Machine Learning (CSE 446): (Supervised) Learning as Loss Minimization: Linear Regression

Sham M Kakade

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

Announcements

- HW2 posted, milestone due this weds
- HW2 extra credit posted
- updated HW Late policy (see website)

The singular value decomposition

- Let M be a symmetric matrix.
  SVDs also work for asymmetric matrices, with a slightly modified thm.
- SVD theorem: there exists a decomposition of the following form:

$$M = UDU^\top$$

  where D is a diagonal matrix and U is an orthogonal matrix (i.e. the columns of U are unit length and orthogonal to each other).
- The columns of U are eigenvectors of M.
- For PCA, you will take Σ to be M.

Projection and Reconstruction: the one dimensional case

- Take out mean μ: $x_i \leftarrow x_i - \mu$
- Find the "top" eigenvector u_1 of the covariance matrix, with eigenvalue λ_1
- What are your projection onto u_1 (i.e. writing x_i in the u_1 basis)?

$$(x_i \cdot u_1)$$

- What are your reconstructions, $\widehat{\mathbf{X}} = [\widehat{\mathbf{x}}_1 | \widehat{\mathbf{x}}_2 | \cdots | \widehat{\mathbf{x}}_N]^\top$?

$$\widehat{x}_i = (x_i \cdot u_1) u_1 + \mu$$

- What is is your reconstruction error?

$$\frac{1}{N} \sum_i \|\mathbf{x}_i - \widehat{\mathbf{x}}_i\|^2 = \lambda_2 + \ldots \lambda_d$$

(also, we if did nothing and projected everything to μ, then:

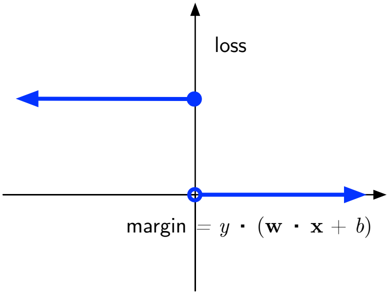
$$\frac{1}{N} \sum_i \|\mathbf{x}_i - \mu\|^2 = \lambda_1 + \lambda_2 + \ldots \lambda_d$$

so we 'save' λ_1 in our error.

Today: how do we *efficiently* do supervised learning?

"Minimize training-set error rate":

$$\min_{\mathbf{w}, b} \frac{1}{N} \sum_{i=1}^{N} \underbrace{\mathbf{1}\{y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \leq 0\}}_{\text{zero-one loss on a point } n}$$



This problem is NP-hard; even for a (multiplicative) approximation. PERCEPTRON ALGORITHM: A model and an algorithm, rolled into one.

Is there a more principled methodology to derive algorithms?

Relax!

- The mis-classification optimization problem:

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\{y_i(\mathbf{w} \cdot \mathbf{x}_i) \leq 0\}$$

- Instead, let's try to choose a "reasonable" loss function $\ell(y_i, \mathbf{w} \cdot \mathbf{x})$ and then try to solve the **relaxation:**

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, \mathbf{w} \cdot \mathbf{x}_i)$$

What is a good "relaxation"?

- Want that minimizing our **surrogate** loss helps with minimizing the mis-classification loss.
  - idea: try to use a (sharp) upper bound of the zero-one loss by ℓ:

$$\mathbf{1}\{y(\mathbf{w} \cdot \mathbf{x}) \leq 0\} \leq \ell(y, \mathbf{w} \cdot \mathbf{x})$$

- want our **relaxed** optimization problem to be easy to solve.
  What properties might we want for $\ell(\cdot)$?
  - differentiable? sensitive to changes in w?
  - **convex?**

The square loss as an upper bound

- We have:
$$\mathbf{1}\{y(\mathbf{w}\cdot\mathbf{x}) \leq 0\} \leq (y - \mathbf{w}\cdot\mathbf{x})^2$$
- Easy to see, by plotting:

A better (convex) upper bound

- The logistic loss:

$$\ell^{\text{logistic}}(y, \mathbf{w} \cdot \mathbf{x}) = \log\left(1 + \exp(-y\mathbf{w} \cdot \mathbf{x})\right).$$

- We have:

$$\mathbf{1}\{y(\mathbf{w} \cdot \mathbf{x}) \leq 0\} \leq \text{constant} \, * \, \ell^{\text{logistic}}(y, \mathbf{w} \cdot \mathbf{x})$$

- Again, easy to see, by plotting:

The square loss! (and linear regression)

- The square loss: $\ell(y, \mathbf{w} \cdot \mathbf{x}) = (y - \mathbf{w} \cdot \mathbf{x})^2$.
- The relaxed optimization problem:

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2$$

- nice properties:
  - for binary classification, it is a an upper bound on the zero-one loss.
  - It makes sense more generally, e.g. if we want to predict real valued y.
  - We have a convex optimization problem.
- For classification, what is your decision rule using a $\mathbf{w}$?

Remember this problem?

Data derived from `https://archive.ics.uci.edu/ml/datasets/Auto+MPG`

mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
18.0	8	307.0	130.0	3504.	12.0	70	1
15.0	8	350.0	165.0	3693.	11.5	70	1
18.0	8	318.0	150.0	3436.	11.0	70	1
16.0	8	304.0	150.0	3433.	12.0	70	1
17.0	8	302.0	140.0	3449.	10.5	70	1
15.0	8	429.0	198.0	4341.	10.0	70	1
14.0	8	454.0	220.0	4354.	9.0	70	1
14.0	8	440.0	215.0	4312.	8.5	70	1
14.0	8	455.0	225.0	4425.	10.0	70	1
15.0	8	390.0	190.0	3850.	8.5	70	1
15.0	8	383.0	170.0	3563.	10.0	70	1
14.0	8	340.0	160.0	3609.	8.0	70	1
15.0	8	400.0	150.0	3761.	9.5	70	1
14.0	8	455.0	225.0	3086.	10.0	70	1
24.0	4	113.0	95.00	2372.	15.0	70	3
22.0	6	198.0	95.00	2833.	15.5	70	1
18.0	6	199.0	97.00	2774.	15.5	70	1
21.0	6	200.0	85.00	2587.	16.0	70	1
27.0	4	97.00	88.00	2130.	14.5	70	3
26.0	4	97.00	46.00	1835.	20.5	70	2
25.0	4	110.0	87.00	2672.	17.5	70	2
24.0	4	107.0	90.00	2430.	14.5	70	2

Input: a row in this table.

Goal: predict whether mpg is < 23 ("bad" $= 0$) or above ("good" $=$ 1) given the input row.

Predicting a real y (often) makes more sense.

Least squares: let's minimize it!

- The optimization problem:

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2 =$$
$$\min_{\mathbf{w}} \|Y - X\mathbf{w}\|^2$$

  where Y is an N-vector and X is our $N \times d$ data matrix.
- How do we interpret $X\mathbf{w}$?

The solution is the **least squares estimator**:

$$\mathbf{w}^{\text{least squares}} = (X^\top X)^{-1} X^\top Y$$

Vector calculus hints I

- suppose we have a function $f(w) = w \cdot c = \sum_j w[j]c[j]$, where w and c is a d-dimensional vector.
- Elementary calculus tells gives us scalar derivatives:

$$\frac{\partial f(w)}{\partial w[i]} = c[i]$$

- The *gradient* is the vector of all the partial derivatives:

$$\nabla f(w) := \left(\frac{\partial f(w)}{\partial w[1]}, \frac{\partial f(w)}{\partial w[2]}, \ldots \frac{\partial f(w)}{\partial w[d]} \right)^\top$$

- So we have that:

$$\nabla f(w) = c$$

Vector calculus hints II

- suppose we have a function
$$f(w) = w^\top M w = \sum_{j,k} w[j]w[k]M[j,k]\,,$$
where M is a *symmetric* $d \times d$ matrix.
- Elementary calculus tells gives us scalar derivatives:
$$\frac{\partial f(w)}{\partial w[i]} = 2\sum_j M[i,j]w[j]$$
- The *gradient* is just the matrix of all the partial derivatives:
$$\nabla f(w) := \left(\frac{\partial f(w)}{\partial w[1]}, \frac{\partial f(w)}{\partial w[2]}, \ldots \frac{\partial f(w)}{\partial w[d]}\right)$$
- It is straightforward to see that a far more compact way to write the gradient is:
$$\nabla f(w) = 2Mw$$
(just equate each coordinate with the scalar derivative).

Lots of questions:

- What could go wrong with least squares?
  - Suppose we are in "high dimensions": more dimensions than data points.
  - Inductive bias: we need a way to control the complexity of the model.
- Optimization: how do we do this all quickly?