MLSS 2014 – Introduction to Machine Learning Lecture 2

J. Zico Kolter

July 8, 2014

Outline

Other machine learning algorithms

Unsupervised learning

Probabilistic models

Evaluating machine learning algorithms

Outline

Other machine learning algorithms

Unsupervised learning

Probabilistic models

Evaluating machine learning algorithms

Kernel methods

- Kernel methods are a very popular approach to non-linear classification, though they are still "linear" in some sense

$$h_\theta(x) = \sum_{i=1}^{m} \theta_i K(x, x^{(i)})$$

  where $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a *kernel function* that measures the similarity between x and $x^{(i)}$ (larger values for more similar)

- For certain K, can be interpreted as working in a high dimensional feature space without explicitly forming features

- Still linear in θ, can use many of the same algorithms as before

- **Important**: $\theta \in \mathbb{R}^m$, as many parameters as examples (*non-parametric* approach)

Nearest neighbor methods

- Predict output based upon closest example in training set

$$h_\theta(x) = y^{(\text{argmin}_i \, \|x - x^{(i)}\|^2)}$$

  where $\|x\|^2 = \sum_{i=1}^{n} x_i^2$

- Can also average over k closest examples: k-nearest neighbor

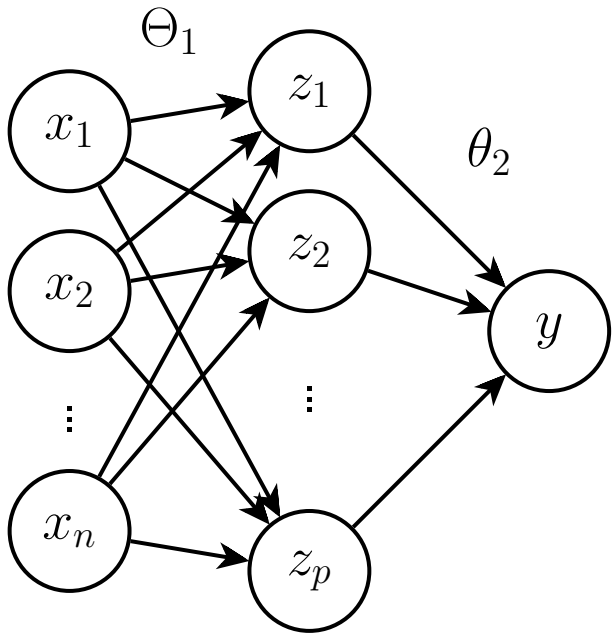
- Requires no separate "training" phase, but (like kernel methods) it is non-parametric, requires that we keep around all the data

Neural networks

- Non-linear hypothesis class

$$h_\theta(x) = \sigma(\theta_2^T \sigma(\Theta_1^T x))$$

for a 2-layer network, where $\theta = \{\Theta_1 \in \mathbb{R}^{n \times p}, \theta_2 \in \mathbb{R}^p$ and $\sigma : \mathbb{R} \to \mathbb{R}$ is an sigmoid function $\sigma(z) = 1/(1 + \exp(-z))$ (applied elementwise to vector)

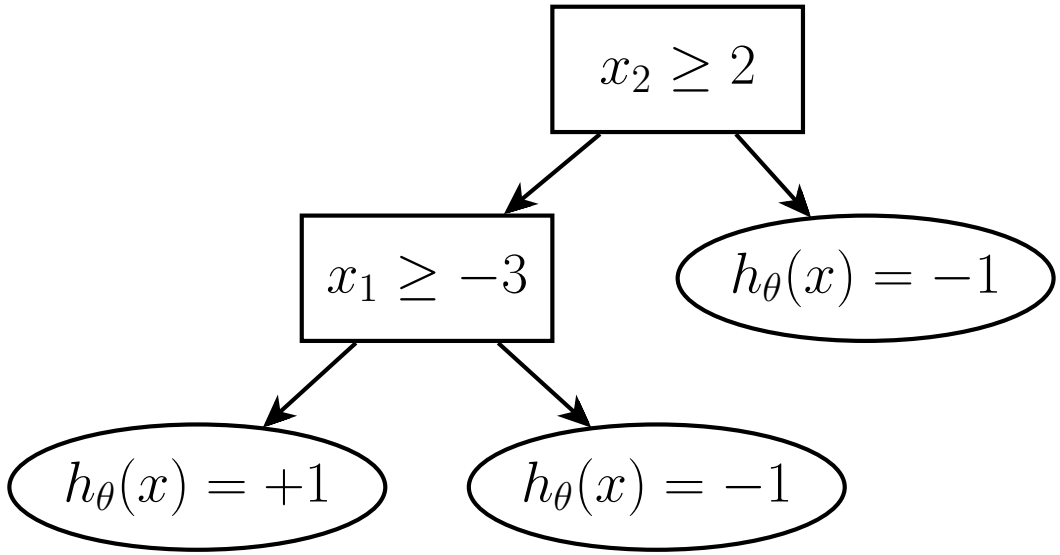


- Non-convex optimization, but smooth (gradient and similar methods can work very well)

- Some major recent success stories in speech recognition, image classification

Decision trees

- Hypothesis class partitions space into different regions



- Can also have linear predictors (regression or classification) at the leaves

- Greedy training find nodes that best separate data into distinct classes

Ensemble methods

- Combine a number of different hypotheses

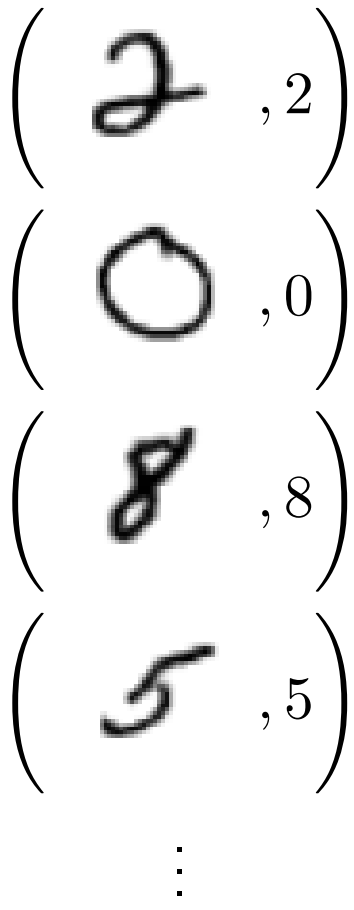
$$h_\theta(x) = \sum_{i=1}^{k} \theta_i \text{sign}(h_i(x))$$

- Popular instances
  - Random forests: ensemble of decision trees built from different subsets of training data
  - Boosting: iteratively train multiple classifiers/regressors on reweighted examples based upon performance of the previous hypothesis

Outline

Supervised learning

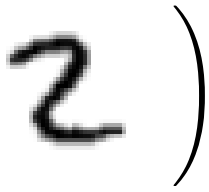
Training Data

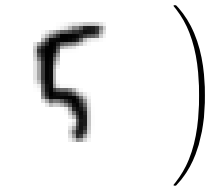


Machine Learning

$\longrightarrow$ Hypothesis function h_θ

Deployment

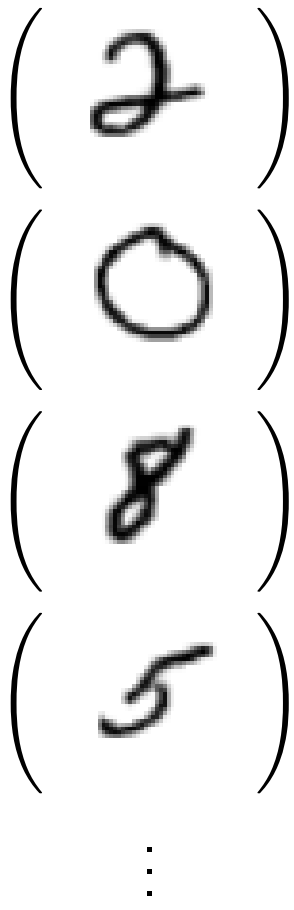
Prediction $= h_\theta$ 

Prediction $= h_\theta$ 

$\vdots$

Unsupervised learning

Training Data

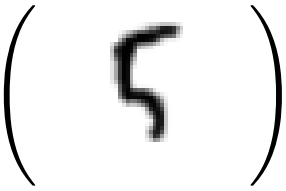


Machine Learning

$\longrightarrow$ Hypothesis function h_θ

Deployment

Prediction $= h_\theta$ 

Prediction $= h_\theta$ 

$\vdots$

Problem setting

- **Input features**: $x^{(i)} \in \mathbb{R}^n, \quad i = 1, \ldots, m$
- **Model parameters**: $\theta \in \mathbb{R}^k$
- How do we specify a hypothesis class or loss function without outputs?

- One way to interpret many unsupervised learning algorithms is that they try to "re-create" the input using a limited hypothesis class

- **Hypothesis function**: $h_\theta : \mathbb{R}^n \to \mathbb{R}^n$
  - Want $h_\theta(x^{(i)}) \approx x^{(i)}$ for all training data

- **Loss function**: $\ell : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$
  - E.g., $\ell(h_\theta(x), x) = \|h_\theta(x) - x\|^2$

- In order to prevent the trivial solution $h_\theta(x) = x$, we need to restrict the class of allowable functions h_θ

k-means

- Parameters are a set of k "centers" in the data

$$\theta = \{\mu^{(1)}, \ldots, \mu^{(k)}\}, \ \mu^{(i)} \in \mathbb{R}^k$$

- Hypothesis class picks the closest center

$$h_\theta(x) = \mu^{(\text{argmin}_i \, \|x - \mu^{(i)}\|^2)}$$

- With this framework, training looks the same as supervised learning

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \|x^{(i)} - h_\theta(x^{(i)})\|^2$$

- Not a convex problem, but can solve by iteratively finding the closest $\mu^{(i)}$ for each example, then setting $\mu^{(i)}$ to be the mean of all examples assigned to it

Principal component analysis

- Parameters are two matrices that reduce the effective dimension of the data, $\theta = \{\Theta_1 \in \mathbb{R}^{n \times k}, \Theta_2 \in \mathbb{R}^{k \times n}\}$ with $k < n$

- Hypothesis class $h_\theta(x) = \Theta_1 \Theta_2 x$

- Interpretation: to reconstruct data $\Theta_2 x \in \mathbb{R}^k$ needs to preserve most of the information in x, so that we can construct it (dimensionality reduction)

- Minimizing loss

$$\underset{\Theta_1, \Theta_2}{\text{minimize}} \sum_{i=1}^{m} \|x^{(i)} - \Theta_1 \Theta_2 x^{(i)}\|_2^2$$

  is not a convex problem, but can be solved (exactly) via an eigenvalue decomposition

Outline

Other machine learning algorithms

Unsupervised learning

Probabilistic models

Evaluating machine learning algorithms

Probability in machine learning

- Probabilistic models lie behind many of the algorithms in machine learning

- Probabilistic models can make the predictions of many algorithms more interpretable in terms of the underlying uncertainty, and dictate certain choices of loss functions

- An example: why did we choose the squared loss function in the previous lecture on linear regression?

$$\ell(h_\theta(x), y) = (h_\theta(x) - y)^2$$

Squared loss and Gaussian likelihood

- Suppose that the each output y in our data really *is* equal to the hypothesis function for that example $h_\theta(x)$, just corrupted by Gaussian noise ϵ

$$y = h_\theta(x) + \epsilon$$

- The probability density of a Gaussian variable given by

$$p(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right)$$

- Subsituting terms, we can use this expression to write the probability of y *given* x (parameterized by θ)

$$p(y|x;\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(h_\theta(x) - y)^2}{2\sigma^2}\right)$$

- Consider the joint probability of *all* training data (assuming samples are independent and identically distributed)

$$p(y^{(1)}, \ldots, y^{(m)} | x^{(1)}, \ldots, x^{(m)}; \theta) = \prod_{i=1}^{m} p(y^{(i)} | x^{(i)}; \theta)$$

- Find the parameters that θ maximize the probability of the data

$$\underset{\theta}{\text{maximize}} \prod_{i=1}^{m} p(y^{(i)} | x^{(i)}; \theta) \quad \equiv \quad \underset{\theta}{\text{minimize}} \; -\sum_{i=1}^{m} \log p(y^{(i)} | x^{(i)}; \theta)$$

$$\equiv \underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \left(\log(\sqrt{2\pi}\sigma) + \frac{1}{2\sigma^2} (h_\theta(x^{(i)}) - y^{(i)})^2 \right)$$

$$\equiv \underset{\theta}{\text{minimize}} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

- This is a procedure known as *maximum likelihood estimation*, a common statistical technique

- Note that we still just pushed the question of "which loss" to "which distribution"

  - But some distributions, like Gaussian, may have reasonable empirical or theoretical justifications for certain problems

Logistic regression

- Another example: for binary classification problem, suppose that

$$p(y|x;\theta) = \frac{1}{1+\exp(-y \cdot h_\theta(x))}$$

and for each data point $x^{(i)}$, $y^{(i)}$ is sampled randomly from this distribution

- Then

$$\begin{aligned}
&\underset{\theta}{\text{minimize}} \; -\sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)};\theta) \\
&\equiv \underset{\theta}{\text{minimize}} \; \log\left(1+\exp\left(-y^{(i)} \cdot h_\theta(x^{(i)})\right)\right)
\end{aligned}$$

which was exactly our logistic loss function

- In both cases, probabilistic models gives both a loss function and a way to interpret predictions as measure of *uncertainty*

Outline

Other machine learning algorithms

Unsupervised learning

Probabilistic models

Evaluating machine learning algorithms

Evaluating ML algorithms

- You have developed a machine learning approach to a certain task, and want to validate that it actually works well (or determine if it doesn't work well)

- Standard: approach, divide data into training and testing sets, train method on training set, and report results on the testing set

- Important: testing set is *not* the same as the validation test

- The proper way to evaluate an ML algorithm
  1. Break all data into training/testing sets (e.g., 70%/30%)
  2. Break training set into training/validation set (e.g., 70%/30% again)
  3. Choose hyperparameters using validation set
  4. (Optional) Once we have selected hyperparameters, retrain using all the training set
  5. Evaluate performance on the testing set