CPSC 340 Tutorial

Linear Classifiers

Dylan Green

Table of Contents

Regression for Binary Classification

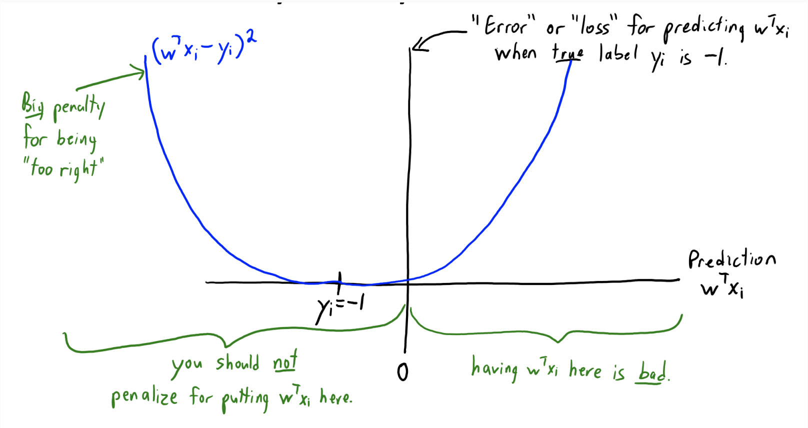
Regression for Binary Classification

- How can we apply all of our machinery (regularization, change of basis, etc) for linear regression problems to binary classification problems?
- Natural idea: $\hat{y}_i = \text{sign}(w^\top \hat{x}_i)$, i.e.:

$$\hat{y}_i = \begin{cases} +1 & \text{if } w^\top \hat{x}_i > 0 \\ -1 & \text{if } w^\top \hat{x}_i \leq 0 \end{cases}$$

- What loss function could we use to train such a model?

Squared loss?



0-1 Loss

- What we *want* is the 0-1 loss:

$$\|\hat{y} - y\|_0 = \sum_{i=1}^{n} 1\left(\hat{y}_i \neq y_i\right)$$

- Good: measures number of classification errors
- Bad: non-convex, gradient is 0 everywhere
- We introduce two convex approximations to the 0-1 loss

But first...

	$w^T x_i$ negative	$w^T x_i$ positive
$y_i = -1$ **(negative)**	✅	❌
$y_i = +1$ **(positive)**	❌	✅

The prediction is correct if and only if $y_i w^\top x_i > 0$.

Hinge Loss

- We can use this to minimize how much this constraint is violated:
  - If $y_i w^\top x_i > 0$, get an error of 0
  - If $y_i w^\top x_i < 0$, get an error of $-y_i w^\top x_i$

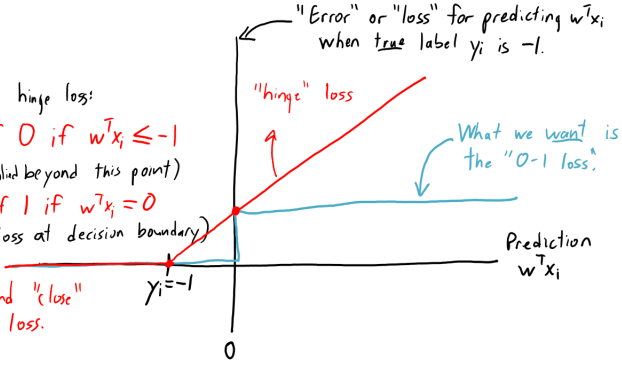
  giving a loss for one example of $\max\{0, -y_i w^\top x_i\}$
- This is convex, but degenerate; $w = 0$ achieves the minimum possible loss.
- One solution: make the condition more strict,
  - If $y_i w^\top x_i > 1$, get an error of 0
  - If $y_i w^\top x_i < 1$, get an error of $1 - y_i w^\top x_i$

  giving a loss for one example of $\max\{0, 1 - y_i w^\top x_i\}$. This is the hinge loss.

Hinge Loss: Convex Approximation to 0-1 Loss

Properties of the hinge loss:

1. Has error of 0 if $w^Tx_i \leq -1$
   (no penalty applied beyond this point)
2. Has a loss of 1 if $w^Tx_i = 0$
   (matches 0-1 loss at decision boundary)
3. Is convex and "close" to 0-1 loss.



Hinge Loss

- SVMs: Hinge loss plus L2 regularization.
- SVMs with RBF kernels (later) one of the best out-of-the-box classifiers, still hugely popular
- Piazza question:
  - From slides: "If the hinge loss is 18.3, number of training errors is at most 18". Why?

Logistic Loss

- We can also use a smooth approximation of the degenerate loss:

$$\max\{0, -y_i w^\top x_i\} \approx \log(1 + \exp(-y_i w^\top x_i))$$

- Convex, smooth, non-degenerate
- Has probabilistic interpretation (later)

Linear Classifiers

- Logistic regression and SVMs remain are hugely popular
- Fast training and testing
- Interpretable weights
- Largely interchangeable (no free lunch)
- Building blocks for neural networks (later)

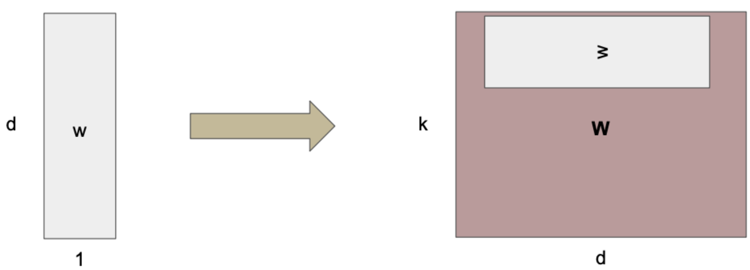
Multi-class Linear Classifiers

Multi-class Linear Classifiers

- Can we extend what we just did to cases with more than two labels?
- Basic idea: have a weight vector for each class c, predict whichever class has the largest output $w_c^\top x_i$
- w_{y_i} is the weight vector for the correct class of example i

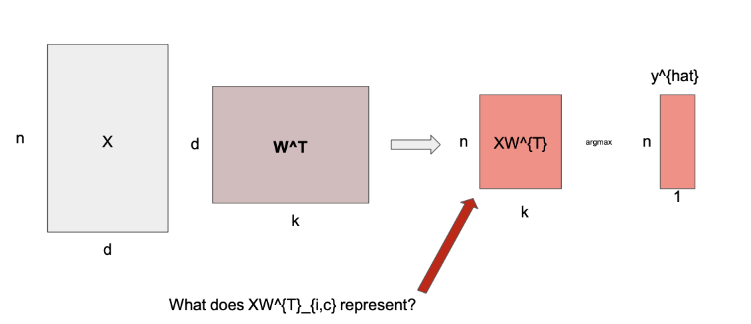
Before we get to losses...

Single class classification



What is k in the multi-class classification weight matrix?

Before we get to losses...



One vs. All

One vs. All approach:

- For k classes, training a linear classifier for each class c
- At prediction time, predict the class with the highest score $w_c^\top x_i$
- Problem: we didn't train the w_c so that the largest $w_c^\top x_i$ would be $w_{y_i}^\top x_i$; each classifier just tries to get the sign right

Multi-class Hinge Loss

Following the same steps as in the single class case...

- Want $w_{y_i}^\top x_i > w_c^\top x_i$ for all c that are not y_i
- Use $w_{y_i}^\top x_i > w_c^\top x_i + 1$ to avoid degeneracy
- Two ways to measure constraint violation:
  - Sum:
$$\sum_{c \neq y_i} \max\{0, 1 + w_c^\top x_i - w_{y_i}^\top x_i\}$$
  - Max:
$$\max_{c \neq y_i}\{\max\{0, 1 + w_c^\top x_i - w_{y_i}^\top x_i\}\}$$

Softmax Loss (Multi-class Logistic Loss)

- Degenerate constraint for multiclass case:

$$w_{y_i}^\top x_i \geq \max_c \{w_c^\top x_i\}$$

- Re-write as

$$0 \geq -w_{y_i}^\top x_i + \max_c \{w_c^\top x_i\}$$

- To make this "as true as possible", make right side as small as possible
- Smoothing the max with log-sum-exp gives a convex, non-degenerate loss for one training example of

$$-w_{y_i}^\top x_i + \log\left(\sum_{c=1}^{k} \exp(w_c^\top x_i)\right)$$

Softmax Function

- Softmax loss also has a probabilistic interpretation (more details later in the course)
- The "scores" $z_c = w_c^\top x_i$ can be mapped to probabilities with the softmax function:

$$p(y = a | z_1, z_2, \ldots, z_k) = \frac{\exp(z_a)}{\sum_{c=1}^{k} \exp(z_c)}$$

Softmax Gradient

The softmax function for k classes:

$$f(W) = \sum_{i=1}^{n} \left[-w_{y_i}^T x_i + \log \left(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i) \right) \right],$$

How are these two weight vectors different?

Softmax Gradient

The softmax function for k classes:

$$\frac{\partial f}{\partial W_{cj}} = \sum_{i=1}^{n} x_{ij}[p(y_i = c \mid W, x_i) - I(y_i = c)]$$

What is this taking the partial derivative of?
What are c and j?
What is the “size” of this (vector or scalar)?

- $I(y_i = c)$ is the indicator function (it is 1 when $y_i = c$ and 0 otherwise)
- $p(y_i = c \mid W, x_i)$ is the predicted probability of example i being class c, defined as

$$p(y_i = c \mid W, x_i) = \frac{\exp(w_c^T x_i)}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)}$$

Expanding the product

The softmax function for k classes:

$$f(W) = \sum_{i=1}^{n} \left[\boxed{-w_{y_i}^T x_i} + \log \left(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i) \right) \right],$$

$$\Downarrow$$

$$w_{y_i}^T x_i = w_{y_i,0} x_{i,0} + w_{y_i,1} x_{i,1} + \ldots + w_{y_i,d} x_{i,d}$$

Softmax Gradient

The softmax function for k classes:

$$f(W) = \sum_{i=1}^{n} \left[\boxed{-w_{y_i}^T x_i} + \log\left(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i) \right) \right],$$

What is the partial derivative of this w.r.t class c?
Is it always non-zero?

$$\frac{\partial f}{\partial W_{cj}} = \sum_{i=1}^{n} x_{ij} [p(y_i = c \mid W, x_i) - I(y_i = c)]$$

- $I(y_i = c)$ is the indicator function (it is 1 when $y_i = c$ and 0 otherwise)

Tips for Coding Softmax

- Implement it with as many for loops as you need and make sure it works!
- Then if you want you can try and speed up the computation through precomputing and vectorization
- Make sure that the dimensions of your gradients is correct
- Make sure your indexing for your matrices are correct (if applicable)

Speeding Up Softmax

- Look for things to pre-compute:
  - Are there any matrix computations used repeatedly?
  - Are certain matrices able to be computed independently and reused?

- Use NumPy broadcasting/vectorization for quick matrix multiplication
- Use NumPy array operations (np.sum, np.exp, np.log) where applicable