Machine Learning

Feature Learning
Feature Learning

- Feature representations are important for supervised learning problems as well as Reinforcement Learning problems
  - Feature representation can change the complexity of the decision boundary
  - Feature representation can change the dimensionality of the problem
- Features can be either designed or learned
  - Supervised: e.g. ANN hidden units learn features
  - Unsupervised: e.g. cluster IDs as features
Unsupervised Feature Learning

- Unsupervised features have to be built around general principles for “good” features
  - Metric for “good” features has to be built into the algorithm

- Clustering can be used to form features IDs can be used as discrete features (similar items have the same/similar features)
  - Cluster ID as discrete feature
  - Cluster probability as continuous feature

- Other criteria for “good” features can be used
Unsupervised Feature Learning

- A common criterion for “good” features are
  - How precisely they can represent the data
  - How compact the basis is
- Representation accuracy is aimed at ensuring that the unsupervised feature learning does not lose significant amounts of information
  - Information loss might make some subsequent tasks impossible to do/learn
- Compactness is aimed at simplicity
  - Reduce overfitting
  - Reduce complexity for subsequent tasks
Principal Component Analysis

- Principal component analysis (PCA) is one of the most used approaches for unsupervised learning of a compact feature space
  - Uses both accuracy of representation of data and compactness of the representation
  - Assumes linear representation of data in terms of learned, constant, unit length basis vectors
    \[ \hat{x}^{(i)} = \bar{x} + \sum_k \phi_k (x^{(i)}) \hat{u}_k \]
    - Accuracy of representation is defined in terms of a common criterion for “good” features are
      \[ E_{k,\phi,\hat{u},D} = \sum_i \left\| x^{(i)} - \hat{x}^{(i)} \right\|^2 = \sum_i \left\| x^{(i)} - \left( \bar{x} + \sum_{j=1}^k \phi_j (x^{(i)}) \hat{u}_j \right) \right\|^2 \]
Principal Component Analysis

- PCA basically forms a new basis for the data in such a way that for every number of features, $k$, the resulting basis minimizes the square reconstruction error.

- Each feature tries to capture as much of the remaining data variation as possible, reducing the squared error as much as possible.
  
  - Feature value that minimizes the error for all $k$:
    \[
    \phi_k(x^{(i)}) = \hat{u}_k^T \left(x^{(i)} - \bar{x}\right)
    \]

  - Corresponding basis vectors have to minimize the error:
    \[
    \hat{u}^* = \arg\min_u \sum_i \left\| x^{(i)} - \left(\bar{x} + \sum_{j=1}^k \phi_j(x^{(i)})\hat{u}_j\right)\right\|^2
    \]
Principal Component Analysis

- To solve for the basis it is important to note:
  - The vectors have to be orthogonal
  - The error for $k=d$ is 0
- Thus the error for a $k$ is equal to the value of the higher components

$$E_{k, \phi, \hat{u}, D} = \sum_i \left\| \sum_{j=k+1}^{d} \hat{u}_j^T (x^{(i)} - \bar{x}) \hat{u}_j \right\|^2$$

$$= \sum_i \sum_{j=k+1}^{d} \left( \hat{u}_j^T (x^{(i)} - \bar{x}) \right)^2$$

$$= \sum_{j=k+1}^{d} \sum_i \left( \hat{u}_j^T (x^{(i)} - \bar{x}) \right)^2 = \sum_{j=k+1}^{d} \sum_i \left( (x^{(i)} - \bar{x})^T \hat{u}_j \right)^2$$

$$= \sum_{j=k+1}^{d} \sum_i \hat{u}_j^T (x^{(i)} - \bar{x})(x^{(i)} - \bar{x})^T \hat{u}_j = \sum_{j=k+1}^{d} \hat{u}_j^T \Sigma \hat{u}_j$$
Principal Component Analysis

- Solving from top down, starting with $k=d$ we can notice that $\hat{u}_d = \arg \min_{\hat{u}} \hat{u}_d^T \Sigma \hat{u}_d$ has its solution for the smallest eigenvector of $\Sigma$
  - In the same way, each earlier basis vector corresponds to the next smaller eigenvector of $\Sigma$
- Principal components of a data set are the eigenvectors of normalized data’s covariance matrix in order of increasing eigenvalue
  - If scales of original dimensions are incompatible, data can be normalized with standard deviation
Principal Component Analysis

- PCA example

Data actually likes along some diagonal axis (the \( u_1 \) direction) capturing the intrinsic piloting "karma" of a person, with only a small amount of noise lying off this axis. (See figure.) How can we automatically compute this \( u_1 \) direction?

We will shortly develop the PCA algorithm. But prior to running PCA per se, typically we first pre-process the data to normalize its mean and variance, as follows:

1. Let \( \mu = \frac{1}{m} \sum_{i=1}^{m} x(i) \).
2. Replace each \( x(i) \) with \( x(i) - \mu \).
3. Let \( \sigma^2_j = \frac{1}{m} \sum_{i=1}^{m} (x(i)^j)^2 \).
4. Replace each \( x(i)^j \) with \( x(i)^j / \sigma_j \).

Steps (1-2) zero out the mean of the data, and may be omitted for data known to have zero mean (for instance, time series corresponding to speech or other acoustic signals). Steps (3-4) rescale each coordinate to have unit variance, which ensures that different attributes are all treated on the same "scale." For instance, if \( x_1 \) was cars' maximum speed in mph (taking values in the high tens or low hundreds) and \( x_2 \) were the number of seats (taking values around 2-4), then this renormalization rescales the different attributes to make them more comparable. Steps (3-4) may be omitted if we had apriori knowledge that the different attributes are all on the same scale. One
Principal Component Analysis

- PCA algorithm
  - Create $n \times d$ data matrix $D$
  - Normalize columns by subtracting column average
    - If desired, normalize columns with standard deviation
  - Compute scaled covariance matrix $\Sigma = D^T D$ of data
  - Find eigenvectors and eigenvalues of $\Sigma$
  - Sort by eigenvalue for Principal Components
- Eigenvalue indicates loss when not using the principal component
  - Shorter representation by ignoring higher components
Principal Component Analysis

- If $d$ is large the eigenvector calculation becomes expensive and potentially numerically unstable
  - Can solve using Singular Value Decomposition
    \[ D = U S V^T \]
    - $S$ is a diagonal matrix of the eigenvalues of $D^TD$
    - The columns of $V$ are the eigenvectors of $D^TD$
  - SVD is more stable and often more efficient
Principal Component Analysis

- PCA is one of the most commonly used feature learning approaches
  - E.g. Eigenfaces:
    - 25 most significant principal components of a set of face images:

![Eigenfaces](image-url)
Eigenfaces

- Using these 25 features we use nearest neighbor to identify the person
  - Reduction to a 25 dimensional representation
  - Recognition rate is above 80% for the test set
  - Reconstruction:
Feature Learning

- Other techniques exist to learn a different feature representation
  - Independent component analysis (ICA)
    - Similar to PCA but finds most statistically independent components
      - Minimizes mutual information between components
      - Or: maximizes non-Gaussianity
    - Used to separate multiple sources of stochastic data
  - Sparse PCA
    - PCA where components can be in at most k dimensions
  - Sparse coding
    - PCA with a regularization term over feature values
Self-Organizing Maps

- Self-organizing maps are neural networks that are trained in an unsupervised fashion
  - Hidden units are arranged in a k-dimensional lattice with a distance function
    - Weight vectors to a unit map into the lattice
    - Neighboring unit’s weight vectors are weakly linked based on the distance function
    - Training “deforms” lattice to map onto data points
      - Topological mapping
  - Units compete for Best Matching Unit (BMU)
    - Units cooperate with BMU, updating based on distance to BMU
Self-Organizing Maps

- Randomly initialize hidden unit weight vectors
- For each data point find the unit that has the weight vector most similar to the data (BMU)
  - Similarity is usually Cartesian distance
- Update the weights of all the units
  \[ w_{j,k} \leftarrow w_{j,k} + \alpha(t) \Theta_t(k, BMU)(x_{j(i)} - w_{j,k}) \]
  - \( \Theta(k,l,t) \) is the similarity between the lattice location of nodes k and l
- Repeat with next data point until iteration limit
Self-Organizing Maps

- SOM example

© Manfred Huber 2015

J. Lampinen
T. Kostiainen
Unsupervised Feature Learning

- Unsupervised feature learning finds possible feature representations based on characteristics built into the algorithm
  - PCA is the most commonly used
    - PCA can always perfectly represent the original data
  - ICA and sparse methods can find more features than in the original space
    - Features can be more expressive
    - Features can be more causal
  - SOM establishes a topological mapping onto a k-dimensional lattice
    - Can be seen as a non-linear PCA