Day 10: Review

Introduction to Machine Learning Summer School
June 18, 2018 - June 29, 2018, Chicago

Instructor: Suriya Gunasekar, TTI Chicago

29 June 2018
Review
Supervised learning – key questions

• **Data**: what kind of data can we get? how much data can we get?

• **Model**: what is the correct model for my data? – want to minimize the effort put into this question!

• **Training**: what resources - computation/memory - does the algorithm need to estimate the model \( \hat{f} \)?

• **Testing**: how well will \( \hat{f} \) perform when deployed? what is the computational/memory requirement during deployment?
Linear regression

• Input $x \in \mathcal{X} \subset \mathbb{R}^d$, output $y \in \mathbb{R}$, want to learn $f: \mathcal{X} \to \mathbb{R}$

• Training data $S = \{(x^{(i)}, y^{(i)}): i = 1, 2, ..., N\}$

• Parameterize candidate $f: \mathcal{X} \to \mathbb{R}$ by linear functions,
  \[ \mathcal{H} = \{x \to w \cdot x: w \in \mathbb{R}^d\} \]

• Estimate $w$ by minimizing loss on training data

\[ \hat{w} = \arg\min_w J^L_S(w) = \sum_{i=1}^{N} (w \cdot x^{(i)} - y^{(i)})^2 \]

  \( J^L_S(w) \) is convex in $w \rightarrow$ minimize $J^L_S(w)$ by setting gradient to 0

  \[ \nabla_w J^L_S(w) = \sum_{i=1}^{N} (w \cdot x^{(i)} - y^{(i)})x^{(i)} \]

  \( \text{Closed form solution } \hat{w} = (X^T X)^{-1} X y \)

• Can get non-linear functions by mapping $x \rightarrow \phi(x)$ and doing linear regression on $\phi(x)$
Overfitting

• For same amount of data, more complex models (e.g., higher degree polynomials) overfit more

• or need more data to fit more complex models

• complexity ≈ number of parameters

Model selection

• m model classes \{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m\}

• \( S = S_{\text{train}} \cup S_{\text{val}} \cup S_{\text{test}} \)

• Train on \( S_{\text{train}} \) to pick best \( \hat{f}_r \in \mathcal{H}_r \)

• Pick \( \hat{f}^* \) based on validation loss on \( S_{\text{val}} \)

• Evaluate test loss \( L_{S_{\text{test}}} (\hat{f}^*) \)
Regularization

• Complexity of model class can also be controlled by norm of parameters – smaller range of values allowed

• Regularization for linear regression

\[
\arg\min_w J_S^{LS}(w) + \lambda \|w\|_2^2
\]

\[
\arg\min_w J_S^{LS}(w) + \lambda \|w\|_1
\]

• Again do model selection to pick \( \lambda \) – using \( S_{val} \) or cross-validation
Classification

• Output $y \in Y$ takes discrete set of values, e.g., $Y = \{0,1\}$ or $Y = \{-1,1\}$ or $Y = \{\text{spam, nospam}\}$
  o Unlike regression, label-values do not have meaning

• Classifiers divide the space of input $X$ (often $\mathbb{R}^d$) to “regions” where each region is assigned a label

• Non-parametric models
  o k-nearest neighbors – regions defined based on nearest neighbors
  o decision trees – structured rectangular regions

• Linear models – classifier regions are halfspaces
Classification – logistic regression

Logistic loss
\[ \ell(f(x), y) = \log(1 + \exp(-f(x)y)) \]

- \( X = \mathbb{R}^d, \ Y = \{-1, 1\}, S = \{(x^{(i)}, y^{(i)}): i = 1, 2, \ldots, N\} \)
- Linear model \( f(x) = f_w(x) = w \cdot x \)
- Output classifier \( \hat{y}(x) = \text{sign}(w \cdot x) \)
- Empirical risk minimization:
  \[ \hat{w} = \arg\min_w \sum_i \log \left(1 + \exp(-w \cdot x^{(i)}y^{(i)})\right) \]
- Alternative, probabilistic formulation:
  \[ \Pr(y = 1|x) = \frac{1}{1 + \exp(-w \cdot x)} \]
- Multi-class generalization: \( Y = \{1, 2, \ldots, m\} \)
  \[ \Pr(y|x) = \frac{\exp(-w_y \cdot x)}{\sum_{y'} \exp(-w_{y'} \cdot x)} \]
- Can again get non-linear decision boundaries by mapping \( x \to \phi(x) \)
Classification – maximum margin classifier

Separable data

• Original formulation

\[ \hat{w} = \arg\max_{w \in \mathbb{R}^d} \min_i \frac{y^{(i)} w . x^{(i)}}{\|w\|} \]

• Fixing \( \|w\| = 1 \)

\[ \hat{w} = \arg\max_w \min_i y^{(i)} (w . x^{(i)}) \quad \text{s.t.} \quad \|w\| = 1 \]

• Fixing \( \min_i y^{(i)} w . x^{(i)} = 1 \)

\[ \hat{w} = \arg\min_w \|w\|^2 \quad \text{s.t.} \quad \forall i, y^{(i)} (w . x^{(i)}) \geq 1 \]

Slack variables for non-separable data

\[ \hat{w} = \arg\min_w \|w\|^2 + \lambda \sum \xi_i \quad \text{s.t.} \quad \forall i, y^{(i)} (w . x^{(i)}) \geq 1 - \xi_i \]

\[ = \arg\min_w \|w\|^2 + \lambda \sum_{\xi_i \geq 0} \max \left(0, 1 - y^{(i)} (w . x^{(i)})\right) \]
Kernel trick

• Using representor theorem $\mathbf{w} = \sum_{i=1}^{N} \beta_i \mathbf{x}^{(i)}$

\[
\begin{align*}
\min_{\mathbf{w}} \quad & \|\mathbf{w}\|^2 + \lambda \sum_i \max(0, 1 - y^{(i)} \mathbf{w} \cdot \mathbf{x}^{(i)}) \\
\equiv \min_{\beta \in \mathbb{R}^N} & \quad \beta^T \mathbf{G} \beta + \lambda \sum_i \max(0, 1 - y^{(i)}(\mathbf{G} \beta)_i)
\end{align*}
\]

$\mathbf{G} \in \mathbb{R}^{N \times N}$ with $G_{ij} = \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$ is called the gram matrix

• Optimization depends on $\mathbf{x}^{(i)}$ only through $G_{ij} = \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$

• For prediction $\hat{\mathbf{w}} \cdot \mathbf{x} = \sum_i \beta_i \mathbf{x}^{(i)} \cdot \mathbf{x}$, we again only need $\mathbf{x}^{(i)} \cdot \mathbf{x}$

• Function $K(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'$ is called the Kernel

• When learning non-linear classifiers using feature transformations $\mathbf{x} \rightarrow \phi(\mathbf{x})$ and $f_w(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x})$
  
  o Classifier fully specified in terms of $K_\phi(\mathbf{x}, \mathbf{x}') = K(\phi(\mathbf{x}), \phi(\mathbf{x}'))$

  o $\phi(\mathbf{x})$ itself can be very very high dimensional (maybe even infinite dimensional)
  
  $\rightarrow$ e.g., polynomial kernels, RBF kernel
Optimization

• ERM+regularization optimization problem

\[
\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J_{S}^{\lambda}(\mathbf{w}): = \sum_{i=1}^{N} \ell(\mathbf{w} \cdot \phi(x^{(i)}), y^{(i)}) + \lambda \|\mathbf{w}\|
\]

• If \( J_{S}^{\lambda}(\mathbf{w}) \) is convex in \( \mathbf{w} \), then \( \hat{\mathbf{w}} \) is optimum if and only if gradient at \( \hat{\mathbf{w}} \) is 0, i.e., \( \nabla J_{S}^{\lambda}(\hat{\mathbf{w}}) = 0 \)

• Gradient descent: start with initialization \( \mathbf{w}^0 \) and iteratively update
  
  \[
  \mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta^{t} \nabla J_{S}^{\lambda}(\mathbf{w}^{t})
  \]
  
  \[
  \text{where} \quad \nabla J_{S}^{\lambda}(\mathbf{w}^{t}) = \sum_{i} \nabla \ell(\mathbf{w}^{t} \cdot \phi(x^{(i)}), y^{(i)}) + \lambda \nabla \|\mathbf{w}^{t}\|
  \]

• Stochastic gradient descent:
  
  \[
  \mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta^{t} \hat{\nabla}^{(i)} J_{S}^{\lambda}(\mathbf{w}^{t})
  \]
  
  \[
  \text{where} \quad \hat{\nabla}^{(i)} J_{S}^{\lambda}(\mathbf{w}^{t}) = \nabla \ell(\mathbf{w}^{t} \cdot \phi(x^{(i)}), y^{(i)}) + \frac{\lambda}{N} \nabla \|\mathbf{w}^{t}\| \quad \text{for a random sample} \ (x^{(i)}, y^{(i)})
  \]
Other classification models

• **Optimal unrestricted predictor**
  - Regression + squared loss: \( f^{**}(x) = \mathbb{E}[y|x] \)
  - Classification + 0-1 loss: \( \hat{y}^{**}(x) = \text{argmax}_c \Pr(y = c|x) \)

• **Discriminative models**: directly model \( \Pr(y|x) \), e.g., logistic regression

• **Generative models**: model full joint distribution \( \Pr(y, x) = \Pr(x|y) \Pr(y) \)

• **Why generative models?**
  - One conditional might be simpler to model with prior knowledge, e.g., compare specifying \( \Pr(\text{image}|\text{digit} = 1) \) vs \( \Pr(\text{digit} = 1|\text{image}) \)
  - Naturally handles missing data

• **Two examples of generative models**
  - Naïve Bayes classifier – digit recognition, document classification
  - Hidden Markov model – POS tagging
Other classifiers

- **Naïve Bayes classifier:** with \( d \) features \( x = [x_1, x_2, ..., x_d] \) where each \( x_1, x_2, ..., x_d \) can take one of \( K \) values \( \rightarrow C K^d \) parameters
  - NB assumption: features are independent given class \( y \) \( \rightarrow C K d \) params.
    \[
    \Pr(x_1, x_2, ..., x_d | y) = \Pr(x_1 | y) \Pr(x_2 | y) ... \Pr(x_d | y) = \prod_{k=1}^{d} \Pr(x_k | y)
    \]
  - Training amounts to averaging samples across classes

- **Hidden Markov model:** variable length input/observations \( \{x_1, x_2, ..., x_m\} \) (e.g., words) and variable length output/state \( \{y_1, y_2, ..., y_m\} \) (e.g., tags)
  - HMM assumption: a) current state conditioned on immediate previous state is conditionally independent of all other variables, and (b) current observation conditioned on current state is conditionally independent of all other variables.
    \[
    \Pr(x_1, x_2, ..., x_m, y_1, y_2, ..., y_m) = \Pr(y_1) \Pr(x_1 | y_1) \prod_{k=2}^{m} \Pr(y_k | y_{k-1}) \Pr(y_k | x_k)
    \]
  - Parameters estimated using MLE dynamic programming
Feed-Forward Neural Networks

Architecture:

- Directed Acyclic Graph $G(V,E)$. Units (neurons) indexed by vertices in $V$.
  - “Input Units” $v_1 \ldots v_d \in V$ : no incoming edges have value $o[v_i] = x_i$
  - Each edge $u \rightarrow v$ has weight $W[u \rightarrow v]$
    - Pre-activation $a[v] = \sum_{u \rightarrow v \in E} W[u \rightarrow v] o[u]$
    - Output value $o[v] = \sigma(a[v])$
  - “Output Unit” $v_{out} \in V$, $f_W(x) = a[v_{out}]$

Figure credit: Nati Srebro
Feed forward fully connected network

- \( L \) hidden layers with layer \( l \) having \( d_l \) hidden units
- Parameters:
  - for each intermediate layer \( W^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l} \) where \( d_0 = d \)
  - final layer weights \( w^{(L+1)} \in \mathbb{R}^{d_L} \)
- For 2-hidden layer \( f_W(x) = w^{(3)\top} \sigma \left( W^{(2)} \sigma (W^{(1)} x) \right) \). More generally,

\[
f_W(x) = w^{(L+1)\top} \sigma \left( W^{(L)} \cdots \sigma (W^{(2)} \sigma (W^{(1)} x)) \right)
\]
Back-Propagation

- Efficient calculation of $\nabla_W \ell(f_W(x), y)$ using chain rule

\[ a[v] = \sum_{u \rightarrow v \in E} W^{(t)}[u \rightarrow v] o[u] \]
\[ o[v] = \sigma(a[v]) \]
\[ z[v_{out}] = \ell'(a[v_{out}], y) \]
\[ z[u] = \sigma'(a[u]) \sum_{u \rightarrow v} W^{(t)}[u \rightarrow v] z[v] \]

- Forward propagation: calculate activations $a[v]$ and outputs $o[v]$
- Backward propagation: calculate $z[v] = \frac{\partial \ell(f_W(x), y)}{\partial a[v]}$
- Gradient descent update: using $\frac{\partial \ell(f_W(x), y)}{\partial W^{(t)}[u \rightarrow v]} = z[v] o[u]$

\[ W^{(t+1)}[u \rightarrow v] = W^{(t+1)}[u \rightarrow v] - \eta^{(t)} \frac{\partial \ell(f_W(x), y)}{\partial W^{(t)}[u \rightarrow v]} \]
Optimization for NN training

• Check
  o Add gradCheck()
  o Randomly permute data for SGD sequence

• Choose activations to avoid
  o Gradient clipping
  o Gradient explosion

• SGD “knobs” in NN training
  o Initialization → Kaiming/Xavier, or warm start initialization.
  o Step size/learning rate → very important to tune based on training/ validation loss
  o SGD variants
    ▪ Momentum for SGD → usually added with SGD (default parameter momentum=0.9 often works well)
    ▪ Adaptive variants of SGD → common alternative to SGD+momentum is Adam with $\beta_2 \gg \beta_1$, e.g., $\beta_2 = 0.999, \beta_1 = 0.9$
  o Mini-batch SGD → ~128 common
  o Batch normalization → use batch normalization
Regularization in NN

• **Explicit regularization**
  
  o Data augmentation → Augment training data with known invariances/noise models → very effective
    
    ▪ think of what is the right data augmentation for your problem
  
  o Weight decay → \( \arg \min_W L_S(f_W) + \frac{\lambda}{2} \|W\|^2 \)
    
    ▪ tune step sizes/\( \lambda \) parameter

• Dropout → Randomly (temporarily) remove \( p \) fraction of the units in each step of SGD → usually very useful

• **Early stopping**

• **Choice of architecture** affects validation performance/generalization!

• Many optimization choices also affect validation performance— unlike convex optimization problems with a unique global minimum, where optimization algorithm only changes the speed/computation of training → Not well understood phenomenon
  
  o Keep in mind while making choices in previous slides
NN architectures – CNNs

Figures taken from lecture slides at http://cs231n.stanford.edu/slides/2017/
NN architectures – CNNs

- Each convolution layer has input of size $W_{\text{in}} \times H_{\text{in}} \times D_{\text{in}}$
- **Hyperparameters**: Number of filters $D_{\text{out}}$; Size of filters $K_1 \times K_2$; Stride $S$; Zero padding $P$
- **Parameters**: $K_1 \times K_2 \times D_{\text{in}} \times D_{\text{out}}$
- **Output**: $W_{\text{out}} \times H_{\text{out}} \times D_{\text{out}}$ where
  - $W_{\text{out}} = (W_{\text{in}} - K_1 - 2P)/S + 1$
  - $H_{\text{out}} = (H_{\text{in}} - K_2 - 2P)/S + 1$

Figures taken from lecture slides at http://cs231n.stanford.edu/slides/2017/
CNNs

• **Typical layers**
  - Convolution+ReLU
  - Max-pooling
  - Final few fully connected layers

• **Common datasets**
  - MNIST (small)
  - CIFAR-10 & CIFAR-100
  - ImageNet
  - MS COCO

• **Tip:** Try warm-start initialization from models pre-trained on ImageNet

Figures taken from lecture slides at http://cs231n.stanford.edu/slides/2017/
Residual Networks

- \( h_l = h_{l-1} + ReLU(Conv(h_{l-1})) \)
- Avoids gradient saturation
- Enabled training of really deep networks
  - Typical choice is 152 layers
  - 1000+ layers have been trained with ResNets
- Can also extend for other architectures like FCNs/RNNs

Figures taken from lecture slides at http://cs231n.stanford.edu/slides/2017/
NN architectures – RNNs

• **Input:** each example is a sequence
  \[
  [x_1, x_2, \ldots, x_n \in \mathbb{R}^d]
  \]

• **Labels:** can be single label \( y \) or another sequence

• **Output of RNNs:** \([h_1, h_2, \ldots, h_n \in \mathbb{R}^{d'}]\)

• **Note:** this is just one example, the training dataset will contain many such examples

• **RNN model:** For \( i = 1, 2, \ldots, n \)
  \[
  h_i = \tanh(Wx_i + Vh_{i-1})
  \]

[Goodfellow et al.]
NN architectures – RNNs

- **RNN model:** For $i = 1, 2, \ldots, n$
  \[ h_i = \tanh(Wx_i + Vh_{i-1}) \]
- $h_n = \tanh(Wx_n + V \tanh(Wx_{n-1} + V (\ldots + \tanh(Wx_1 + Vh_0))))$
  - Like fully connected networks, but parameters are reused
- **Loss** $\ell([h_1, h_2, \ldots, h_n], y)$

- Can create deeper networks by using $[h_1, h_2, \ldots, h_n \in \mathbb{R}^{d'}]$ as sequential input to next layer
NN Architectures LSTMs

- RNN produces a sequence of output vectors
  \[ x_1 \ldots x_N \quad \rightarrow \quad h_1 \ldots h_N \]

- LSTM produces “memory cell vectors” along with output
  \[ x_1 \ldots x_N \quad \rightarrow \quad c_1 \ldots c_N, \quad h_1 \ldots h_N \]

- These \( c_1 \ldots c_N \) enable the network to keep or drop information from previous states.
NN Architectures LSTMs

- Simple RNNs

- In LSTMs, each time frame associated with a complex cell

Figures taken from blog post on LSTMs by C. Olah
NN Architecture LSTMs

- Cell state $c_t$
- Forget gate $q_t$
- Input gate
- Cell state update
- Output gate

See lecture slides for exact equations

Figures taken from blog post on LSTMs by C. Olah
NN architectures – encoder-decoder

- **Encoder RNN**: First encodes in the input and captures the context in $\xi$
- **Decoder RNN**: decodes the output from $\xi$
- **Decoder with attention**: instead of relying just on final context $\tilde{\xi}$, use a linear combination of all the hidden states in the encoder (not depicted in figure)

See lecture slides for exact equations

Slide credit: Greg Shaknarovich
Ensembles

• **Reduce bias:**
  - build **ensemble of low-variance, high-bias predictors** sequentially to reduce bias
  - AdaBoost: binary classification, exponential surrogate loss

• **Reduce variance:**
  - build **ensemble of high-variance, low-bias predictors** in parallel and use randomness and averaging to reduce variance
  - random forests, bagging

• **Problems**
  - Computationally expensive (train and test time)
  - Often loose interpretability
Bagging: Bootstrap aggregation

Averaging independent models reduces variance without increasing bias.

• But we don’t have independent datasets!
  o Instead take repeated bootstrap samples from training set $S$

• Bootstrap sampling: Given dataset $S = \{(x^{(i)}, y^{(i)}): i = 1, 2, ..., N\}$, create $S'$ by drawing $N$ examples at random with replacement from $S$

• Bagging:
  o Create $M$ bootstrap datasets $S_1, S_2, ..., S_M$
  o Train distinct models $f_m: \mathcal{X} \rightarrow \mathcal{Y}$ by training only on $S_m$
  o Output final predictor $F(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x)$ (for regression) or $F(x) = \text{majority}(f_m(x))$ (for classification)

Figure credit: David Sontag
Adaboost

Training data $S = \{(x^{(i)}, y^{(i)}): i = 1, 2, \ldots, N\}$

- Maintain weights $W_i^{(t)}$ for each example $(x^{(i)}, y^{(i)})$, initially all $W_i^{(1)} = \frac{1}{N}$
- For $t = 1, 2, \ldots, T$
  - Normalize weights $D_i^{(t)} = \frac{W_i^{(t)}}{\sum_i W_i^{(t)}}$
  - Pick a classifier $f_t$ has better than 0.5 weighted loss
    $\epsilon_t = \sum_{i=1}^{N} D_i^{(t)} \ell^{01}(f_t(x^{(i)}), y^{(i)})$
  - Set $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
  - Update weights
    $W_i^{(t+1)} = W_i^{(t)} \exp \left( -\alpha_t y^{(i)} f_t(x^{(i)}) \right)$

Example credit: Greg Shaknarovich
Adaboost

Training data $S = \{(x^{(i)}, y^{(i)}): i = 1, 2, \ldots, N\}$

- Maintain weights $W_i^{(t)}$ for each example $(x^{(i)}, y^{(i)})$, initially all $W_i^{(1)} = \frac{1}{N}$

- For $t = 1, 2, \ldots, T$
  - Normalize weights $D_i^{(t)} = \frac{w_i^{(t)}}{\sum_i w_i^{(t)}}$
  - Pick a classifier $f_t$ has better than 0.5 weighted loss
    $\epsilon_t = \sum_{i=1}^N D_i^{(t)} \ell^{01}(f_t(x^{(i)}), y^{(i)})$
  - Set $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
  - Update weights
    $W_i^{(t+1)} = W_i^{(t)} \exp \left( -\alpha_t y^{(i)} f_t(x^{(i)}) \right)$

- Output strong classifier $F_T(x) = \text{sign} (\sum_t \alpha_t f_t(x))$

Example credit: Greg Shaknarovich
Supervised learning summary

• Linear regression

• Classification
  o Logistic regression
  o Maximum margin classifiers, kernel trick
  o Generative models: Naïve Bayes, HMMs
  o Neural networks

• Ensemble methods

• Main concepts:
  o Detecting and avoiding overfitting and the tradeoff between bias and complexity
  o Learning parameters using empirical risk minimization (ERM) plus regularization
  o Optimization techniques: specially (stochastic) gradient descent for both convex and non-convex problems
Unsupervised learning

- **Unsupervised learning:** Requires data $x \in \mathcal{X}$, but no labels
- **Goal?** Compact representation of the data by detecting patterns
  - e.g. Group emails by topic
- **Useful when we don’t know what we are looking for**
  - makes evaluation tricky
- **Applications in visualization, exploratory data analysis, semi-supervised learning**

Figure credit: David Sontag
Linear dimensionality reduction

- **Problem:** Given high dimensional feature \( \mathbf{x} = [x_1, x_2, ..., x_d] \in \mathbb{R}^d \)
  find transformations \( \mathbf{z} = [z_1, z_2, ..., z_k] \in \mathbb{R}^k \)
  so that “almost all useful information” about \( \mathbf{x} \) is retained in \( \mathbf{z} \)
  - Learn \( \mathbf{z} \) from dataset of examples \( S = \{x^{(i)} \in \mathbb{R}^d : i = 1, 2, ..., N\} \)

- **Linear dimensionality reduction:** \( \mathbf{z} \) restricted to be a linear function
- **PCA:** given data \( \mathbf{x} \in \mathbb{R}^d \), find \( \mathbf{U} \in \mathbb{R}^{k \times d} \) to minimize
  \[
  \min_{\mathbf{U}} \sum_i \| \mathbf{U}^T \mathbf{x}^{(i)} - \mathbf{x}^{(i)} \|^2 \quad \text{s.t.} \quad \mathbf{U} \mathbf{U}^T = \mathbf{I}
  \]
  - solution given by eigenvalue decomposition of \( \hat{\Sigma}_{xx} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}^{(i)} \mathbf{x}^{(i)^T} \)
  - finds directions of maximum variation in data
  - check: make sure to center the data so that each feature has zero mean

- Can get non-linear embedding by doing PCA on \( \phi(\mathbf{x}) \) \( \rightarrow \) Kernel PCA
Non linear dimensionality reduction

- **Isomap**: Neighborhood of points represented using the kNN-graph with weights proportional to distance between the points
  - geodesic distance $d(x, x') = \text{length of shortest path in the graph}$
  - Use any shortest path algorithm can be used to construct a matrix $M \in \mathbb{R}^{N \times N}$ with $M_{ij} = d(x^{(i)}, x^{(j)})$ for all $x^{(i)}, x^{(j)} \in S$
  - **MDS**: Find a (low dimensional) embedding $z(x)$ of $x$ so that geodesic distance match the Euclidean distance in the transformed space
    \[
    \min_z \sum_{i,j \in [N]} (\|z(x^{(i)}) - z(x^{(j)})\| - M_{ij})^2
    \]
- Works well for small scale problems
Non linear dimensionality reduction

• **Autoencoders:**

  \[ \phi(x) = f_{W_1}(x) \]
  \[ \tilde{x} = f_{W_2}(\phi(x)) \]
  • some loss \( \ell(\tilde{x}, x) \)
  \[
  \hat{W}_1, \hat{W}_2 = \min_{W_1, W_2} \sum_{i=1}^{N} \ell \left( f_{W_2} \left( f_{W_1}(x^{(i)}) \right), x^{(i)} \right)
  \]
  • learn using SGD with backpropagation
MLE of latent variable models

• **Generative model:**
  - Observed variables \( x \in \mathcal{X} \)
  - Latent variables \( z \in \mathcal{Z} \)
  - Probabilistic generative model parameterized by parameters \( \Phi \) is
    \[
    P_\Phi(x, z) = P_\Phi(z)P_\Phi(x|z)
    \]
    - For each example \( x \), first sample \( z \sim P_\Phi(z) \), then sample \( x \sim P_\Phi(x|z) \)
    - Note: we never see \( z \), appears only in generative assumption
    - Latent variables allows for easier specification of \( \Pr(x) \)

• **MLE estimation:** given dataset \( S = \{x^{(i)}: i = 1, 2, \ldots, N\} \)
  \[
  \Phi^* = \arg\max_{\Phi} \sum_{i=1}^{N} \log \Pr(x^{(i)})
  \]
  \[
  \Phi^* = \arg\max_{\Phi} \sum_{i=1}^{N} \left( \log \sum_{z \in \mathcal{Z}} P_\Phi(x^{(i)}, z) \right)
  \]
Expectation Maximization high-level algo

\[ \Phi^* = \arg\max_\Phi \sum_{i=1}^{N} \left( \log \sum_{z^{(i)} \in Z} P_\Phi(x^{(i)}, z^{(i)}) \right) \]

• **Main idea:** Say we are looking at problems where the above optimization is “easy” if we “know” \( z^{(i)} \) but we don’t know \( z^{(i)} \).
  - Fix-alternate between estimating \( z^{(i)} \) and \( \Phi \)

• **Start with some estimate** \( \Phi^{(0)} \) of parameters we want to estimate:
  - **Expectation step (E-step):** Compute an expectation to “fill in” missing variables \( z^{(i)} \) assuming our current estimate of parameter \( \Phi^{(t)} \) is correct.
  - **Maximization step (M-step):** Assuming our estimates \( z^{(i)} \) from above E-step is correct, solve maximization to estimate \( \Phi^{(t+1)} \)
    - Recall that if we pretend to know \( z^{(i)} \), the optimization is “easy”

• **No magic!** still optimizing hard non-convex function with lots of local optima
  - not guaranteed to converge to global optima and
  - but often also give good enough solutions even if they are local optima
EM algorithm

\[ \Phi^* = \arg\max_{\Phi} \sum_{i=1}^{N} \left( \log \sum_{z^{(i)} \in \mathcal{Z}} P_{\Phi}(x^{(i)}, z^{(i)}) \right) \]

- **Expectation step (E-step):** “fill in” missing variables \(z^{(i)}\) assuming our current estimate of \(\Phi^{(t)}\) is correct.

- **How to do this?**
  - Specify an auxiliary model \(P_{\Psi}(z|x)\)
  - Instead of filling in one value of \(z\) this gives a distribution over \(z|x\)
  - Idea: find a way to estimate \(\Psi\) under this model! If the model is correct, we in turn get a good estimate of \(z\)

\[
ELBO_x(\Phi, \Psi) = \mathbb{E}_{z \sim P_{\Psi}(\cdot|x)} \log P_{\Phi}(x|z) + D_{KL}(P_{\Psi}(z|x) || P_{\Phi}(z))
\]

- For any \(\Psi\), \(ELBO_x(\Phi, \Psi) \leq \log(\sum_{z \in \mathcal{Z}} P_{\Phi}(x, z))\) and maximized when \(P_{\Psi}(z|x) = P_{\Phi}(z) = \sum_{x \in \mathcal{X}} P_{\Phi}(z, x)\)
EM algorithm

$$\Phi^* = \operatorname*{argmax}_\Phi \sum_{i=1}^{N} \left( \log \sum_{z^{(i)} \in \mathcal{Z}} P_\Phi(x^{(i)}, z^{(i)}) \right)$$

- Specify joint models $P_\Phi(z, x)$ and auxiliary model $P_\Psi(z|x)$
- Initialize $\Phi^{(0)}, \Psi^{(0)}$
- For $t = 1, 2, \ldots$,
  - $\Psi^{(t)} = \max_\Psi \text{ELBO}(\Phi^{(t-1)}, \Psi)$
  - $\Phi^{(t)} = \max_\Phi \text{ELBO}(\Phi, \Psi^{(t)})$
Unsupervised learning – clustering

• **k-means clustering**
  - hard clustering
  - Initialize cluster centroid
  - Alternatingly
    - Compute cluster memberships (hard memberships)
    - Update cluster centroids

• **Gaussian mixture models**
  - soft clustering: cluster membership is a probability vector $\pi \in \Delta^{k-1}$ over $k$ mixture components and mixture components are Gaussians with means $\mu_1, \mu_2, \ldots, \mu_k$
  - EM algorithm alternatingly:
    - Computes soft cluster memberships $\pi^{(t)}$
    - Updates mixture component means $\mu_1^{(t)}, \mu_2^{(t)}, \ldots, \mu_k^{(t)}$

• Main modeling in specifying distance or learning representation
Topics not covered
Semi-Supervised Learning
Using unlabeled data to help predictions

Slide credit: Nati Srebro
Active Learning

• Training data is randomly drawn/fixed
• What if we could explicitly ask for specific training data?
  o E.g. we could query an expert (a teacher, a user, someone on mechanical turk) about a specific point
• Setting
  o We have a large collection of unlabeled points
  o Can query labels for specific unlabeled examples
  o Each query has a cost associated, so we want to minimize the number of queries
  o Goal is to still learn a mapping from input to some label/output
• How to design the querying system so that we learn good models with smallest amount of data?
Limited/partial Feedback

• Instead of getting correct label, we only know if the prediction was correct or not
• Only know loss/payoff of label/action chosen, not of others
• “Bandit” problems: ad placement, recommendation systems, ...

• New challenge: Exploration vs Exploitation
Reinforcement Learning

• Control agent (robot) in environment, only see reward when you get it

• Long term planning to finish a task
  o At time $t$ you are in some (unknown to you) state $s_t$
  o You choose an action $a_t$, based on which you move to a new state $s_{t+1} = f(s_t, a_t)$ (maybe with some randomness) and receive reward $r(s_{t+1})$.
  o You don’t know $f(\cdot, \cdot)$ and $r(\cdot)$ (need to learn them)
  o You only know the rewards $r(s_t)$ you get, and possibly other limited feedback about the state $o(s_t)$
  o Goal: maximize rewards

• E.g.: mouse moving in a maze
  o State = location and direction
  o Action = move forward, turn left or turn right
  o Reward = cheese
  o Observation(State) = (front wall, left wall, right wall, back wall)

Slide credit: Nati Srebro
Probabilistic Models

• Probabilistic models define models for $\Pr(x, y)$ or $\Pr(x|y)$ or $\Pr(y|x)$

• We saw some simple examples of this flavor

• More complex models often use many latent variables
  o typically represented as using graphical models such as Bayesian Networks and Markov Random Fields

• Techniques for
  o Modeling: how to represent $\Pr(x, y)$ or $\Pr(x|y)$ or $\Pr(y|x)$
  o Inference: inferring the values of latent variables
  o Learning: prediction

• Many times the optimization problems are non-convex and sometimes even non-computable
  o approximate inference algorithms are very common
Machine Learning Landscape

**Convex (= Linear)**
- Linear/logistic reg.
- SVMs
- Boosting
- Many other models

Main optimization tools: LP/SDP solvers and SGD

**Non-Convex**
- **Neural Networks**
- Dictionary and representation learning

Main optimization tools: SGD with tricks

**Combinatorial Classes**
- Formulas (DNFs)
- **Decision trees**

Main optimization tools: greedy, combinatorial search (using pruning, genetic programming, simulated annealing, etc)

**Non-Parametric**
- **Nearest-Neighbor**
- Parzan Window
- Random walk on example graph

**Probabilistic Models**
- Fit data to generative model
- Bayes nets, graphical models
- Latent variable models

Typically non-convex, same issues as non-convex models

Slide credit: Nati Srebro
Expert designed → data driven

Expert designed systems

machine learning

Just dump all data into the machine

C. M. Bishop: “...a training set is used to tune the parameters of an adaptive model”