Lecture 17

Unsupervised Learning: Dimensionality Reduction
CS542—Contents 17

- Data Preprocessing and Dimensionality Reduction
  - Data Preprocessing
    - Scaling and Centering of Data
    - Missing Data
    - Time Series Prediction
  - Dimensionality Reduction
    - PCA and Auto-encoders
    - Factor Analysis
    - Partial Least Squares
    - Spatially Localized Versions

- Handout:
  - Class Notes

- Reading Assignment for Next Class
  - Bishop Ch.12
Remarks About Data Preprocessing

- **Possible Ways of Preprocessing**
  - linear transformations on the data
  - feature extraction (nonlinear transformation)
  - dimensionality reduction

- **Advantages of Preprocessing**
  - better interpretability of the learning parameters
  - easier to initialize learning systems
  - significant reduction of the open parameters $=>$ better generalization
  - faster training
  - avoid curse of dimensionality
  - outlier and missing data removal

- **Disadvantages of Preprocessing**
  - data preprocessing can only reduce the information in the data, never increase it!
  - not REALLY necessary since theoretically a good learning system could do everything by itself
  - preprocessing is not independent of subsequent processing
Scaling and Centering

- **Goal:**
  - mean zero data
  - unit variance data
  - whitening (a.k.a. sphering)

- **Why:**
  - in spatially localized systems: make sure that distance calculations are not biased by different spread (units) of different input dimensions
  - in projection networks: make the magnitude of the network weights interpretable, and allow easier initialization
  - some algorithms (e.g., PCA) require mean-zero data to work properly

- **Danger:**
  - magnify noise (thus, scaling based on physical insights is often better than unit variance scaling!)
Scaling and Centering (cont’d)

- **Batch Version**
  \[ x_i = \frac{1}{N} \sum_{n=1}^{N} x_i^n \]
  \[ \sigma_i^2 = \frac{1}{N-1} \sum_{n=1}^{N} (x_i^n - \bar{x}_i)^2 \]
  \[ \bar{x}_i^n = \frac{x_i^n - \bar{x}_i}{\sigma_i} \]

- **Incremental (Stochastic) Version**

  With no forgetting:
  \[ x_i^{k+1} = \bar{x}_i^k + \frac{1}{k+1} (x_i^n - \bar{x}_i^k) \]
  \[ \sigma_i^{2k+1} = \sigma_i^{2k} + \frac{1}{k+1} \left( (x_i^n - \bar{x}_i^k)^2 - \sigma_i^{2k} \right) \]

  With forgetting:
  \[ x_i^{k+1} = x_i^k + \alpha (x_i^n - \bar{x}_i^k) \text{ where } \alpha = [0,1] \]
  \[ \sigma_i^{2k+1} = \sigma_i^{2k} + \alpha \left( (x_i^n - \bar{x}_i^k)^2 - \sigma_i^{2k} \right) \]
Missing Data

- Some of the input variables are missing
  - e.g., because of sensor failures, incompletely filled out questionnaires, inability of a subject to perform a subtask, etc.

- It is mostly bad to fill in missing data just from heuristics, i.e., fill in the mean:

- Probabilistic Approaches (usually EM-based) allow to solve the missing data problem properly
Time Series Prediction

- The inputs to the network are a set of time-delayed values.
- The goal is to predict k steps into the future.
  - NOTE: this is another way to build internal models of systems.
  - Taken theorem (1981):
    - “There exists a smooth function of t most 2d+1 past measurement that correctly predicts the future value of the time series” (d is the dimensionality of the true data generating process).
Dimensionality Reduction

- Goals:
  - fewer dimensions for subsequent processing
  - better numerical stability due to removal of correlations
  - simplify post processing due to “advanced statistical properties” of pre-processed data
  - don’t lose important information, only redundant information or irrelevant information
  - perform dimensionality reduction spatially localized for nonlinear problems (e.g., each kernel has its own local dimensionality reduction)
Hebbian Learning

- Hebbian learning is among the most classical methods of dimensionality reduction

\[ w^{n+1} = w^n + \alpha xy \]
Hebbian Learning

- Properties of Hebbian Learning
  - Unstable learning rule
  - Finds direction of maximal variance

\[ J = \frac{1}{2} y^2 = \frac{1}{2} w^T xx^T w \]

\[ E[J] = E\left[\frac{1}{2} w^T xx^T w\right] = \frac{1}{2} w^T E[xx^T] w = \frac{1}{2} w^T C w \]

\( C \) is the correlation matrix

\[ \Delta w = \alpha xy \]
Oja’s Rule

- Fix the instability of Hebbian Learning by requiring unit length projection \( \mathbf{w}^T \mathbf{w} = 1 \)

- Oja’s update rule

\[
\Delta \mathbf{w} = \mathbf{w}^{n+1} - \mathbf{w}^n = \alpha y (\mathbf{x} - y \mathbf{w}) = \alpha (y \mathbf{x} - y^2 \mathbf{w})
\]

Verification:

\[
E\{\Delta \mathbf{w}\} = E\{\alpha (\mathbf{x}y - y^2 \mathbf{w})\}
\]

\[
= \alpha E\{\mathbf{x} \mathbf{x}^T \mathbf{w} - \mathbf{w}^T \mathbf{x} \mathbf{x}^T \mathbf{w} \mathbf{w}\}
\]

\[
= \alpha (C \mathbf{w} - \mathbf{w}^T C \mathbf{w} \mathbf{w})
\]

After convergence:

\[
E\{\Delta \mathbf{w}\} = 0 \Rightarrow
\]

\[
C \mathbf{w} = (\mathbf{w}^T C \mathbf{w}) \mathbf{w} = \lambda \mathbf{w}, \text{ thus } \mathbf{w} \text{ is eigenvector}
\]

\[
\mathbf{w}^T C \mathbf{w} = \mathbf{w}^T (\mathbf{w}^T C \mathbf{w}) \mathbf{w} = (\mathbf{w}^T C \mathbf{w}) \mathbf{w}^T \mathbf{w}
\]

thus, \( \mathbf{w}^T \mathbf{w} = 1 \)
Example of Oja’s Rule

Usually, the data is transformed to be zero mean!
Autoencoder as Motivation for Oja’s Rule

- Oja’s Rule looks like supervised learning with a “bottleneck” network

\[
J = \frac{1}{2} (x - \hat{x})^T (x - \hat{x}) = \frac{1}{2} (x - yw)^T (x - yw)
\]

\[
\frac{\partial J}{\partial w} = -\left( \frac{\partial y}{\partial w} w + y \frac{\partial w}{\partial w} \right)^T (x - yw)
\]

\[
= -\left( \frac{\partial (w^T x)}{\partial w} w + y \frac{\partial w}{\partial w} \right)^T (x - yw)
\]

\[
= -(x^T w + y)(x - yw)
\]

\[
= -(y + y)(x - yw)
\]

\[
= -2y(x - yw)
\]

\[
\Delta w = -\alpha \frac{\partial J}{\partial w} = \alpha' y(x - yw)
\]

This is Oja’s rule!
Extension to Multiple Dimensions

- **Oja’s Rule**

  \[ y = Wx \]
  \[ \hat{x} = W^T y \]
  \[ \Delta W = \alpha y (x - W^T y)^T \]

- **Sanger’s Rule**

  \[
  [\Delta W]_{r-th-row} = \alpha y_r \left( x - \sum_{i=1}^{r} ([W]_{i-th-row})^T y_i \right)^T \\
  = \alpha y_r \left( x - ([W]_{1:r-th-row})^T [y]_{1:r-th-row} \right)^T
  \]

  Matlab Notation:

  \[ W(r,:) = \alpha \ast y(r) \ast (x - W(1:r,:)') \ast y(1:r)) \]

  This rule makes the rows of \( W \) become the eigenvectors of the data, ordered in descending sequence according to the magnitude of the eigenvalues.
Summary of Principle Components

- PCA finds the eigenvectors of the correlation (covariance) matrix of the data.
- PCA can be learned by bottle-neck neural networks (auto-associator).
PCA for Supervised Learning

- In Linear Algebra Notation:

\[ U = \text{eigenvectors} \left( \sum_{n=1}^{N} (x^n - \bar{x})(x^n - \bar{x})^T \right) \]

\[ = \text{eigenvectors} \left( \frac{\tilde{X}^T \tilde{X}}{N-1} \right)_{\max(1:k)} \]

where \( \tilde{X} \) contains mean-zero data

Subsequent Linear Regression for Network Weights:

\[ W = (U^T \tilde{X}^T \tilde{X}U)^{-1} U^T \tilde{X}^T Y \]

NOTE: Inversion of the above matrix is very cheap since it is diagonal! No numerical problems!

Problems of this pre-processing:
Important regression data might have been clipped!
PCA in Joint Data Space

- A straightforward extension to take supervised learning step into account:
  - perform PCA in joint data space
  - extract linear network parameters from PCA results
PCA in Joint Data Space: Formalization

\[
Z = \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix}
\]

\[
U = \left[ \text{eigenvectors} \left( \frac{\sum_{n=1}^{N} (z^n - \bar{z})(z^n - \bar{z})^T}{N-1} \right) \right]_{\max(1:k)}
\]

\[
W = U_x \left( U_y^T - U_y^T \left( U_y U_y^T - I \right)^{-1} U_y U_y^T \right), \text{ where } U = \begin{bmatrix} U_x (= d \times k) \\ U_y (= c \times k) \end{bmatrix}
\]

Note: this new kind of linear network can actually tolerate noise in the input data! But only the same noise level in all (joint) dimensions!
Why PCA can only tolerate spherical noise:

- **No noise**
  - Input: \(X_{i,n} \sim N(0,1)\)
  - Output: \(Y_{i,n} = X_{i,n} + \epsilon_{i,n}\)

- **Noise in inputs**
  - Input: \(X_{i,n} \sim N(0,1)\)
  - Output: \(Y_{i,n} = X_{i,n} + \epsilon_{i,n}\)

- **Noise in outputs**
  - Input: \(X_{i,n} \sim N(0,1)\)
  - Output: \(Y_{i,n} = X_{i,n} + \epsilon_{i,n}\)

- **Noise in inputs & outputs**
  - Input: \(X_{i,n} \sim N(0,1)\)
  - Output: \(Y_{i,n} = X_{i,n} + \epsilon_{i,n}\)
The Probabilistic Way: Factor Analysis

- Data Generating Model:

\[ z = Uv + \varepsilon \quad \text{where} \quad \varepsilon = N(0, \Omega) \quad \text{and} \quad v = N(0, 1) \]

- The parameters U and \( \Omega \) can be estimated by max. likelihood, in particular the EM algorithm
Factor Analysis

- Probability Model:
  \[ v_m \sim N(0,1) \]
  \[ \epsilon \sim N(0,\Omega) \text{ where } \Omega \text{ is diagonal} \]

- Complete log likelihood

\[
\begin{align*}
\log p(Z,V) &= \log \prod p(z_i,v_i) \\
&= \log \prod p(z_i | v_i) p(v_i) \\
&= \sum \log p(z_i | v_i) + \sum \log p(v_i) \\
&= \sum \left[ \log \left( \frac{1}{\sqrt{(2\pi)^d | \Omega|}} \right) - \frac{1}{2} (z_i - Uv_i)^T \Omega^{-1} (z_i - Uv_i) \right] + \sum \left[ \log \left( \frac{1}{\sqrt{(2\pi)^m | I|}} \right) - \frac{1}{2} v_i^T v_i \right]
\end{align*}
\]
Factor Analysis

- Sketch of derivation of the EM algorithm:
  - Derive the posterior of the hidden variables (E-step):
    - $p(v|z) = \frac{p(z|v)p(v)}{p(z)}$
  - Derive expectations $E\{v\}$ and $E\{vv^T\}$
  - Derive maximization w.r.t. $U$ and $\Omega$
The EM-Algorithm for Factor Analysis

**E - Step:**

\[ \beta = U^T(\Omega + UU^T)^{-1} \]

\[ E\{v \mid z\} = \beta x \]

\[ E\{vv^T \mid z\} = I - \beta U + \beta zz^T \beta^T \]

**M - Step:**

\[ U^{new} = \left( \sum_{n=1}^{N} z^n E(v \mid z^n)^T \right) \left( \sum_{n=1}^{N} E\{vv^T \mid z^n\}^T \right)^{-1} \]

\[ \Omega^{new} = \frac{1}{N} \text{diag} \left\{ \sum_{n=1}^{N} z^n z^n^T - U^{new} E(v \mid z^n)z^n^T \right\} \]
Factor Analysis for Supervised Learning

- A straightforward extension allows factor analysis to be used for supervised learning:

\[
\mathbf{z} = \begin{bmatrix} \mathbf{x} - \mathbf{x} \\ \mathbf{y} - \mathbf{y} \end{bmatrix}
\]

\[
\mathbf{v} = \tilde{\mathbf{x}}, \text{ i.e., the TRUE (non-noise contaminated inputs)}
\]

\[
\mathbf{U} = [\mathbf{I}, \mathbf{W}]^T
\]

- After performing EM on joint data, the network weights are:

\[
E\left[ \begin{bmatrix} \mathbf{y} \\ \mathbf{v} \end{bmatrix} | \mathbf{x} \right] = \begin{bmatrix} \mathbf{W}^T \\ \mathbf{B} \end{bmatrix} \mathbf{x} = \Psi_{21} \Psi_{11}^{-1} \mathbf{x}, \text{ where}
\]

\[
\Psi = \begin{bmatrix} \Omega + \mathbf{UU}^T & \mathbf{U} \\ \mathbf{U}^T & \mathbf{I} \end{bmatrix}
= \begin{bmatrix} \Psi_{11} (= \mathbf{d} \times \mathbf{d}) & \Psi_{12} (= \mathbf{d} \times (\mathbf{c} + \mathbf{k})) \\ \Psi_{21} (= (\mathbf{c} + \mathbf{k}) \times \mathbf{d}) & \Psi_{22} (= (\mathbf{c} + \mathbf{k}) \times (\mathbf{c} + \mathbf{k})) \end{bmatrix}
\]

\[
d := \text{dimensionality of observed data}
\]

\[
c := \text{dimensionality of (supervised) outputs}
\]

\[
k := \text{reduced dimensionality}
\]
Factor Analysis: Example

noise in inputs and outputs is different!

Note that the weights $W$ will not be identical to the true weights of the data generating model, since they are optimized to average out the noise in the data.
Partial Least Squares Regression

- A linear regression method with dimensionality reduction

\[ u = X^T y \]  Determine projection from correlation

\[ s = Xu \]  Project the input data

\[ \beta = \frac{s^T y}{s^T s} \]  Perform single variate regression

Iterate multiple times on residual error

Note: PLS does NOT need to span the entire input space -- indeed for spherical input distributions, a SINGLE projection will be optimal!

\[ \beta = (X^T X)^{-1} X^T y = (X^T X)^{-1} u \]

if \( X^T X = \sigma^2 I \) then \( \beta = \frac{1}{\sigma^2} u \)
Partial Least Squares Regression

- Partial Least Squares is a linear regression method that includes dimensionality reduction

For Training:
Initialize (assume mean zero \( X \) and \( y \)):
\[
D_0 = X, \quad e_0 = y
\]
For \( i = 1 \) to \( k \):
\[
\begin{align*}
    u_i &= D_{i-1}^T e_{i-1} \\
    s_i &= D_{i-1} u_i \\
    w_i &= \frac{s_i^T e_{i-1}}{s_i^T s_i} \\
    p_i &= \frac{D_{i-1}^T s_i}{s_i^T s_i} \\
    D_i &= D_{i-1} - s_i p_i^T
\end{align*}
\]

For Lookup:
Initialize:
\[
\begin{align*}
    d_0 &= x, \quad y = 0
\end{align*}
\]
For \( i = 1 \) to \( k \):
\[
\begin{align*}
    s_i &= d_{i-1}^T u_i \\
    y &= y + w_i s_i \\
    d_i &= d_{i-1} - s_i p_i \\
    W &= [w_1 \ w_2 \ \ldots \ w_c]
\end{align*}
\]
Comparing Dimensionality Reduction Methods

![Comparison of Dimensionality Reduction Methods](image.png)

- **a) Regression Results with 4 Factors**
  - Equal Noise in all Inputs and Outputs
  - Unequal Noise in all Inputs and Outputs

- **b) Regression Results with 5 Factors**
  - LWFA
  - LWPCA
  - LWPCR
  - LWPLS
  - LWPLS_1

- **c) Regression Results with 6 Factors**

- **d) Summary Results**
  - Average nMSE on Test Set

![Summary Results](image.png)
Variational Bayesian Least Squares

- Start with linear regression …

\[ y_i = x_i^T \beta + \epsilon \]
\[ \epsilon \sim N(0, \psi_y) \]
\[ \beta = (X^T X)^{-1} X y \]
Variational Bayesian Least Squares

- Add PLS projection as hidden variables …

\[ z_{i,m} = x_{i,m} \beta_j + \eta_m \]

\[ y_i = \sum_{m=1}^{d} z_{i,m} + \varepsilon \]

\[ \varepsilon \sim N(0, \psi_y) \]

\[ \eta_m \sim N(0, \psi_{z,m}) \]
PPLS Algorithm

- **E-Step**

  \[ 1^T \langle z_i \rangle = \frac{y_i}{s} s + \left( 1 - \frac{s}{s} \right) b^T x_i \]
  \[ \langle z_{i,m}^2 \rangle = \psi_{z,m} - \frac{\psi_{z,m}^2}{s} + \langle z_{i,m} \rangle^2 \]

  \[ 1^T \langle z_i z_i^T \rangle 1 = S - \frac{S^2}{s} + \left( 1^T \langle z_i \rangle \right)^2 \]

  \[ \langle z_{i,m} \rangle = b_m x_{i,m} + \frac{\psi_{z,m}}{s} \left( y_i - b^T x_i \right) \]

  Resembles “weighted” backfitting

- **M-Step**

  \[ b_m = \frac{\sum_{i=1}^{N} w_i \langle z_{i,m} \rangle x_{i,m}}{\sum_{i=1}^{N} w_i x_{i,m}^2} \]

  \[ \psi_y = \frac{1}{W} \left[ \sum_{i=1}^{N} w_i y_i^2 - 2 1^T \sum_{i=1}^{N} w_i y_i \langle z_i \rangle + 1^T \left( \sum_{i=1}^{N} w_i \langle z_i z_i^T \rangle \right) 1 \right] \]

  \[ \psi_{z,m} = \frac{1}{W} \left[ \sum_{i=1}^{N} w_i \langle z_{i,m}^2 \rangle - 2 b_m \sum_{i=1}^{N} w_i \langle z_{i,m} \rangle x_{i,m} + b_m^2 \sum_{i=1}^{N} w_i x_{i,m}^2 \right] \]

  \[ d \text{ univariate regressions!} \]
Variational Bayesian Least Squares

- Add automatic relevance detection …

\[ z_{i,m} = x_{i,m} \beta_j + \eta_m \]
\[ y_i = \sum_{m=1}^{d} z_{i,m} + \epsilon \]
\[ \epsilon \sim N(0, \psi_y) \]
\[ \eta_m \sim N(0, \psi_{z,m}) \]
\[ \beta_m \sim N\left(0, \frac{1}{\alpha_m}\right) \]
\[ \alpha_m \sim \text{Gamma}(a_\alpha, b_\alpha) \]
PCA Summary and Notes

- PCA finds the directions of maximal variance of data
- Hebbian learning, Oja’s rule, Sanger’s rule are classical neural network methods for PCA
- Factor analysis is the probabilistic method of PCA, which can be proven to include PCA as a special case
- Partial Least Squares is a useful regression algorithm with dimensionality reduction
- Variational Bayesian Least Squares is a full Bayesian treatment of regression and dimensionality reduction
- Dimensionality reduction for supervised learning needs to be done carefully, taking the output data into account