# Linear Models

Pramook Khungurn

December 11, 2019

## 1 Univariate Linear Regression

- Given examples $(x_1, y_1)$, $(x_2, y_2)$, $\ldots$, $(x_N, y_N)$ where $x_i, y_i \in \mathbb{R}$, we want to find a function of the form $y = w_1 x + w_0$ that fits the examples best.
- The values $w_0$ and $w_1$ are called **weights**. Let $\mathbf{w} = [w_0, w_1]$, define

$$h_{\mathbf{w}}(x) = w_1 x + w_0.$$

- By "fits best", we mean we want to find $\mathbf{w}$ that minimizes the empirical loss.
- It is natural to use the the squared lost function $L_2$:

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^{N} (y_j - (w_1 x + w_0))^2.$$

- The loss above is minimized when

$$\frac{\partial}{\partial w_0} Loss(h_{\mathbf{w}}) = 0, \text{ and } \frac{\partial}{\partial w_1} Loss(h_{\mathbf{w}}) = 0.$$

  Solving these equations yields a unique pairs of $w_0$ and $w_1$, which gives us the optimal linear hypothesis.
- Most linear learning models involve searching for appropriate weights in the **weight space**.
- If the loss function is **convex**, then it can be shown that the weight space contains no local optima. This is true of all $L_p$ loss function (or norm).
- Most of the time, however, the equations that define the locus of minimum loss does not have a closed-form solution.

  So, we face a general optimization problem in a continuous weight space.
- Such a problem can be tackled by a hill-climbing algorithm that follows the **gradient** of the function to be optimized.
- Because we try to minimize the loss, we use the *gradient descent* algorithm, which goes as follows:

```
1: w ← any point in the weight space
2: loop until convergence do
3:     for each w_i in w do
4:         w_i ← w_i − α ∂/∂w_i Loss(h_w)
```

  The parameter $\alpha$ is called the **step size** or the **learning rate**.

- Let us figure out the update rule in Line 4 of the above algorithm for the $L_2$ loss function.

  We have that

  $$\frac{\partial}{\partial w_0} Loss_{L_2}(h_{\mathbf{w}}) = -2\sum_j (y_j - \mathbf{h}_{\mathbf{w}}(x_j)), \text{ and } \frac{\partial}{\partial w_1} Loss_{L_2}(h_{\mathbf{w}}) = -2\sum_j (y_j - \mathbf{h}_{\mathbf{w}}(x_j))x_j$$

  Therefore, the update rules are

  $$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)), \text{ and}$$
  $$w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j))x_j.$$

  These update rules are called the **batch gradient descent** learning rule for univariate linear regression.

- There is an alternative update rule called the **stochastic gradient descent** where we update using only one example at a time (as opposed to the sum of every examples as in the above rules). This can be used in online learning setting where data comes in one example at a time.

- With a fixed learning rate $\alpha$, the stochastic gradient descent does not converge. In some cases, however, a schedule of decreasing learning rates does guarantee convergence.

# 2 Multivariate Linear Regression

- In **multivariate regression problem**, the input to the hypothesis is a vector $\mathbf{x}$ of $n$ dimensions. A hypothesis is a function of the form

  $$h(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = w_0 + \sum_i w_i x_i.$$

  To make the hypothesis easier to write, we may introduce a component $x_0$, which is always equal to 1. (This is basically putting $\mathbf{x}$ in homogeneous coordinate.) The hypothesis then becomes

  $$h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x}$$

  where $\mathbf{w} = (w_0, w_1, \ldots, w_n)^T$ and $\mathbf{x} = (1, x_1, x_2, \ldots, x_n)^T$.

- The best weight vector is again the one that minimizes the empirical loss:

  $$\mathbf{w}^* = \text{argmin} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j).$$

- We can use gradiant descent with multivariate regression. The update rule is pretty much identical to that of the univariate case:

  $$w_i \leftarrow w_i + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j))x_{j,i}.$$

- It is possible to solve for $\mathbf{w}$ that minimizes $L_2$ loss. This is just basically the least square problem, and we have

  $$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

  where $\mathbf{X}$ is the **data matrix** where each column is an example attributes and $\mathbf{y}$ is the column vectors of target function values.

- With multivariate regression in high-dimensional space, we need to worry about overfitting because it is possible that some irrelevant dimension migght influence the function values.
- We can use regularization on regression to reduce overfitting. The complexity function of a linear hypothesis that is often used is:

$$L_q(\mathbf{w}) = \sum_i |w_i|^q.$$

- Using $L_1$ tends to produce **sparse model**. Sparse models are easier to understand and less likely to overfit.

# 3 Linear Models for Classification

- Linear models can be used for classification as well.
- Given points of two classes, a **decision boundary** is a curve (or a surface) that separates the two classes.
- A **linear separator** is a decision boundary that takes the form

$$\mathbf{w} \cdot \mathbf{x} = 0$$

  for some weight vector $\mathbf{w}$. Again, the vectors are in homogeneous coordinates.
- The classification hypothesis with linear separators can be written as:

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} \geq 0 \\ -1 & \text{otherwise} \end{cases}.$$

  Alternatively, we can think of the hypothesis as passing $\mathbf{w} \cdot \mathbf{x}$ to the **threshold function**:

$$h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x})$$

  where

$$Threshold(z) = \begin{cases} 1 & z \geq 0 \\ -1 & z < 0 \end{cases}.$$

- We cannot learn the compute the linear classifier by computing the gradients and set it to zero. The reason is that the gradient is zero almost everywhere except at the points where $\mathbf{w} \cdot \mathbf{x} = 0$. At these points, however, the gradient is undefined.
- There is an update rule that converges to a correct separator given that the data is separable:

$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))x_i$$

  which is identical to the update rule for linear regression. This update rule is called the **perceptron update rule**. It is just the linear regression update rule when the loss function is a 0/1 loss function and the hypothesis is a classification hypothesis instead of a continuous one.
- When the data is not linearly separable, the perceptron learning rule might not converge.

  In general, with a fixed rate $\alpha$, the perceptron might not converge.

  However, if the rate $\alpha = O(1/t)$ where $t$ is the iteration number, then the rule can be shown to converge to a minimum-error solution when examples are presented in a random sequence.
- Finding minimum error separator is NP-hard. So, one expects a lot of iterations are required for convergence.

# 4 Optimal Separator

- Let us assume that the training examples are linearly separable.
- **Definition 4.1.** *For a linear classifier $h_{\mathbf{w}}$, the* margin $\delta$ *of an example* $(\mathbf{x}, y)$ *is* $\delta = (\mathbf{w} \cdot \mathbf{x})y$.
- **Definition 4.2.** *The margin is called* geometric margin *if* $\sqrt{w_1^2 + w_2^2 + \cdots + w_n^2} = 1$. *Otherwise, it is called* functional margin.
- We might want to find an optimal separator. The one with the largest distance to the closest traning examples.

  The optimization problem we want to solve is the following:

  minimize $\frac{1}{2}(w_1^2 + w_2^2 + \cdots + w_n^2)$
  subjected to:

$$\begin{aligned}(\mathbf{w} \cdot \mathbf{x_1})y_1 &\geq 1\\ (\mathbf{w} \cdot \mathbf{x_2})y_2 &\geq 1\\ &\vdots\\ (\mathbf{w} \cdot \mathbf{x_n})y_n &\geq 1\end{aligned}$$

- **Support vectors** are examples with minimal margin.
- **Definition 4.3.** *The* hard margin *of a linear classifier $h_{\mathbf{w}}$ on data $D$ is*

$$\delta = \min_{(\mathbf{x},y)\in D}\{(\mathbf{w} \cdot \mathbf{x})y\}.$$

- For non-separable data, we cannot define a hard margin.
- Moreover, complete separation (zero training error) can lead to suboptimal prediction error. (Consider the case where one outlier negative example is very close to the set of positive examples.)
- We can relax the notion of hard margin by adding some slack variables to the semi-definite program above:

  minimize $\frac{1}{2}(w_1^2 + w_2^2 + \cdots + w_n^2) + C\sum_{i=1}^{n}\xi_i$
  subjected to:

$$\begin{aligned}(\mathbf{w} \cdot \mathbf{x_1})y_1 &\geq 1 - \xi_1\\ (\mathbf{w} \cdot \mathbf{x_2})y_2 &\geq 1 - \xi_2\\ &\vdots\\ (\mathbf{w} \cdot \mathbf{x_n})y_n &\geq 1 - \xi_n\\ \xi_1 &\geq 0\\ \xi_2 &\geq 0\\ &\vdots\\ \xi_n &\geq 0\end{aligned}$$

  The slack variable $\xi_i$ measures by how much $(\mathbf{x}_i, y_i)$ fails to achieve margin $\delta$.

  Moreover, $\sum \xi_i$ is the upperbound on training error.

  The parameter $C$ controls tradeoff between margin and training error.

# 5 Logistic Regression

- When creating a linear classifier, the hard nature of the threshold function causes many problems.
  - The hypothesis $h_{\mathbf{w}}$ is not differentiable.
  - The prediction is "hard." There's no room to indicate uncertainty in the prediction.
- We can solve this problem by replacing it with a differentiable alternative. The most popular one is the **logistics function**.

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

  Using the logisticcs function, a hypothesis takes the following form:

  $$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}.$$

- The output of the logistic function is always between 0 and 1. So, we can interpret the output of the hypothesis as the probability of the input belonging to class 1.
- The process of learning this new hypothesis space is called **logistic regression**. There is no close-form solution, so we do it by gradient descent.
- Let us find the gradient of the loss function of a hypothesis. Let $g(x) = Logistic(x)$. We have

  $$\begin{aligned}
  \frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\
  &= 2(y - h_{\mathbf{w}}(\mathbf{x}))\frac{\partial}{\partial w_i}(y - h_w(\mathbf{x})) \\
  &= -2(y - h_{\mathbf{w}}(\mathbf{x}))g'(\mathbf{w} \cdot \mathbf{x})\frac{\partial}{\partial w_i}\mathbf{w} \cdot \mathbf{x} \\
  &= -2(y - h_{\mathbf{w}}(\mathbf{x}))g'(\mathbf{w} \cdot \mathbf{x})x_i.
  \end{aligned}$$

  The derivative of the logistic function satisfies $g'(z) = g(z)g(1 - z)$. So, we have

  $$g'(\mathbf{w} \cdot \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})(1 - \mathbf{w} \cdot \mathbf{x}) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})).$$

  The weight update rule is thus:

  $$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))x_i.$$