2. Machine Learning

Background

CS 519: Deep Learning, Winter 2016

Object Classification Example

• We want to learn what is in the image
• Input:
  • Lots and lots of examples with **data** and their **labels**
Object Classification Example

- We want to learn what is in the image

Input:
- Lots and lots of examples with data and their label
- We transformed the data into features (usually vectors) of the data
Data: Classification

• Input Data:

\[ D = \{ \mathbf{X}, Y \} \]

\[ \mathbf{X} = \{ x_1, x_2, \ldots, x_n \} \text{ examples and } x_i \in \mathbb{R}^d \]

\[ Y = \{ y_1, y_2, \ldots, y_n \} \text{ are labels } 0, \ldots, k \]
Model: Discriminative Parameterized Function

\[ f(x; w) \approx y \]

- Classifier
- Input (vector)
- Parameters
- Output (scalar or vector)

• Typically output \( y \) is a score
**Model:** Object Classification Example

Key notion of discriminative models: **Separability**
Linear classifiers:

\[ y = mx + b \]

- Our model is: \( f(x_i, w, b) = w^T x_i + b \)

Usually refer \([w, b]\) as \(w\)
Linear Classifiers
What does this classifier do?

• Scores input based on linear combination of features
  • > 0 above hyperplane
  • < 0 below hyperplane

• Changes in weight vector (per classifier)
  • Rotate hyperplane

• Changes in Bias
  • Offset hyperplane from origin
Optimization of parameters

• Want to find \( \mathbf{w} \) that achieves best result

• Empirical Risk Minimization principle
  • Find \( \mathbf{w} \) that

\[
\min_{\mathbf{w}} \sum_{i=1}^{n} L(y_i, f(x_i; \mathbf{w}))
\]

• Real goal:
  • Find \( \mathbf{w} \) that

\[
\min_{\mathbf{w}} \mathbb{E}[L_{c}(y_i, f(x_i; \mathbf{w}))]
\]

\[L_{c}:
\begin{cases}
1, y \neq f(x) \\
0, y = f(x)
\end{cases}
\]

• Bayes error: Theoretically optimal error
Loss Function: Some examples

• Binary: \( y \in \{-1, 1\} \)
• L1/L2
\[
L_i = |y - w^T x_i |
\]
\[
L_i = \frac{1}{2} (y - w^T x_i)^2
\]
• Logistic
\[
L_i = log(1 + e^{yw^T x_i})
\]
• Hinge (SVM)
\[
L_i = max(0, 1 - yw^T x_i)
\]
• Lots more
  • e.g. treat “most offending incorrect answer” in a special way
Vanilla (Binary) Linear SVMs

- **Model:** $w^T x + b$
- **Loss:** Hinge loss
- **Objective:** Maximize margin subject to correct classification

\[
\frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i(w \cdot x_i + b)) + \lambda \|w\|^2.
\]
Is linear sufficient?

• Very typical model is a set of hyperplanes to separate classes

• Many interesting functions (as well as some non-interesting functions) not linearly separable

The probability that $P$ samples of dimension $N$ are linearly separable goes to zero very quickly as $P$ grows larger than $N$ (Cover’s theorem, 1966).

Some seemingly simple dichotomies are not linearly separable

Question: How do we make a given problem linearly separable?
**Model:** Expansion of Dimensionality

- **Representations:**
  - Simple idea: Quadratic expansion
    \[ [x_1, x_2, ..., x_d] \mapsto [x_1^2, x_2^2, ..., x_d^2, x_1x_2, x_1x_3, ..., x_{d-1}x_d] \]
  - A better idea: Kernels
    \[ K(x, x_i) = \exp(-\beta ||x_i - x||^2) \quad f(x) = \sum_i \alpha_i K(x, x_i) \]
  - Another idea: Fourier domain representations (Rahimi and Recht 2007)
    \[ \cos(w^T x + b), w \sim N^d(0, \beta I), b \sim U[0,1] \]
  - Another idea: Sigmoids (early neural networks)
    \[ \text{sigmoid}(w^T x + b), \text{optimized } w \]
Distance-based Learners (kNN)
Distance-based Learners (Gaussian SVM)

SVM: Linear

SVM - Radial Kernel in Feature Space

Training Error: 0.270
Test Error: 0.288
Bayes Error: 0.210

Training Error: 0.160
Test Error: 0.216
Bayes Error: 0.210
Machine Learning Theory

Theoretic intuitions that are important to know
Machine Learning Theory: Basic setup

- Generic supervised learning setup:
- For $(x_i, y_i)_{1 \ldots n}$ i.i.d. drawn from the joint distribution $P(x, y)$, find a best function $f \in F$ that minimizes the error $E_{x,y}[L(f(x), y)]$
  - $L$ is a loss function, e.g.
  - Classification:
    $$L(f(x), y) = \begin{cases} 1, & f(x) \neq y \\ 0, & f(x) = y \end{cases}$$
  - Regression: $L(f(x), y) = (f(x) - y)^2$
  - $F$ is a function class (consists many functions, e.g. all linear functions, all quadratic functions, all smooth functions, etc.)
Machine Learning Theory: Generalization

• Machine learning theory is about **generalizing** to unseen examples
  • **Not** the training set error!
  • And those theory **doesn’t always** hold (holds with probability less than 1)

• A generic machine learning generalization bound:
  • For \((x_i, y_i)_{1...n}\) drawn from the joint distribution \(P(x, y)\), with probability \(1 - \delta\)
    \[
    E_{x,y}(f(x) \neq y) \leq \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) + \Omega(F, \delta)
    \]
    - Error on the whole distribution
    - Error on the training set
    - Flexibility of the function class

How to represent “flexibility”? That’s a course on ML theory
What is “flexibility”? 

• Roughly, the more functions in $F$, the more flexible it is 

• Function class: all linear functions $F: \{f(x) | f(x) = w^T x + b\}$ 
  • Not very flexible, cannot even solve XOR 
  • Small “flexibility” term, testing error not much more than training error 

• Function class: all 9-th degree polynomials $F: \{f(x) | f(x) = w_1^T x^9 + \cdots \}$ 
  • Super flexible 
  • Big “flexibility” term, testing error can be much more than training
Flexibility and overfitting

• For a very flexible function class
  • Training error is **NOT** a good measure of testing error
  • Therefore, out-of-sample error estimates are needed
    • Separate validation set to measure the error
    • Cross-validation
      • K-fold
      • Leave-one-out
    • Many times this will show to be worse than the training error with a flexible function class
Another twist of the generalization inequality

• Nevertheless, you still want training error to be **small**
• So you don’t always want to use linear classifiers/regressors

\[
E_{x,y}(f(x) \neq y) \leq \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) + \Omega(F, \delta)
\]

- Error on the whole distribution
- Error on the training set
- Flexibility of the function class

If this is 60% error... Add-on term
How to deal with it when you do use a flexible function class

• Regularization
  • To make the chance of choosing a highly flexible function to be low
  • Example:
    • Ridge Regression:
      \[
      \min_w (w^T X - Y)^2 + \lambda ||w||^2
      \]
      In order to choose a \( w \) with big \( ||w||^2 \),
      you need to overcome this term

    • Kernel SVM
      \[
      \min_f \sum_i L(f(x_i), y_i) + \lambda ||f||^2
      \]
      In order to choose a very unsmooth function \( f \)
      you need to overcome this term
Bayesian Interpretation of Regularization

• Assume that a certain prior of the parameters exist, and optimize for the MAP estimate
  • Example:
    • Ridge Regression: Gaussian prior on $w$: $P(w) = C \exp(-\lambda ||w||^2)$
      $$ \min_w (w^T X - Y)^2 + \lambda ||w||^2 $$
    • Kernel SVM: Gaussian process prior on $f$ (too complicated to explain simply..)
      $$ \min_f \sum_i L(f(x_i), y_i) + \lambda ||f||^2 $$
Universal Approximators

- Universal Approximators
  - (Barron 1994, Bartlett et al. 1999) Meaning that they can approximate (learn) any smooth function efficiently (meaning using a polynomial number of hidden units)
  - Kernel SVM
  - Neural Networks
  - Boosted Decision Trees

- Machine learning cannot do much better
  - No free lunch theorem
No Free Lunch

• (Wolpert 1996, Wolpert 2001) For any 2 learning algorithms, averaged over any training set d and over all possible distributions P, their average error is the same

• Practical machine learning only works because of certain correct assumptions about the data
  • SVM succeeds by successfully representing the general smoothness assumption as a convex optimization problem (with global optimum)
  • However, if one goes for more complex assumptions, convexity is very hard to achieve!
The Most Basic Optimization Concepts

- Convex = Good, great, fantastic, global optimality

- Non-convex = I don’t know what’s going on!
Deep Learning/Neural Networks

• March into the nonconvexity swamp
  • A step hard to take by mathematically-minded people
    • But necessary given that our comfort zone is this small
  • Difficulties ahead
    • Your algorithm may not work off-the-shelf
    • You need intuition
    • You need to spend time by trying many things
    • Sometimes it will work better!
    • But there is no formula for success
    • Creativity and hard-work both valued
High-dimensionality

Philosophical discussion about high-dimensional spaces
Distance-based Algorithms

• K-Nearest Neighbors: weighted average of k-nearest neighbors
Curse of Dimensionality

- Dimensionality brings interesting effects:
- In a 10-dim space, to cover 10% of the data in a unit cube, one needs a box to cover 80% of the range.

**FIGURE 2.6.** The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction $r$ of the volume of the data, for different dimensions $p$. In ten dimensions we need to cover 80% of the range of each coordinate to capture 10% of the data.
High Dimensionality Facts

• Every point is on the boundary
  • With N uniformly distributed points in a p-dimensional ball, the closest point to the origin has a distance of
    \[ d(p, N) = \left(1 - \frac{1}{2^{1/N}}\right)^{1/p} \]

• Every vector is almost always orthogonal to each other
  • Pick 2 unit vectors \( x_1 \) and \( x_2 \), then the probability that
    \[ \cos(x_1, x_2) = |x_1^\top x_2| \geq \sqrt{\frac{\log p}{p}} \]
    is less than \( 1/p \)
Avoiding the Curse

• Regularization helps us with the curse
  • Smoothness constraints also grow stronger with the dimensionality!

\[ \int |f'(x)| dx \leq C \]

\[ \int \left| \frac{\partial f}{\partial x_1} \right| dx_1 + \int \left| \frac{\partial f}{\partial x_2} \right| dx_2 + \cdots + \int \left| \frac{\partial f}{\partial x_p} \right| dx_p \leq C \]

• We do not suffer from the curse if we ONLY estimate sufficiently smooth functions!
References


