CS 6140: Machine Learning
Spring 2016

Instructor: Lu Wang
College of Computer and Information Science
Northeastern University
Webpage: www.ccs.neu.edu/home/luwang
Email: luwang@ccs.neu.edu
Exam Guidelines
Basic concepts in machine learning

- Supervised Learning vs. unsupervised learning
- Classification vs. regression
- Parametric vs. non-parametric models
- Overfitting vs. underfitting
- Generative vs. discriminative models
- How to do model selection?
Supervised vs. Unsupervised Learning

- Supervised learning

\[ D = \{(x_i, y_i)\}_{i=1}^{N} \]

- Training set
- Training sample
- Gold-standard label
  - Classification, if categorical
  - Regression, if numerical
Supervised Learning

- **Goal:** $y = f(x)$
  - Generalizable to new input samples

- **Typical setup:**
  - Training set, test set, development set
  - Features
  - Evaluation
Supervised vs. Unsupervised Learning

• Unsupervised Learning

\[ D = \{ x_i \}_{i=1}^{N} \]

• More about “knowledge discovery”
Parametric vs. Non-parametric model

- Fixed number of parameters?
  - If yes, parametric model

- Number of parameters grow with the amount of training data?
  - If yes, non-parametric model

- Computational tractability
Generative VS. Discriminative Model

• Generative model
  – Learn $P(X, Y)$ from training sample
  – $P(X, Y) = P(Y)P(X|Y)$
  – Specifies how to generate the observed features $x$ for $y$

• Discriminative model
  – Learn $P(Y|X)$ from training sample
  – Directly models the mapping from features $x$ to $y$
Evaluation

• Accuracy
• Precision/recall/f-measure
• ROC
• How to compute/plot each of them
• Advantages and disadvantages
Confusion Matrix

\[
\begin{array}{ccc}
\text{Predicted 1} & \text{Predicted 0} \\
\hline
\text{True 1} & a & b \\
\text{True 0} & c & d \\
\hline
\end{array}
\]

Accuracy = \( \frac{a+d}{a+b+c+d} \)
<table>
<thead>
<tr>
<th>Predicted 1</th>
<th>Predicted 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>True 1</td>
<td></td>
</tr>
<tr>
<td>true positive</td>
<td>false negative</td>
</tr>
<tr>
<td>True 0</td>
<td></td>
</tr>
<tr>
<td>false positive</td>
<td>true negative</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Predicted 1</th>
<th>Predicted 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>True 1</td>
<td></td>
</tr>
<tr>
<td>TP</td>
<td>FN</td>
</tr>
<tr>
<td>True 0</td>
<td></td>
</tr>
<tr>
<td>FP</td>
<td>TN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Predicted 1</th>
<th>Predicted 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>True 1</td>
<td></td>
</tr>
<tr>
<td>hits</td>
<td>misses</td>
</tr>
<tr>
<td>True 0</td>
<td></td>
</tr>
<tr>
<td>false alarms</td>
<td>correct rejections</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Predicted 1</th>
<th>Predicted 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>True 1</td>
<td></td>
</tr>
<tr>
<td>P(pr1</td>
<td>tr1)</td>
</tr>
<tr>
<td>True 0</td>
<td></td>
</tr>
<tr>
<td>P(pr1</td>
<td>tr0)</td>
</tr>
</tbody>
</table>
ROC Plot

• Sweep threshold and plot
• Sensitivity = \( \frac{a}{a+b} \) = Recall
  - True positive rate
• \( 1 - \text{Specificity} = 1 - \frac{d}{c+d} = \frac{c}{c+d} \)
  - False positive rate

\[
\begin{array}{c|cc}
  & \text{Predicted 1} & \text{Predicted 0} \\
\hline
\text{True 1} & a & b \\
\text{True 0} & c & d \\
\end{array}
\]
A non-parametric classifier: K-nearest neighbors (KNN)

• Basic idea: memorize all the training samples
  – The more you have in training data, the more the model has to remember

• K-Nearest neighbor:
  – Testing phase: find the K nearest neighbors, and return the majority vote of their labels

\[ \hat{y}(x) = y_{n^*} \text{ where } n^* = \arg \min_{n \in D} \text{dist}(x, x_n) \]
About K

- $K=1$: just piecewise constant labeling
- $K=N$: global majority vote (class)
Problems of kNN

• Can be slow when training data is big
  – Searching for the neighbors takes time

• Needs lots of memory to store training data

• Needs to tune k and distance function

• Not a probability distribution
Probabilistic kNN

• We prefer a probabilistic output because sometimes we may get an “uncertain” result – 99 samples as “yes”, 101 samples as “no” → ?

• Probabilistic kNN:

\[ p(y|x, D) = \frac{1}{K} \sum_{j \in \text{nbr}(x,K,D)} I(y = y_j) \]
K-nearest neighbors (KNN)

• Definition
• How to choose a proper K
• The problems and potential solution
A parametric classifier: linear regression

- Assumption: the response is a linear function of the inputs

\[ y(x) = w^T x + \epsilon = \sum_{j=1}^{D} w_j x_j + \epsilon \]

Inner product between input sample X and weight vector W

Residual error: difference between prediction and true label
A parametric classifier: linear regression

\[ y(x) = w^T x + \epsilon = \sum_{j=1}^{D} w_j x_j + \epsilon \]

Inner product between input sample X and weight vector W

- Assume residual error has a normal distribution

\[ p(y|x, \theta) = \mathcal{N}(y|\mu(x), \sigma^2(x)) \]
A parametric classifier: linear regression

\[ p(y|x, \theta) = \mathcal{N}(y|\mu(x), \sigma^2(x)) \]

- We can further assume
  \[ \mu = w^T x \]
  \[ \sigma^2(x) = \sigma^2 \]

- Basic function expansion
  \[ p(y|x, \theta) = \mathcal{N}(y|w^T \phi(x), \sigma^2) \]
  \[ \phi(x) = [1, x, x^2, \ldots, x^d] \]
Learning with Maximum Likelihood Estimation (MLE)

- Maximum Likelihood Estimation (MLE)

\[ \hat{\theta} \triangleq \arg \max_{\theta} \log p(D|\theta) \]
Learning with Maximum Likelihood Estimation (MLE)

• Log-likelihood

\[ \ell(\theta) \triangleq \log p(\mathcal{D}|\theta) = \sum_{i=1}^{N} \log p(y_i|x_i, \theta) \]

• Maximize log-likelihood is equivalent to minimize negative log-likelihood (NLL)

\[ \text{NLL}(\theta) \triangleq - \sum_{i=1}^{N} \log p(y_i|x_i, \theta) \]
Learning with Maximum Likelihood Estimation (MLE)

• With our normal distribution assumption

$$\ell(\theta) = \sum_{i=1}^{N} \log \left[ \left( \frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left( -\frac{1}{2\sigma^2} (y_i - w^T x_i)^2 \right) \right]$$

$$= \frac{-1}{2\sigma^2} RSS(w) - \frac{N}{2} \log(2\pi\sigma^2)$$

$$RSS(w) \triangleq \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

Residual sum of squares (RSS) → We want to minimize it!
Derivation of MLE for Linear Regression

- Rewrite our objective function as

\[
\text{NLL}(w) = \frac{1}{2} (y - Xw)^T (y - Xw) = \frac{1}{2} w^T (X^T X) w - w^T (X^T y)
\]

- Get the derivative (or gradient)

\[
g(w) = [X^T Xw - X^T y] = \sum_{i=1}^{N} x_i (w^T x_i - y_i)
\]

- Set our derivative to 0

\[
X^T Xw = X^T y \iff \hat{w}_{OLS} = (X^T X)^{-1} X^T y
\]

Ordinary least squares solution
Linear regression

• Definition and assumption
• How to estimate the weight vector?
• What is maximum likelihood estimation?
• What is ridge regression? What’s the relation between ridge regression and a vanilla linear regression?
Logistic Regression

• A discriminative model

\[ p(y|x, w) = \text{Ber}(y|\mu(x)) \]

– y is 0 or 1
– Ber is a Bernoulli distribution
– \( \mu(x) = \mathbb{E}[y|x] = p(y = 1|x) \)

• Remember in linear regression

\[ p(y|x, \theta) = \mathcal{N}(y|\mu(x), \sigma^2(x)) \]
\[ \mu = w^T x \]
Logistic Regression

- A discriminative model

\[ p(y|x, w) = \text{Ber}(y|\mu(x)) \]

- \( \mu(x) = \mathbb{E}[y|x] = p(y = 1|x) \)
- \( \mu(x) = \text{sigm}(w^T x) \)
- sigm is sigmoid function
Sigmoid function

- Definition

\[ \text{sigm}(\eta) \triangleq \frac{1}{1 + \exp(-\eta)} = \frac{e^\eta}{e^\eta + 1} \]
Logistic Regression

• A discriminative model

\[ p(y|x, w) = \text{Ber}(y|\mu(x)) \]

- \( \mu(x) = \mathbb{E}[y|x] = p(y = 1|x) \)
- \( \mu(x) = \text{sigm}(w^T x) \)
- \text{sigm} is sigmod function
- \( p(y|x, w) = \text{Ber}(y|\text{sigm}(w^T x)) \)

\[
\text{sigm}(\eta) \triangleq \frac{1}{1 + \exp(-\eta)} = \frac{e^\eta}{e^\eta + 1}
\]
Logistic Regression

• Definition and assumption
• Understand parameter estimation, derivation is not required
Bayesian Inference

\[ p(h \mid d) = \frac{p(d \mid h) p(h)}{\sum_{h' \in H} p(d \mid h') p(h')} \]
Bayesian Inference

\[ p(h \mid d) = \frac{p(d \mid h) p(h)}{\sum_{h' \in H} p(d \mid h') p(h')} \]

- Hypothesis space: H
- Prior \( p(h) \)
- Likelihood \( p(D \mid h) \)
- Computing posterior \( p(h \mid D) \)
Bayesian Inference

• What are prior, likelihood, posterior?
• Occam’s razor
• Posterior predictive distribution
• Maximum a posteriori (MAP)
Naïve Bayes

- Document classification example
- \( Y \in \{1, \ldots, C\}, \; x \in \{0, 1\}^d \)
  - \( Y \in \{\text{spam, urgent, normal}\} \)
  - \( x_i = 1 \) (word \( i \) is present in message)
Bayes Rules

\[ p(y = c | x) = \frac{p(x | y = c)p(y = c)}{\sum_{c'} p(x | y = c')p(y = c')} \]

- **Class posterior**
- **Class-conditional density**
- **Class prior**
- **Normalization constant**
Naïve Bayes

• Definition
• Know how to build a NB classifier and predict on new samples
Decision Tree

• Play tennis?

```
Decision	Tree

Outlook

Sunny

Humidity

High

No

Normal

Yes

Overcast

Yes

Rain

Wind

Strong

No

Weak

Yes
```
Top-Down Induction of Decision Trees

• Which attribute is best?
Decision tree

• Definition

• Know how to build a decision tree based on information gain and predict on new samples
Monte Carlo Approximation

• In general, computing the distribution of a function of an random variable using the change of variable is difficult.

• A powerful way:
  – Generate samples from the distribution \( x_1, \ldots, x_S \)
  – Use Monte Carlo to approximate the expected value of any function of a random variable

\[
\mathbb{E}[f(X)] = \int f(x)p(x)dx \approx \frac{1}{S} \sum_{s=1}^{S} f(x_s)
\]
Conjugate priors

• For simplicity, we will mostly focus on a special kind of prior which has nice mathematical properties.

• A prior $p(\theta)$ is said to be conjugate to a likelihood $p(D|\theta)$ if the corresponding posterior $p(\theta|D)$ has the same functional form as $p(\theta)$.
Conjugate priors: The beta-Bernoulli model

• Consider the probability of heads, given a sequence of $N$ coin tosses, $X_1, \ldots, X_N$.

• Likelihood

$$p(D|\theta) = \prod_{n=1}^{N} \theta^{X_n} (1 - \theta)^{1-X_n} = \theta^{N_1} (1 - \theta)^{N_0}$$

• Natural conjugate prior is the Beta distribution

$$p(\theta) = Be(\theta|\alpha_1, \alpha_0) \propto \theta^{\alpha_1-1} (1 - \theta)^{\alpha_0-1}$$

• Posterior is also Beta, with updated counts

$$p(\theta|D) = Be(\theta|\alpha_1 + N_1, \alpha_0 + N_0) \propto \theta^{\alpha_1-1+N_1} (1 - \theta)^{\alpha_0-1+N_0}$$
Bayesian Model Selection

- Cross-validation
  - Divide training set into N partitions
  - Train on N-1 partitions, and evaluate on the rest
  - In total, fitting the model for N times
Bayesian Model Selection

• Compute posterior

\[ p(m|\mathcal{D}) = \frac{p(\mathcal{D}|m)p(m)}{\sum_{m \in \mathcal{M}} p(m, \mathcal{D})} \]

• Then compute MAP

\[ \hat{m} = \arg \max p(m|\mathcal{D}) \]
Bayesian Model Selection

• Compute posterior

\[ p(m|\mathcal{D}) = \frac{p(\mathcal{D}|m)p(m)}{\sum_{m \in \mathcal{M}} p(m, \mathcal{D})} \]

• Uniform prior over models

\[ p(m) \propto 1 \]

• Then we are picking the model which maximizes

\[ p(\mathcal{D}|m) = \int p(\mathcal{D}|\theta)p(\theta|m)d\theta \quad \text{Marginal likelihood, Integrated likelihood, Or evidence} \]
Bayes Factors

• To compare two models, use posterior odds

\[ O_{ij} = \frac{p(M_i|D)}{p(M_j|D)} = \frac{p(D|M_i)p(M_i)}{p(D|M_j)p(M_j)} \]

• The Bayes factor is a Bayesian version of a likelihood ratio test, that can be used to compare models of different complexity
Unbiased estimators

• The bias of an estimator is defined as
  \[
  \text{bias}(\hat{\theta}) = E\left[\hat{\theta}(D) - \theta | D \sim \theta\right]
  \]

• An estimator is unbiased if bias=0.
Consistent estimators

• An estimator is consistent if it converges (in probability) to the true value with enough data

\[ P(|\hat{\theta}(D) - \theta| > \epsilon |D \sim \theta) \to 0 \text{ as } |D| \to \infty \]

• MLE is a consistent estimator.
Feature Selection

• If predictive accuracy is the goal, often best to keep all predictors and use L2 regularization

• We often want to select a subset of the inputs that are “most relevant” for predicting the output, to get sparse models – interpretability, speed, possibly better predictive accuracy
Bayesian statistics and frequentist statistics

- Monte Carlo
- Conjugate prior
- Bayesian model selection
- Unbiased, consistent estimators
- How to do feature selection
- Difference between Bayesian and frequentist
Perceptron

• Weighted combination
  – The output of the neuron is a linear combination of the inputs

• Decision Function
  – At the end the results are combined into

\[ f(x) = \sigma \left( \sum_{i=1}^{n} w_i x_i + b \right) \]
Learning Goal: Linear Separation

\[ f(x) = \langle w, x \rangle + b \]
Perceptron Algorithm

• Nothing happens if we classify \((x_i, y_i)\) correctly
• If we see incorrectly classified observation we update \(w\) and \(b\)
• Positive reinforcement of observations

\[
\begin{align*}
\text{argument:} & \quad X := \{x_1, \ldots, x_m\} \subset \mathcal{X} \text{ (data)} \\
& \quad Y := \{y_1, \ldots, y_m\} \subset \{\pm 1\} \text{ (labels)} \\
\text{function} & \quad (w, b) = \text{Perceptron}(X, Y) \\
& \quad \text{initialize } w, b = 0 \\
\text{repeat} & \\
& \quad \text{Pick } (x_i, y_i) \text{ from data} \\
& \quad \quad \text{if } y_i(w \cdot x_i + b) \leq 0 \text{ then} \\
& \quad \quad \quad \quad \quad w' = w + y_ix_i \\
& \quad \quad \quad \quad \quad b' = b + y_i \\
& \quad \quad \text{until } y_i(w \cdot x_i + b) > 0 \text{ for all } i
\end{align*}
\]
Kernels

• Definition

• A kernel function \( k : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \) is a symmetric function in its arguments for which the following property holds

\[
k(x, x') = \langle \Phi(x), \Phi(x') \rangle \quad \text{for some feature map } \Phi
\]
Some choices of kernel functions

Linear  \[ \langle x, x' \rangle \]
Laplacian RBF  \[ \exp(-\lambda \| x - x' \|) \]
Gaussian RBF  \[ \exp(-\lambda \| x - x' \|^2) \]
Polynomial  \[ (\langle x, x' \rangle + c)^d, \; c \geq 0, \; d \in \mathbb{N} \]
Kernel Perceptron

argument: \( X := \{x_1, \ldots, x_m\} \subset \mathcal{X} \) (data)  
\[ Y := \{y_1, \ldots, y_m\} \subset \{\pm 1\} \) (labels) 

function \( f = \text{Perceptron}(X, Y, \eta) \)
  initialize \( f = 0 \)
  repeat
    Pick \((x_i, y_i)\) from data
    if \( y_i f(x_i) \leq 0 \) then
      \[ f(\cdot) \leftarrow f(\cdot) + y_i k(x_i, \cdot) + y_i \]
      until \( y_i f(x_i) > 0 \) for all \( i \)
  end

Important detail
\[ w = \sum_j y_j \phi(x_j) \] and hence \( f(x) = \sum_j y_j k(x_j, x) + b. \]
Perceptron

• Definition
• Perceptron algorithm
• Kernel version, and why Kernel
Support Vector Machine (SVM)

- Reasons:
  - Intuition
  - Theoretical guarantee (skip here)
  - In practical tasks: SVM became famous when, using images as input, it gave accuracy comparable to neural-network with hand-designed features in a handwriting recognition task.
Dual SVM derivation

Original optimization problem:

\[ \text{minimize}_{w, b} \quad \frac{1}{2} w \cdot w \]
\[ \left( w \cdot x_j + b \right) y_j \geq 1, \quad \forall j \]

Lagrangian:

\[ L(w, \alpha) = \frac{1}{2} w \cdot w - \sum_j \alpha_j \left[ \left( w \cdot x_j + b \right) y_j - 1 \right] \]
\[ \alpha_j \geq 0, \quad \forall j \]

Our goal now is to solve:

\[ \min_{\vec{w}, b} \quad \max_{\vec{\alpha} \geq 0} \quad L(\vec{w}, \vec{\alpha}) \]
Dual for the non-separable case

**Primal:**

\[
\begin{align*}
\text{minimize}_{w,b} & \quad \frac{1}{2} w \cdot w + C \sum_j \xi_j \\
(\mathbf{w} \cdot \mathbf{x}_j + b) y_j & \geq 1 - \xi_j, \quad \forall j \\
\xi_j & \geq 0, \quad \forall j
\end{align*}
\]

**Dual:**

\[
\begin{align*}
\text{maximize}_\alpha & \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \\
\sum_i \alpha_i y_i & = 0 \\
C & \geq \alpha_i \geq 0
\end{align*}
\]

**Solve for w, b, \alpha:**

\[
\begin{align*}
w & = \sum_i \alpha_i y_i \mathbf{x}_i \\
b & = y_k - w \cdot \mathbf{x}_k
\end{align*}
\]

for any \( k \) where \( C > \alpha_k > 0 \)
SVMs

• Definition
• Primal and dual form
• Slack variables, and how to choose the parameters
**Topics:** single hidden layer neural network

- **Hidden layer pre-activation:**
  \[ a(x) = b^{(1)} + W^{(1)}x \]
  \[ (a(x)_i = b^{(1)}_i + \sum_j W^{(1)}_{i,j} x_j) \]

- **Hidden layer activation:**
  \[ h(x) = g(a(x)) \]

- **Output layer activation:**
  \[ f(x) = o \left( b^{(2)} + w^{(2)\top} h^{(1)}x \right) \]
CAPACITY OF NEURAL NETWORK

Topics: universal approximation

• Universal approximation theorem (Hornik, 1991):
  
  ‣ “a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units”

• The result applies for sigmoid, tanh and many other hidden layer activation functions

• This is a good result, but it doesn’t mean there is a learning algorithm that can find the necessary parameter values!
Empirical Risk Minimization

Topics: empirical risk minimization, regularization

• Empirical risk minimization
  - framework to design learning algorithms

\[
\arg\min_{\theta} \frac{1}{T} \sum_{t} l(f(x^{(t)}; \theta), y^{(t)}) + \lambda \Omega(\theta)
\]

• \( l(f(x^{(t)}; \theta), y^{(t)}) \) is a loss function
• \( \Omega(\theta) \) is a regularizer (penalizes certain values of \( \theta \))

• Learning is cast as optimization
  - ideally, we’d optimize classification error, but it’s not smooth
  - loss function is a surrogate for what we truly should optimize (e.g. upper bound)
Backpropagation
Feedforward Neural Networks

• Definition and structure
• Activation functions
• Understand why neural network is powerful and what is the challenge
• Empirical risk minimization
• Backpropagation training: loss function, gradient, regularizer, initialization
**RESTRICTED BOLTZMANN MACHINE**

**Topics:** RBM, visible layer, hidden layer, energy function

![Diagram of a Restricted Boltzmann Machine (RBM)]

Energy function: 
\[ E(x, h) = -h^T W x - c^T x - b^T h \]
\[ = -\sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \]

Distribution: 
\[ p(x, h) = \exp(-E(x, h))/Z \]

(partition function (intractable))
**Topics:** contrastive divergence, negative sample

- Idea:
  1. replace the expectation by a point estimate at $\tilde{x}$
  2. obtain the point $\tilde{x}$ by Gibbs sampling
  3. start sampling chain at $x^{(t)}$
**Topics:** autoencoder, encoder, decoder, tied weights

- Feed-forward neural network trained to reproduce its input at the output layer

![Autoencoder Diagram](image)

**Decoder**

\[
\hat{x} = o(\hat{a}(x)) \\
= \text{sigmoid}(c + W^*h(x))
\]

for binary inputs

**Encoder**

\[
h(x) = g(a(x)) \\
= \text{sigmoid}(b + Wx)
\]
**Topics:** denoising autoencoder

- Idea: representation should be robust to introduction of noise:
  - random assignment of subset of inputs to 0, with probability \( \nu \)
  - Gaussian additive noise
- Reconstruction \( \hat{\mathbf{x}} \) computed from the corrupted input \( \tilde{\mathbf{x}} \)
- Loss function compares \( \hat{\mathbf{x}} \) reconstruction with the noiseless input \( \mathbf{x} \)
Topics: denoising autoencoder

• Idea: representation should be robust to introduction of noise:
  ‣ random assignment of subset of inputs to 0, with probability \( \nu \)
  ‣ Gaussian additive noise

• Reconstruction \( \hat{\mathbf{x}} \) computed from the corrupted input \( \tilde{\mathbf{x}} \)

• Loss function compares \( \hat{\mathbf{x}} \) reconstruction with the noiseless input \( \mathbf{x} \)
SPARSE CODING

Topics: sparse coding

- For each $x^{(t)}$ find a latent representation $h^{(t)}$ such that:
  - it is sparse: the vector $h^{(t)}$ has many zeros
  - we can reconstruct the original input $x^{(t)}$ as well as possible
- More formally:

$$\min_D \frac{1}{T} \sum_{t=1}^{T} \min_{h^{(t)}} \frac{1}{2} \| x^{(t)} - D h^{(t)} \|_2^2 + \lambda \| h^{(t)} \|_1$$

- we also constrain the columns of $D$ to be of norm 1
  - otherwise, $D$ could grow big while $h^{(t)}$ becomes small to satisfy the prior
- sometimes the columns are constrained to be no greater than 1
Topics: sparse coding

- For each $x^{(t)}$ find a latent representation $h^{(t)}$ such that:
  - it is sparse: the vector $h^{(t)}$ has many zeros
  - we can reconstruct the original input $x^{(t)}$ as well as possible
- More formally:

  \[
  \min_D \frac{1}{T} \sum_{t=1}^{T} \min_{h^{(t)}} \frac{1}{2} \|x^{(t)} - Dh^{(t)}\|_2^2 + \lambda \|h^{(t)}\|_1
  \]

  - we also constrain the columns of $D$ to be of norm 1
    - otherwise, $D$ could grow big while $h^{(t)}$ becomes small to satisfy the prior
  - sometimes the columns are constrained to be no greater than 1
The full learning algorithm

**Topics:** learning algorithm (putting it all together)

- Learning alternates between inference and dictionary learning

- **While** $D$ has not converged
  - find the sparse codes $h(x^{(t)})$ for all $x^{(t)}$ in my training set with ISTA
  - update the dictionary:
    - $A \leftarrow \sum_{t=1}^{T} x^{(t)} h(x^{(t)})^T$
    - $B \leftarrow \sum_{t=1}^{T} h(x^{(t)}) h(x^{(t)})^T$
    - run block-coordinate descent algorithm to update $D$
Unsupervised Learning with Neural Networks

• Definition, usage, and difference
• Understand training objective and algorithm
• Advantage and disadvantage of each model
What is Principal Component Analysis?

• Principal component analysis (PCA)
  – Reduce the dimensionality of a data set by finding a new set of variables, smaller than the original set of variables
  – Retains most of the sample's information.
  – Useful for the compression and classification of data.

• By information we mean the variation present in the sample, given by the correlations between the original variables.
  – The new variables, called principal components (PCs), are uncorrelated, and are ordered by the fraction of the total information each retains.
Geometric picture of principal components (PCs)

- the 1\textsuperscript{st} PC \( Z_1 \) is a minimum distance fit to a line in \( X \) space
- the 2\textsuperscript{nd} PC \( Z_2 \) is a minimum distance fit to a line in the plane perpendicular to the 1\textsuperscript{st} PC

PCs are a series of linear least squares fits to a sample, each orthogonal to all the previous.
Algebraic definition of PCs

Given a sample of $n$ observations on a vector of $p$ variables

$$\{x_1, x_2, \cdots, x_n\} \in \mathbb{R}^p$$

define the first principal component of the sample by the linear transformation

$$z_1 = a_1^T x_j = \sum_{i=1}^{p} a_{i1} x_{ij}, \quad j = 1,2,\cdots,n.$$  

where the vector

$$a_1 = (a_{11}, a_{21}, \cdots, a_{p1})$$

$$x_j = (x_{1j}, x_{2j}, \cdots, x_{pj})$$

is chosen such that $\text{var}[z_1]$ is maximum.
Algebraic derivation of PCs

In general

$$\text{var}[z_k] = a_k^T S a_k = \lambda_k$$

- The $k^{\text{th}}$ largest eigenvalue of $S$ is the variance of the $k^{\text{th}}$ PC.
- The $k^{\text{th}}$ PC $z_k$ retains the $k^{\text{th}}$ greatest fraction of the variation in the sample.
Principal component analysis (PCA)

• Usage and definition
• How to find principal component
• How to use PCA for compression and classification
Singular value decomposition
SVD - Definition

\[ A_{[m \times n]} = U_{[m \times r]} \Sigma_{[r \times r]} (V_{[n \times r]})^T \]

- A: Input data matrix
  - \( m \times n \) matrix (e.g., \( m \) documents, \( n \) terms)
- U: Left singular vectors
  - \( m \times r \) matrix (\( m \) documents, \( r \) concepts)
- \( \Sigma \): Singular values
  - \( r \times r \) diagonal matrix (strength of each ‘concept’)
    \( r : \) rank of the matrix \( A \)
- V: Right singular vectors
  - \( n \times r \) matrix (\( n \) terms, \( r \) concepts)
CUR decomposition

• In large-data applications, it is normal for the matrix A being decomposed to be very sparse
  – Documents
• With SVD, even if A is sparse, U and V will be dense
CUR Decomposition

- **Goal:** Express $A$ as a product of matrices $C, U, R$
- Make $\|A - C \cdot U \cdot R\|_F$ small
- “Constraints” on $C$ and $R$:

$$
\begin{pmatrix}
A \\
A
\end{pmatrix} \approx
\begin{pmatrix}
C \\
U
\end{pmatrix} \cdot
\begin{pmatrix}
R
\end{pmatrix}
$$

Frobenius norm:

$$
\|X\|_F = \sqrt{\sum_{ij} X_{ij}^2}
$$
Singular value decomposition

• Usage, definition and interpretation
• How to use SVD for compression and classification
• Why CUR is better than SVD?
Clustering

• When do we need clustering?
• What are the common challenges?
• What’s the difference among the algorithms?
K means

- The basic idea is to describe each cluster by its mean value.

- Goal: assign data to clusters and define these clusters with their means.
Coordinate descent

\[ F(z_{1:N}, m_{1:k}) = \frac{1}{2} \sum_{n=1}^{N} ||x_n - m_{z_n}||^2 \]

Holding the means fixed, assigning each point to its closest mean minimizes \( F \) with respect to \( z_{1:N} \).

Holding the assignments fixed, computing the centroids of each cluster minimizes \( F \) with respect to \( m_{1:k} \).

Thus, \( k \)-means is a coordinate descent algorithm.

However, it finds a local minimum. (Multiple restarts are often necessary.)
K means

• Definition and algorithm
• How to choose K
Hierarchical clustering

• Hierarchical clustering is a widely used data analysis tool.
• The idea is to build a binary tree of the data that successively merges similar groups of points.
• Visualizing this tree provides a useful summary of the data.
Group similarity

- Given a distance measure between points, the user has many choices for how to define intergroup similarity.
- Three most popular choices
  - *Single-linkage:* the similarity of the closest pair
    \[
    d_{SL}(G, H) = \min_{i \in G, j \in H} d_{i,j}
    \]
  - *Complete linkage:* the similarity of the furthest
    \[
    d_{CL}(G, H) = \max_{i \in G, j \in H} d_{i,j}
    \]
  - *Group average:* the average similarity between groups
    \[
    d_{GA} = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{j \in H} d_{i,j}
    \]
Hierarchical clustering

• Know how to construct the dendrogram with different group similarity metric
Spectral Clustering

- Algorithms that cluster points using eigenvectors of matrices derived from the data
- Obtain data representation in the low-dimensional space that can be easily clustered
- Difficult to understand
- Easy to implement
Spectral Clustering

• The intuition and algorithm
• How to decide on the cluster number
• How to do K-way clustering
The Markov Assumption

\[ P(X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m) \]
\[ = P(X_1 = x_1) \prod_{j=2}^{m} P(X_j = x_j | X_1 = x_1, \ldots, X_{j-1} = x_{j-1}) \]
\[ = P(X_1 = x_1) \prod_{j=2}^{m} P(X_j = x_j | X_{j-1} = x_{j-1}) \]

- The first equality is exact (by the chain rule).
- The second equality follows from the Markov assumption: for all \( j = 2 \ldots m, \)

\[ P(X_j = x_j | X_1 = x_1, \ldots, X_{j-1} = x_{j-1}) = P(X_j = x_j | X_{j-1} = x_{j-1}) \]
Hidden Markov Models (HMMs)

In HMMs, we assume that:

$$P(X_1 = x_1, \ldots, X_m = x_m, S_1 = s_1, \ldots, S_m = s_m)$$

$$= P(S_1 = s_1) \prod_{j=2}^{m} P(S_j = s_j | S_{j-1} = s_{j-1}) \prod_{j=1}^{m} P(X_j = x_j | S_j = s_j)$$
Formally

- The model takes the following form:

\[ p(x_1 \ldots x_m, s_1 \ldots s_m; \theta) = t(s_1) \prod_{j=2}^{m} t(s_j | s_{j-1}) \prod_{j=1}^{m} e(x_j | s_j) \]

- Parameters in the model:

1. Initial state parameters \( t(s) \) for \( s \in \{1, 2, \ldots, k\} \)
2. Transition parameters \( t(s' | s) \) for \( s, s' \in \{1, 2, \ldots, k\} \)
3. Emission parameters \( e(x | s) \) for \( s \in \{1, 2, \ldots, k\} \) and \( x \in \{1, 2, \ldots, o\} \)
The Viterbi Algorithm: Backpointers

- **Initialization:** for $s = 1 \ldots k$
  \[ \pi[1, s] = t(s)e(x_1|s) \]

- For $j = 2 \ldots m$, $s = 1 \ldots k$:
  \[ \pi[j, s] = \max_{s' \in \{1 \ldots k\}} [\pi[j - 1, s'] \times t(s|s') \times e(x_j|s)] \]
  and
  \[ bp[j, s] = \arg \max_{s' \in \{1 \ldots k\}} [\pi[j - 1, s'] \times t(s|s') \times e(x_j|s)] \]

- The $bp$ entries are backpointers that will allow us to recover the identity of the highest probability state sequence.
Hidden Markov Models (HMMs)

- Definition and assumption
- Know how to train the model and how to decode
Log-Linear Models

We have sets $\mathcal{X}$ and $\mathcal{Y}$: we will assume that $\mathcal{Y}$ is a finite set. We have a feature-vector definition $\phi : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d$. We also assume a parameter vector $w \in \mathbb{R}^d$. Given these definitions,

$$p(y|x; w) = \frac{\exp \left( w \cdot \phi(x, y) \right)}{\sum_{y' \in \mathcal{Y}} \exp \left( w \cdot \phi(x, y') \right)}$$

This is the conditional probability of $y$ given $x$, under parameters $w$. 
MEMMs use the following decomposition:

\[
p(s_1, s_2 \ldots s_m | x_1 \ldots x_m) = \prod_{i=1}^{m} p(s_i | s_1 \ldots s_{i-1}, x_1 \ldots x_n)
\]

\[
= \prod_{i=1}^{m} p(s_i | s_{i-1}, x_1 \ldots x_n)
\]

The first step is exact (by the chain rule)

The second step follows from an independence assumption, i.e., that for all \( i \),

\[
p(s_i | s_1 \ldots s_{i-1}, x_1 \ldots x_m) = p(s_i | s_{i-1}, x_1 \ldots x_m)
\]
Using Log-Linear Models

We then model each term using a log-linear model:

\[
p(s_i|s_{i-1}, x_1 \ldots x_m) = \frac{\exp \left( w \cdot \phi(x_1 \ldots x_m, i, s_{i-1}, s_i) \right)}{\sum_{s' \in S} \exp \left( w \cdot \phi(x_1 \ldots x_m, i, s_{i-1}, s') \right)}
\]

Here \( \phi(x_1 \ldots x_m, i, s, s') \) is a feature vector where:

- \( x_1 \ldots x_m \) is the sequence of \( m \) words to be tagged
- \( i \) is the position to be tagged (any value from 1 \ldots m)
- \( s \) is the previous state
- \( s' \) is the new state
Maximum-Entropy Markov Models (MEMMMs)

- Definition and assumption
- Difference between MEMMMs and HMMs
Conditional Random Fields (CRFs)

- Notation: for convenience we’ll use $\underline{x}$ to refer to the sequence of input words, $x_1 \ldots x_m$, and $\underline{s}$ to refer to a sequence of possible states, $s_1 \ldots s_m$. The set of possible states is $S$. We use $S^m$ to refer to the set of all possible state sequences (we have $|S^m| = |S|^m$).

- We’re again going to build a model of

$$p(s_1 \ldots s_m | x_1 \ldots x_m) = p(s | x)$$
Conditional Random Fields (CRFs)

- We use $\Phi(x, s) \in \mathbb{R}^d$ to refer to a feature vector for an entire state sequence.
- We then build a giant log-linear model,

$$p(s|x; w) = \frac{\exp (w \cdot \Phi(x, s))}{\sum_{s' \in S^m} \exp (w \cdot \Phi(x, s'))}$$

- The model is “giant” in the sense that: 1) the space of possible values for $s$, i.e., $S^m$, is huge. 2) The normalization constant (denominator in the above expression) involves a sum over a huge number of possibilities (i.e., all members of $S^m$).
Conditional Random Fields (CRFs)

• Definition
• Difference among CRFs, MEMMs, HMMs
Gaussian Mixtures

• Linear super-position of Gaussians

\[ p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \]

• Normalization and positivity require

\[ \sum_{k=1}^{K} \pi_k = 1 \quad 0 \leq \pi_k \leq 1 \]

• Can interpret the mixing coefficients as prior probabilities

\[ p(x) = \sum_{k=1}^{K} p(k)p(x|k) \]
Gaussian Mixtures

- What is the generative process
- How to estimate the parameters with EM