CS229 binary classification and general loss function notes

James Chuang

April 7, 2017

Contents

My notes on John Duchi's CS229 binary classification and general loss function supplemental notes.

binary classification

- **binary classification**
  - target y can take on only two values
    - represent by $y \in \{-1, +1\}$
    - assume input features $x \in \mathbb{R}^n$
- use standard approach to supervised learning:
  1. pick a representation for the hypothesis class
  2. pick a loss function to minimize
  - in binary classification, often use hypothesis of the form:

$$h_\theta(x) = \theta^T x$$

  - then, classify based on the sign of $\theta^T x$, i.e. sign$(\theta^T x)$
    - an example (x, y) is classified correctly if:

$$\text{sign}(h_\theta(x)) = y$$

    - or equivalently, if:

$$y\theta^T x > 0$$

    - $y\theta^T x$ is called the **margin** for the example (x, y)
    - often (not always), $h_\theta(x) = x^T\theta$ is interpreted as a measure of the confidence with which the parameter vector θ assigns a label for the point x
      - $x^T\theta$ very negative (positive), then we more strongly believe that the label y is negative (positive)
  - having chosen a hypothesis class, now choose a loss function
    - intuitively, want a loss function which:
      - given training data $\left\{x^{(i)}, y^{(i)}\right\}_{i=1}^m$, the chosen θ makes the margin $y^{(i)}\theta^T x^{(i)}$ very large for each training example
    - fix a hypothetical example (x, y), and let:
      - $z = yx^T\theta$ denote the margin
      - $\varphi : \mathbb{R} \to \mathbb{R}$ be the loss function
    - for a particular loss function, the empirical risk to minimize is then:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \varphi\left(y^{(i)} \theta^T x^{(i)}\right)$$

- desired behavior:
  - want $y^{(i)} \theta^T x^{(i)}$ positive for each training example $i = 1, \ldots, m$
  - should penalize θ for which $y^{(i)} \theta^T x^{(i)} < 0$ frequently in the training data
- an intuitive choice for loss function:
  - $\varphi(z)$ small if $z > 0$ (margin is positive)
  - $\varphi(z)$ large if $z < 0$ (margin is negative)
- a natural choice is then **zero-one loss**:

$$\varphi_{\text{zo}}(z) = \begin{cases} 1 & \text{if } z \leq 0 \\ 0 & \text{if } z > 0 \end{cases}$$

- with zero-one loss, the risk $J(\theta)$ is the average number of misclassifications that the parameter θ makes on the training data
- negatives:
  - zero-one loss is discontinuous, non-convex, NP-hard to minimize
    - therefore, prefer losses which satisfy:

$$\begin{cases} \varphi(z) \to 0 & \text{as } z \to \infty \\ \varphi(z) \to \infty & \text{as } z \to -\infty \end{cases}$$

- three loss functions commonly used in ML:
  - **logistic loss**

$$\varphi_{\text{logistic}}(z) = \log\left(1 + e^{-z}\right)$$

  - **hinge (SVM) loss**

$$\begin{aligned} \varphi_{\text{hinge}}(z) &= [1 - z]_+ \\ &= \max\{1 - z, 0\} \end{aligned}$$

  - **exponential loss**

$$\varphi_{\text{exp}}(z) = e^{-z}$$

- minimizing different loss functions leads to different ML algorithms:
  - logistic loss $\to$ logistic regression
  - hinge loss $\to$ support vector machines
  - exponential loss $\to$ boosting

logistic regression

- use binary labels $y \in \{-1, 1\}$
- use logistic loss:

$$\varphi_{\text{logistic}}(yx^T\theta) = \log\left(1 + \exp\left(-yx^T\theta\right)\right)$$

- logistic regression corresponds to choosing θ to minimize the empirical risk:

$$\begin{aligned} J(\theta) &= \frac{1}{m} \sum_{i=1}^{m} \varphi_{\text{logistic}}\left(y^{(i)} \theta^T x^{(i)}\right) \\ &= \frac{1}{m} \sum_{i=1}^{m} \log\left(1 + \exp\left(-y^{(i)} \theta^T x^{(i)}\right)\right) \end{aligned}$$

- **probabilistic interpretation:**
  - define **sigmoid**, aka **logistic** function:

$$g(z) = \frac{1}{1 + e^{-z}}$$

  - the sigmoid function satisfies

$$g(z) + g(-z) = \frac{1}{1 + e^{-z}} + \frac{1}{1 + e^{z}} = \frac{e^{z}}{1 + e^{z}} + \frac{1}{1 + e^{z}} = 1$$

  - therefore, the sigmoid function can be used to define a probability model for binary classification
  - for $y \in \{-1, 1\}$, define the **logistic model** for classification:

$$p\left(Y = y \mid x; \theta\right) = g\left(yx^{T}\theta\right) = \frac{1}{1 + e^{-yx^{T}\theta}}$$

  - interpretation:
    - margin $yx^{T}\theta$ is very positive $\rightarrow p(Y = y \mid x; \theta) = g(yx^{T}\theta) \approx 1$
    - margin $yx^{T}\theta$ is very negative $\rightarrow p(Y = y \mid x; \theta) = g(yx^{T}\theta) \approx 0$
  - redefine the hypothesis class as:

$$h_{\theta}(x) = g(\theta^{T}x) = \frac{1}{1 + e^{-\theta^{T}x}}$$

  - get likelihood of the training data:

$$\begin{aligned}
L(\theta) &= \prod_{i=1}^{m} p\left(Y = y^{(i)} \mid x^{(i)}; \theta\right) \\
&= \prod_{i=1}^{m} h_{\theta}\left(y^{(i)}x^{(i)}\right) \\
l(\theta) &= \sum_{i=1}^{m} \log h_{\theta}\left(y^{(i)}x^{(i)}\right) && \text{get log-likelihood} \\
&= -\sum_{i=1}^{m} \left(1 + e^{-y^{(i)}\theta^{T}x^{(i)}}\right) \\
&= -mJ(\theta) && J(\theta) \text{ is the logistic regression risk (see above)}
\end{aligned}$$

  - therefore, maximum likelihood in the logistic model is equivalent to minimizing the average logistic loss
- **gradient descent methods**
  - to fit the logistic regression model, consider gradient-descent-based minimization
  - the derivative of the logistic loss:

$$\begin{aligned}
&\frac{d}{dz}\varphi_{\text{logistic}}(z) \\
&= \varphi'_{\text{logistic}}(z) \\
&= \frac{1}{1 + e^{-z}} \cdot \frac{d}{dz}e^{-z} \\
&= -\frac{e^{-z}}{1 + e^{-z}} \\
&= -\frac{1}{1 + e^{z}} \\
&= -g(-z)
\end{aligned}$$

  - for a single training example (x, y) (applying chain rule), we have:

$$\begin{aligned}
&\frac{\partial}{\partial \theta_k} \phi_{\text{logistic}}(yx^T\theta) \\
&= -g\left(-yx^T\theta\right) \frac{\partial}{\partial \theta_k}\left(yx^T\theta\right) \\
&= -g\left(-yx^T\theta\right) yx_k
\end{aligned}$$

- thus, a stochastic gradient procedure for minimizing $J(\theta)$ iteratively performs the following for iterations $t = 1, 2, \ldots$, where α_t is a step size at time t:
  1. Choose an example $i \in \{1, \ldots, m\}$ uniformly at random
  2. Perform the gradient update

$$\begin{aligned}
\theta^{(t+1)} &= \theta^{(t)} - \alpha_t \cdot \nabla_\theta \varphi_{\text{logistic}}\left(y^{(i)} x^{(i)T} \theta^{(t)}\right) \\
&= \theta^{(t)} + \alpha_t g\left(-y^{(i)} x^{(i)T} \theta^{(t)}\right) y^{(i)} x^{(i)} \\
&= \theta^{(t)} + \alpha_t h_{\theta^{(t)}}\left(-y^{(i)} x^{(i)}\right) y^{(i)} x^{(i)}
\end{aligned}$$

  - intuition:
    - if our current hypothesis $h_{\theta^{(t)}}$ assigns probability close to 1 for the *incorrect* label $-y^{(i)}$:
      - try to reduce the loss by moving θ in the direction of $y^{(i)} x^{(i)}$
    - conversely, if current hypothesis $h_{\theta^{(t)}}$ assigns probability close to 0 for the incorrect label $-y^{(i)}$:
      - update essentially does nothing

general loss functions

- supervised learning:
  1. choose a representation for the problem (i.e. a hypothesis class)
  2. choose a loss function
  3. minimize the loss
- consider a more general formulation for supervised learning
  - input data $x \in \mathbb{R}^n$
  - targets y from a space $\mathcal{Y}$
    - e.g. in linear regression $\mathcal{Y} = \mathbb{R}$, for binary classification $y \in \mathcal{Y} = \{-1, 1\}$
    - for each of these problems:
      - make predictions based on $\theta^T x$ for some vector θ
      - construct a loss function $\mathcal{L} : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$
      - given a training set of pairs $\left\{x^{(i)}, y^{(i)}\right\}$, choose θ by minimizing the empirical risk

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(\theta^T x^{(i)}, y^{(i)}\right)$$

the representer theorem

- consider an empirical risk with ℓ_2-regularization, i.e. the **regularized risk**

$$J_\lambda(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(\theta^T x^{(i)}, y^{(i)}\right) + \frac{\lambda}{2} \|\theta\|_2^2$$

- consider the structure of any θ that minimizes the risk
  - assume that for each fixed target $y \in \mathcal{Y}$, the loss $\mathcal{L}(z, y)$ is convex in z
    - this is true for linear regression, binary/multiclass logistic regression, and many other losses we will consider

- under these assumptions, the solution (i.e. the θ that minimizes the risk) can always be written as a *linear combination* of the input variables $x^{(i)}$
- this is the **representer theorem**:
  - Suppose in the definition of the regularized risk that $\lambda \geq 0$. Then there is a minimizer of the regularized risk that can be written

$$\theta = \sum_{i=1}^{m} \alpha_i x^{(i)}$$

  - where α_i are real-valued weights
  - an informal proof
    - assume that $\mathcal{L}(z, y)$ is differentiable w.r.t. z, and $\lambda > 0$

$$J_\lambda(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(\theta^T x^{(i)}, y^{(i)}\right) + \frac{\lambda}{2}\|\theta\|_2^2 \qquad \text{the regularized risk}$$

$$\nabla J_\lambda(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \mathcal{L}\left(\theta^T x^{(i)}, y^{(i)}\right) + \lambda \nabla_\theta \frac{1}{2}\|\theta\|_2^2$$

$$\nabla J_\lambda(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}'\left(\theta^T x^{(i)}, y^{(i)}\right) x^{(i)} + \lambda\theta = \vec{0} \qquad \nabla J_\lambda(\theta) = 0 \text{ at the minimum}$$

$$\vec{0} = \frac{1}{m} \sum_{i=1}^{m} w_i x^{(i)} + \lambda\theta \qquad \text{let } w_i = \mathcal{L}'\left(\theta^T x^{(i)}, y^{(i)}\right)$$

$$\theta = -\frac{1}{\lambda m} \sum_{i=1}^{m} w_i x^{(i)}$$

$$\theta = \sum_{i=1}^{m} \alpha_i x^{(i)} \qquad \text{let } \alpha_i = -\frac{w_i}{\lambda m}$$

nonlinear features and kernels

- the representer theorem means that the parameter vector θ can always be written as a linear combination of the data $\left\{x^{(i)}\right\}_{i=1}^{m}$
  - this means we can **always** make predictions

$$\theta^T x = x^T \theta = \sum_{i=1}^{m} \alpha_i x^T x^{(i)}$$

  - i.e., in **any** learning algorithm, we can replace all appearances of $\theta^T x$ with $\sum_{i=1}^{m} \alpha_i x^{(i)T} x$, and then minimize directly over $\alpha \in \mathbb{R}^m$
    - consider this idea in more generality:
      - "original" input values = **attributes**
      - quantities passed to the learning algorithm = **features**
      - ϕ = the **feature mapping** from the attributes to the features
        - to learn with features $\phi(x)$, simply replace x everywhere in the algorithm with $\phi(x)$
        - write the algorithm entirely in terms of inner products $\langle x, z\rangle$, and simply replace the inner products with $\langle \phi(x), \phi(z)\rangle$
        - define the corresponding **kernel** to be

$$K(x, z) = \phi(x)^T \phi(z)$$

        - then, replace $\langle x, z\rangle$ with $K(x, z)$
        - kernelizing the regularized risk

$$J_\lambda(\theta) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(\theta^T x^{(i)}, y^{(i)}\right) + \frac{\lambda}{2}\|\theta\|_2^2 \qquad \text{the regularized risk}$$

$$\begin{aligned} J_\lambda(\alpha) &= \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(\phi\left(x^{(i)}\right)^T \sum_{j=1}^{m}\alpha_j\phi\left(x^{(j)}\right), y^{(i)}\right) + \frac{\lambda}{2}\left\|\alpha_i\phi\left(x^{(i)}\right)\right\|_2^2 \qquad \theta = \sum_{i=1}^{m}\alpha_i\phi\left(x^{(i)}\right) \\ &= \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(\sum_{j=1}^{m}\alpha_j\phi\left(x^{(i)}\right)^T\phi\left(x^{(j)}\right), y^{(i)}\right) + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\phi\left(x^{(i)}\right)^T\phi\left(x^{(j)}\right) \\ &= \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(\sum_{j=1}^{m}\alpha_j K\left(x^{(i)}, x^{(j)}\right), y^{(i)}\right) + \frac{\lambda}{2}\sum_{i,j}\alpha_i\alpha_j K\left(x^{(i)}, x^{(j)}\right) \end{aligned}$$

- i.e., we can write the entire loss function to be minimized in terms of the kernel matrix

$$K = \left[K\left(x^{(i)}, x^{(j)}\right)\right]_{i,j=1}^{m} \in \mathbb{R}^{m\times m}$$

- we could compute $K(x,z)$ by finding $\phi(x)$ and $\phi(z)$ and taking their inner product
  - however, $K(x,z)$ may be very inexpensive to calculate, even though $\phi(x)$ may be very expensive or impossible to calculate
  - by using an efficient way to calculate $K(x,z)$, we can learn in the high dimensional feature space given by ϕ, but without ever having to explicitly calculate or represent vectors $\phi(x)$ (this is the **kernel trick**)
- examples of kernels:
  - **Gaussian/Radial Basis Function (RBF)** kernel:

$$K(x,z) = \exp\left(-\frac{1}{2\tau^2}\|x-z\|_2^2\right)$$

  - min-kernel (applicable when $x \in \mathbb{R}$)

$$K(x,z) = \min\{x,z\}$$

stochastic gradient descent for kernelized machine learning

- let $K \in \mathbb{R}^{m\times m}$ denote the kernel matrix
- for shorthand, define the vectors

$$K^{(i)} = \begin{bmatrix} K\left(x^{(i)}, x^{(1)}\right) \\ K\left(x^{(i)}, x^{(2)}\right) \\ \vdots \\ K\left(x^{(i)}, x^{(m)}\right) \end{bmatrix}$$

- then,

$$K = \begin{bmatrix} | & | & & | \\ K^{(1)} & K^{(2)} & \cdots & K^{(m)} \\ | & | & & | \end{bmatrix}$$

- so the regularized risk can be written as:

$$J_\lambda(\alpha) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(K^{(i)T}\alpha, y^{(i)}\right) + \frac{\lambda}{2}\alpha^T K\alpha$$

- consider taking a stochastic gradient of this risk:

$$J_\lambda(\alpha) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(K^{(i)T}\alpha, y^{(i)}\right) + \frac{\lambda}{2}\alpha^T K\alpha$$

$$\begin{aligned}\nabla_\alpha J_\lambda(\alpha) &= \frac{1}{m}\sum_{i=1}^{m}\nabla_\alpha\mathcal{L}\left(K^{(i)T}\alpha, y^{(i)}\right) + \lambda\nabla_\alpha\left[\frac{1}{2}\alpha^T K\alpha\right] \\ &= \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}'\left(K^{(i)T}\alpha, y^{(i)}\right)K^{(i)} + \lambda\sum_{i=1}^{m}K^{(i)}\alpha_i\end{aligned}$$

- if we choose a random index $i \in \{1, \ldots, m\}$, a stochastic gradient for $J_\lambda(\alpha)$ is then:

$$\mathcal{L}'\left(K^{(i)T}\alpha, y^{(i)}\right)K^{(i)} + m\lambda K^{(i)}\alpha_i$$

- pseudocode for SGD for kernel supervised learning problems:
  - input:
    - loss function $\mathcal{L}$
    - kernel matrix $K = \left[K^{(1)} \cdots K^{(m)}\right]$
    - labels $\left\{y^{(i)}\right\}_{i=1}^m$
    - sequence of positive stepsizes $\eta_1, \eta_2, \eta_3, \ldots$
  - **iterate** for $t = 1, 2, \ldots$
    1. Choose index $i \in \{1, \ldots, m\}$ uniformly at random
    2. Update

$$\alpha \leftarrow \alpha - \eta_t\left[\mathcal{L}'\left(K^{(i)T}\alpha, y^{(i)}\right)K^{(i)} + m\lambda K^{(i)}\alpha_i\right]$$

  - note: because the $\lambda K^{(i)}\alpha_i$ term is multiplied by m to keep the gradient unbiased, it is important that $\lambda > 0$ not be too large, as the algorithm can be unstable otherwise
  - note: a common stepsize is $\eta_t = \frac{1}{\sqrt{t}}$, or a constant multiple thereof

support vector machines

- one approach to SVMs:
  - use the margin-based loss function

$$\mathcal{L}(z, y) = [1 - yz]_+ = \max\{0, 1 - yz\}$$

  - the empirical regularized risk is then:

$$J_\lambda(\alpha) = \frac{1}{m}\sum_{i=1}^{m}\left[1 - y^{(i)}K^{(i)T}\alpha\right]_+ + \frac{\lambda}{2}\alpha^T K\alpha$$

gaussian/RBF kernel example

$$K(x, z) = \exp\left(-\frac{1}{2\tau^2}\|x - z\|_2^2\right)$$

- $\tau > 0$ controls the **bandwidth** of the kernel
  - for small τ, $K(x, z) \approx 0$ unless $x \approx z$
  - for large τ, the kernel function K is much smoother
- the feature function ϕ for the RBF kernel is infinite dimensional (it is the Fourier transform of the Gaussian distribution with mean zero and variance τ^2)
- to make a new prediction:

$$
\begin{aligned}
& \theta^T x \\
&= \sum_{i=1}^{m} \alpha_i x^T x^{(i)} && \text{representer theorem} \\
&= \sum_{i=1}^{m} K\left(x^{(i)}, x\right) \alpha_i && \text{substitute kernel function} \\
&= \sum_{i=1}^{m} \exp\left(-\frac{1}{2\tau^2} \|x^{(i)} - x\|_2^2\right) \alpha_i
\end{aligned}
$$

- this represents something like a weighting depending on how close x is to each $x^{(i)}$
  - i.e., the contribution of weight α_i is scaled by the similarity of x to $x^{(i)}$ as determined by the kernel function
- large $\tau \rightarrow$ a very simple, close to linear classifier
- small $\tau \rightarrow$ a variable, highly non-linear classifier