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

Sham M Kakade
© 2019

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

- HW2 posted, milestone due this wed
- 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 $\Sigma$ to be $M$. 


Projection and Reconstruction: the one dimensional case

- Take out mean $\mu$: $x_i \leftarrow x_i - \mu$
- Find the “top” eigenvector $u_1$ of the covariance matrix, with eigenvalue $\lambda_1$
- 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, $\hat{X} = [\hat{x}_1 | \hat{x}_2 | \cdots | \hat{x}_N]^\top$?
  \[\hat{x}_i = (x_i \cdot u_1)u_1 + \mu\]
- What is is your reconstruction error?
  \[\frac{1}{N} \sum_i \|x_i - \hat{x}_i\|^2 = \lambda_2 + \ldots \lambda_d\]
  (also, we if did nothing and projected everything to $\mu$, then:
  \[\frac{1}{N} \sum_i \|x_i - \mu\|^2 = \lambda_1 + \lambda_2 + \ldots \lambda_d\]
  so we ’save’ $\lambda_1$ in our error.
Today: how do we efficiently do supervised learning?

“Minimize training-set error rate”:

\[
\min_{\mathbf{w},b} \frac{1}{N} \sum_{i=1}^{N} 1\{y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \leq 0\}
\]

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?
The mis-classification optimization problem:

\[
\min_w \frac{1}{N} \sum_{i=1}^{N} 1\{y_i(w \cdot x_i) \leq 0\}
\]

Instead, let’s try to choose a “reasonable” loss function \(\ell(y_i, w \cdot x)\) and then try to solve the relaxation:

\[
\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, w \cdot x_i)
\]
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 $\ell$:

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

▶ 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:
  \[1\{y(w \cdot x) \leq 0\} \leq (y - w \cdot x)^2\]

- Easy to see, by plotting:
A better (convex) upper bound

- The logistic loss:

\[ \ell_{\text{logistic}}(y, w \cdot x) = \log (1 + \exp(-yw \cdot x)). \]

- We have:

\[ 1 \{ y(w \cdot x) \leq 0 \} \leq \text{constant} \times \ell_{\text{logistic}}(y, w \cdot x) \]

- 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 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

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

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:

\[
\min_w \frac{1}{N} \sum_{i=1}^{N} (y_i - w \cdot x_i)^2 = \\
\min_w \|Y - Xw\|^2
\]

where \(Y\) is an \(N\)-vector and \(X\) is our \(N \times d\) data matrix.

- How do we interpret \(Xw\)?

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

\[
w^{\text{least squares}} = (X^\top X)^{-1} X^\top Y
\]
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:

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

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 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?