.py), along with a PDF document with answers to the questions and the requested figures. Do not send any of the datasets for this assignment. Since we sent them to you, we already have them! Compress the directory, name the archive with the same name as your GT username, and email it to the TA at niche@cc.gatech.edu. The subject line should be "CSE6740 - PS2" without the quotes.

1 PCA

In `datasets/faces/image_data.txt`, you will find image data of 20 people's faces at different pose angles. In all, there are $N = 575$ images. The data have been stored such that each of $D = 2576$ pixels is a dimension and the pixels are stored column after column in one vector. For consistency with the dimensionality reduction lecture, when you load the data it will be in a $N \times D$ matrix, for N points and D dimensions; i.e. the file is $D \times N$ but since `data::Load()` transposes the data, your matrix will be $N \times D$ upon loading.

For the following, you should center (remove the mean from) the data, and add the mean back in after you obtain some rank- k approximation of the data via PCA.

a) Implement the power method to obtain the first 50 principal components of the image data. For each principal component, use 20 power iterations. The algorithm for the power method was covered in the lecture on dimensionality reduction methods. In order to use this method, you first need to form the square matrix $A = X^T X$, where $X \in \mathbf{R}^{N \times D}$, N is the number of points, and D is the number of dimensions.

Finding the first principal component is straightforward. Let $A^{(1)} = A$. To find the k^{th} principal component for $k > 1$, you apply the power method to the matrix $A^{(k)}$, which is the result of removing the $(k-1)^{\text{th}}$ principal component from $A^{(k-1)}$. This can be done via $A^{(k)} = A^{(k-1)} - A^{(k-1)} x x^T$, where x is the $(k-1)^{\text{th}}$ principal component. You should do $(Ax)x^T$ rather than $A(xx^T)$. The difference in computational complexity is a factor of D , and D is large.

b) Implement PCA using SVD. Find the first 50 principal components of the image data. These should closely match the principal components found via the power method, though they may differ in sign and there may be a some discrepancy that decreases as the number of power iterations used increases.

c) Using the first 50 principal components of the image data, find the best rank-50 approximation of the images. Do this for both the principal components obtained via the power method and the principal components obtained via SVD. Using the below MATLAB code, display the original versions of the first 2 images, their best rank-50 approximations using the power method, and their best rank-50 approximations images using PCA via SVD. Save these plots and include them in your submission.

MATLAB code:

```
% some_image is some image vector.
% The image vector is in the format in which you were given the data.
colormap(gray(256));
image(reshape(some_image, 56, 46));
```

2 Kernels

a) Prove by counterexample that the Epanechnikov kernel is not a Mercer kernel. This is true generally, but for the proof you may use $h = 1$. One property that you may find useful for the proof is that all kernel matrices induced by a Mercer kernel are positive semi-definite ($K \succeq 0$). If you are able to construct a more general proof (i.e., for all bandwidths), we will offer extra credit.

b) *Extra Credit* - Devise a good kernel for the face data, and provide an argument for why your kernel may capture interesting features for this data.

3 Graphical Models

Stock market data is provided in `datasets/stocks`. The data are closing prices from NTDOY, Nintendo's stock (actually an ADR) listed on the NYSE, from November 26th, 1996 until September 4th 2008. The data in `ntdoy_training.txt` and `ntdoy_test.txt` is the raw data. The data in `training_data.txt` and `test_data.txt` have been preprocessed to make them usable by a hidden Markov model. The adjustment used involves taking the 5 day moving average of

$$x_t = \alpha \log \frac{p_t}{p_{t-1}},$$

where p_t is the closing price on day t and α is a normalization constant to make the distribution of the points x_t unit-variance.

This problem involves using Viterbi training of a hierarchical hidden Markov model to label a stock at any given time as Bull or Bear. Below, we introduce Viterbi training of HMMs and a hierarchical HMM model.

A hidden Markov model is parameterized by $\lambda = [\pi, A, B]$, where π is the vector of initial state probabilities, $[A]_{i,j}$ is the probability of transitioning from state s_i to state s_j , and B is the set of state emission probability distribution parameters (the mean and variance of each state if the states emit univariate Gaussian random variables).

Training hidden Markov models (HMMs) using the Baum-Welch algorithm can be computationally intensive. An alternative, inconsistent but empirically decent training method for HMMs is called Viterbi training.

This involves 3 steps:

- 1 Initialize the model λ .
- 2 Find the most likely sequence of hidden states, known as the Viterbi path.
- 3 Update the model λ to maximize $P(X, Y|\lambda)$.

The last 2 steps are repeated until either the Viterbi path does not change between iterations or you have run the algorithm for many repetitions and the change is very small. The Viterbi path can be found using the Viterbi algorithm. This algorithm takes as input a set of observations y_1, \dots, y_T and returns the Viterbi path. The Viterbi path is

$$\arg \max_X P(X, Y|\lambda),$$

and it is the maximum likelihood sequence of hidden states that explain our observations Y .

The Viterbi algorithm operates by finding at each time t the maximum likelihood sequence of states terminating at state s_j , for all states $s_j \in S$.

Let the Viterbi path $V_t(s_j)$ be the maximum likelihood sequence of states terminating at state s_j and including all observations up to y_{t-1} . Then the Viterbi path $V_1(s_j)$ is simply s_j and occurs with probability $P(s_j|\lambda)$, because there are no observations prior to y_1 :

$$\begin{aligned} V_1(s_j) &= s_j \\ \Pr(V_1(s_j)) &= \Pr(s_j) \end{aligned}$$

The Viterbi algorithm exploits the Markov property of a hidden Markov model to consider only the Viterbi paths $V_t(s_i)$, for all states s_i , when computing the Viterbi path $V_{t+1}(s_j)$. It turns out that the length $t+1$ maximum likelihood sequence ending at state s_j is equivalent to the most likely sequence, for all s_i , of a length t maximum likelihood sequence ending at state s_i that is followed by a transition to state s_j . This result allows the following recurrence:

$$V_t(s_j) = [V_{t-1}(s_i) s_j],$$

$$\text{for } s_i = \arg \max_{s_i} \Pr(V_{t-1}(s_i)) \Pr(x_{t-1} = s_i, y_{t-1}) \Pr(x_t = s_j | x_{t-1} = s_i)$$

$$\Pr(V_t(s_j)) = (\max_{s_i} \Pr(V_{t-1}(s_i)) \Pr(x_{t-1} = s_i, y_{t-1}) \Pr(x_t = s_j | x_{t-1} = s_i))$$

Given the optimal Viterbi path

$$V_{T+1} = \arg \max V_{T+1}(s_j) \Pr(V_{T+1}(s_j)),$$

we can update the parameters λ of the HMM similar to the Baum-Welch algorithm:

Update each state by considering only the points which were generated by that state in the Viterbi path:

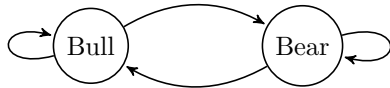
$$\mu_i = \text{mean of points generated from state } s_i$$

$$\Sigma_i = \text{covariance of points generated from state } s_i$$

Update the state transition probabilities matrix:

$$a_{i,j} = \frac{\text{number of transitions from state } s_i \text{ to state } s_j}{\text{number of transitions from state } s_i}$$

For your problem, we will consider a modified hierarchical HMM consisting of two HMMs related to stock market data: a Bull HMM and a Bear HMM. Each HMM will consist of 5 states, and, for simplicity, each state will be a single univariate Gaussian. Below is the graphical model structure:



Bull and Bear are HMMs, HMM_{Bull} and HMM_{Bear} respectively, each consisting of 5 states. Every state in Bull can transition to every state in Bear, and every state in Bear can transition to every state in Bull.

The difference between this model and a fully connected HMM of 10 states is that the observations are not just univariate real numbers, but also include labels *Bull* (1) and *Bear* (0) (provided by market analysts). Any state in HMM_{Bull} can generate only Bull labels, and likewise any state in HMM_{Bear} can generate only Bear labels. Hence,

$$\Pr(y_t, z_t = Bull | x_t = s_i^{Bull}) = \Pr(y_t | x_t = s_i^{Bull})$$

$$\Pr(y_t, z_t = Bear | x_t = s_i^{Bull}) = 0$$

$$\Pr(y_t, z_t = Bear | x_t = s_i^{Bear}) = \Pr(y_t | x_t = s_i^{Bear})$$

$$\Pr(y_t, z_t = Bull | x_t = s_i^{Bear}) = 0$$

where s_i^{Bull} and s_i^{Bear} are the i^{th} states of HMM_{Bull} and HMM_{Bear} respectively.

The state distributions can be initialized by using k -means clustering: cluster all training data labeled as Bull to initialize HMM_{Bull} 's state distributions, and likewise for the training data labeled as Bear to initialize HMM_{Bear} 's state distributions.

- a) Implement a modified Viterbi training algorithm that takes into account the Bull/Bear labels as described above.
- b) Use your modified Viterbi training algorithm to estimate the model parameters from the training data and labels at `training_data.txt` and `training_labels.txt`. Note that 1 corresponds to the Bull label and 0 corresponds to the Bear label.
- c) For the test data at `test_data.txt`, find the Viterbi path (do not use the true Bull/Bear labels) and plot the raw data, in `ntdoy_test.txt`, such that all observations labeled by a Bull state in the Viterbi path are in red and all observations labeled by a Bear state in the Viterbi path are in blue. This is easy to do in MATLAB using multiple scatterplots in one figure.
- d) Using the true Bull/Bear labeling for the test data at `test_labels.txt`, calculate the error of your prediction using zero-one loss:

$$\frac{1}{T} \sum_{t=1}^T |\hat{z}_t - z_t|$$

where \hat{z}_t is your Bull/Bear prediction at time t and z_t is the ground truth at that time.

- e) **Extra Credit** - Design a better graphical model for the stock market data, and explain how it captures the underlying relationships necessary for solving the problem of predicting a Bull versus a Bear market.