Machine Learning (CSE 446): Multi-Class Classification; Kernel Methods

Sham M Kakade

University of Washington
`cse446-staff@cs.washington.edu`

Announcements

- HW3 due date as posted.
  - make sure to update the HW pdf file today for clarifications: always use *average* squared error; always report your lowest test losses; show all curves (train/dev/test) together.
  - check canvas for updates/announcements.
- Extra Credit will be due Monday, Feb 26th
  - You must do the *all* of HW3 if you seek any extra credit.
- Office Hours: Tue, the 20th, at ~~3-4p~~

  update:

  2-3 pm
- Today:
  - Multi-class classification
  - non-linearities; kernel methods

Review/Comments

Some questions on Probabilistic Estimation

coin

- Remember: think about taking derivatives 'from scratch' (pretend like you no one told you about vector/matrix derivatives)
- Can you rework that HTHHH example by using the log loss to estimate the bias, π? Do you see why it gives the same result?
- Suppose you see 51 Heads and 38 Tails. Do you see why it helpful to consider maximizing the log probability rather than directly trying to maximize the probability? Try working this example out with both methods!
- Can you formulate this problem of estimating π as a logistic regression problem, i.e. without x? We would just use a bias term where $\Pr(y = 1) = \frac{1}{1+\exp(-b)}$.
- your learned this in 312 ? do you remember what you did?

Some questions on Training and Optimization

- Do you understand that our optimization is *not* directly minimizing our number of mistakes? And how what we are doing is different from the perceptron? You will be making two sets of plots in all these HWs.
- Do you see what we are minimizing? Try to gain intuition as to how the parameters adapt based on the underlying error.
- Do you see what you do minimize the $0/1$ loss for the example $HTHHH$?
- Do you understand what a (trivial) lower bound on the square loss is? on the log loss? Do you understand the general conditions when this is achievable?
  - Problem 2.3 forces you to think about these issues.
    Do you see what loss you will converge to in problem 2.3?

MNIST: comments

- understanding the MNIST table of results
  tricks to get lower performance:
  - "make your dataset bigger": pixel jitter, distortions, deskewing. These lower the error for pretty much any algorithm.
  - convolutional methods:
- there is no dev set for Q5??

  1.2 – 1.4 %

- two class problem is clearly easier than 10 class problem
  all datasets are from the same dataset!

Today

Multi-class classification

- suppose $y \in \{1, 2, \ldots k\}$.
- MNIST: we have $k = 10$ classes. How do we learn?
- Misclassification error:
  the fraction of times (often measured in %) in which our prediction of the label does not agree with the true label.
- Like binary classification, we do not optimize this directly
  it is often computationally difficult

misclassification error: one perspective...

- misclassification error is a terrible objective function to use anyways:
  - it only gives feedback of “correct” or “not”
  - even if you don’t predict the true label (e.g. you make a mistake), there is a major difference between your model still “thinking” the true label is likely v.s. thinking the true label is “very unlikely”.
- how do give our model better ’feedback’?
  - Our must provide probabilities of **all** outcomes
  - Then we reward/penalize our model based on its “confidence” of the correct answer...

Multi-class classification: “one vs all”

$y \in \{1, \dots 10\}$

- Simplest method: consider each class separately.
- make 10 binary prediction problems: $y^{(1)} \in \{0,1\}$ $y^{(2)} \in \{0,1\}$ $\dots$ $y^{(10)} \in \{0,1\}$
- Build a separate model of $\Pr(y^{\text{class}}) = 1 | x, \mathbf{w}^{\text{class}})$.
- Example (just like in HW Q1): build $k = 10$ separate regression models.

model

$\hat{y}^{(1)} \approx w^{(1)} \cdot x \quad \dots \quad \hat{y}^{(10)} = w^{(10)} \cdot x$

A better probabilistic model: the soft max

$w^{(2)} \cdot x$

$Pr(y = \text{class } 2) \propto e$

- $y \in \{1, \ldots k\}$: Let's turn the probabilistic crank....
- The model: we have k weight vectors, $w^{(1)}, w^{(2)}, \ldots w^{(k)}$. For $\ell \in \{1, \ldots k\}$,

$$p(y = \ell | x, w^{(1)}, w^{(2)}, \ldots w^{(k)}) = \frac{\exp(w^{(\ell)} \cdot x)}{\sum_{i=1}^{k} \exp(w^{(i)} \cdot x)}$$

- ~~It is "over-parameterized":~~

$Pr(y = 1 | x) = \frac{1}{1 + e^{-\omega x}}$

$$p_W(y = k | x) = 1 - \sum_{i=1}^{k-1} p_W(y = i | x)$$

$Pr(y = 0 | x) = 1 - \ldots$

- max. likelihood estimation is still a convex problem!

Aside: why might square loss be 'ok' for binary classification?

$E[(y-f(x))^2]$

- Using the square loss for $y \in \{0,1\}$?
  - it doesn't look like a great surrogate loss.
  - also, it doesn't look like a faithful probabilistic model:

$y = w \cdot x + \varepsilon \qquad \varepsilon \sim N(0, \sigma^2)$

- What is the "Bayes optimal" predictor for the square loss?

$\rightarrow$ best you can do is $E[y|x]$

- The Bayes optimal predictor for the square loss with $y \in \{0,1\}$:

$\Pr(y=1|x)$

- Can we utilize something more non-linear in our regression?

$\hat{y} = w \cdot x \longrightarrow \hat{y} = w \cdot \phi(x)$

Can We Have Nonlinearity *and* Convexity?

	expressiveness	convexity
Linear classifiers	☹	☺
Neural networks	☺	☹

Can We Have Nonlinearity *and* Convexity?

	expressiveness	convexity
Linear classifiers	☹	☺
Neural networks	☺	☹

Kernel methods: a family of approaches that give us nonlinear decision boundaries without giving up convexity.

Let's try to build feature mappings

- Let $\phi(x)$ be a mapping from d-dimensional x to $\tilde{d}$-dimensional x.
- 2-dimensional example: quadratic interactions

- What do we call these quadratic terms for binary inputs?

Another example

- 2-dimensional example: bias$+$linear$+$quadratics interactions
- What do we call these quadratic terms for binary inputs?

The Kernel Trick

- Some learning algorithms, like the (lin. or logistic) regression, only need you to specify a way to take *inner products* between your feature vectors.
- A **kernel** function (implicitly) computes this inner product:

  $$K(\mathbf{x}, \mathbf{v}) = \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{v})$$

  for some ϕ. Typically it is *cheap* to compute $K(\cdot, \cdot)$, and we never explicitly represent $\phi(\mathbf{v})$ for any vector $\mathbf{v}$.
- Let's see!

Examples...