CS 181 Spring 2019 Section 3 Notes
(Bayesian Approaches, Gradient Descent, Linear Classification)
Interactive Demo

1 Bayesian Linear Regression

1.1 Parameter Distributions

Let $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, $\mathbf{x}_i \in \mathbb{R}^m$, $y_i \in \mathbb{R}$. Consider the generative model:

$$y_i \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x}_i, \beta^{-1}) \tag{1}$$

The likelihood of the data has the form:

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, \beta^{-1}\mathbf{I}) \tag{2}$$

Put a conjugate prior on the weights (assume precision β^{-1} known):

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_0, \mathbf{S}_0) \tag{3}$$

We want a posterior distribution on $\mathbf{w}$. Using Bayes' Theorem:

$$p(\mathbf{w}|D) \propto p(D|\mathbf{w})p(\mathbf{w}) \tag{4}$$

It turns out that our posterior after n examples is also Gaussian:

$$p(\mathbf{w}|D) = \mathcal{N}(\mathbf{w}|\mathbf{m}_n, \mathbf{S}_n) \tag{5}$$

where

$$\mathbf{S}_n = \left(\mathbf{S}_0^{-1} + \beta\mathbf{X}^\top\mathbf{X}\right)^{-1} \tag{6}$$

$$\mathbf{m}_n = \mathbf{S}_n(\mathbf{S}_0^{-1}\mathbf{m}_0 + \beta\mathbf{X}^\top\mathbf{y}) \tag{7}$$

1.2 Posterior Predictive Distributions

We have seen how to obtain a posterior distribution over $\mathbf{w}$. But, given this posterior and a new data point $\mathbf{x}^*$, how do we actually make a prediction y^*? How do we deal with *uncertainty* about $\mathbf{w}$? Consider this:

$$p(y^*|\mathbf{x}^*, \mathcal{D}) = \int_{\mathbf{w}} p(y^*|\mathbf{x}^*, \mathbf{w}) p(\mathbf{w}|\mathcal{D}) d\mathbf{w} \tag{8}$$

$$= \int_{\mathbf{w}} \mathcal{N}(y^*|\mathbf{w}^\top \mathbf{x}^*, \beta^{-1}) \mathcal{N}(\mathbf{w}|\mathbf{m}_n, \mathbf{S}_n) d\mathbf{w} \tag{9}$$

This is the **posterior predictive** distribution over y^*. This can be interpreted as a weighted average of many predictors, one for each choice of $\mathbf{w}$, weighted by how likely $\mathbf{w}$ is according to the posterior. Since each of the terms on the right hand side follows a normal distribution, we can use some math (see CS181 2017 lecture 5, slide 33) to find that:

$$p(y^*|\mathbf{x}^*, \mathcal{D}) = \mathcal{N}(y^*|\mathbf{x}^{*\top} \mathbf{m}_n, \mathbf{x}^{*\top} \mathbf{S}_n \mathbf{x}^* + \beta^{-1}) \tag{10}$$

2 Gradient Descent

Gradient descent is an iterative algorithm for finding the minimum of a function.

2.1 Gradient vector

If f is a scalar-valued, its derivative is a column vector we call the gradient vector.

$$\frac{df(\mathbf{x})}{d\mathbf{x}} = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \dots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{pmatrix} \tag{11}$$

The gradient vector points in the direction of steepest ascent in $f(\mathbf{x})$. This is useful for optimization.

2.2 Gradient Descent Algorithm

If we want to find the value of a vector $\mathbf{w}$ that minimizes a function $f(\mathbf{w})$, we start with an initial guess $\mathbf{w}^{(0)}$ and at each time step i we update our guess by going in the direction

of the greatest descent (opposite the direction of the gradient vector).

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \frac{df(\mathbf{w}^{(i)})}{d\mathbf{w}^{(i)}} \tag{12}$$

where $\eta > 0$ is known as the learning rate.

See this notebook for an interactive demo of gradient descent with visualizations.
Note: The notebook also has information about one-hot encodings.

2.3 Stocastic Gradient Descent

This method makes an update to the weight vector based on one data point at a time.

The algorithm is as follows:

1. Sample data point i, corresponding to $(\mathbf{x}_i, y_i)$
   This is done by cycling through the data either in sequence or by selecting points at random with replacement.
2. Compute loss and gradient for for just that data point
3. Update w based on this stochastic gradient.

3 Classification

The goal in classification is to take an input vector $\mathbf{x}$ and assign it to one of K discrete classes C_k, where $k = 1, \ldots, K$. The input space is thus divided into **decision regions** whose boundaries are called **decision boundaries or surfaces**.

3.1 Binary Linear Classification

A discriminant function is one that directly assigns each vector $\mathbf{x}$ to a specific class. We first assume two classes, i.e. our responses are binary and $K = 2$. Linear classification seeks to divide the 2 classes by a linear separator in the feature space: if $m = 2$, the separator is a line; if $m = 3$, the separator is a plane; for general m, the separator is a $(m-1)$-dimensional hyperplane. The simplest representation of a linear discriminant function is obtained by taking a linear function of the input vector as such:

$$h(\mathbf{x}; \mathbf{w}, w_0) = \mathbf{w}^\top \mathbf{x} + w_0 \tag{13}$$

The corresponding decision boundary is defined by the relation $h(\mathbf{x}; \mathbf{w}, w_0) = 0$. The classifier will predict $\hat{y} = 1$ if $h(\mathbf{x}; \mathbf{w}, w_0) > 0$, and predict $\hat{y} = -1$ otherwise.

Weight vector $\mathbf{w}$ is orthogonal to every vector lying within the decision surface, and so $\mathbf{w}$ determines the orientation of the decision boundary. Remember the familiar plane equations $c_1x + c_2y + c_3z + c_4 = 0$ for fixed constants $\mathbf{c}$ and variables x, y, z? These are just dot products with an orthogonality constraint, and thus, they are precisely the kinds of boundaries described here. The input space can also be transformed through a (potentially non-linear) basis function: $h(\mathbf{x}; \mathbf{w}, w_0) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) + w_0$,. This can help with linear separability. More on this soon with neural networks.

3.2 Perceptron Algorithm

An important way to train a linear discriminant model is via the perceptron algorithm. This works for a two-class model. Rather than a $0/1$ error function (or sum-squared error), the perceptron algorithm adopts an alternative error function known as hinge loss, which uses the following function :

$$ReLU(z) = \begin{cases} z & z > 0 \\ 0 & o.w. \end{cases} = \max\{0, z\} \tag{14}$$

called a rectified linear activation (ReLU). This function is useful in the following way. Since $h() > 0$ for $y = 1$ and $h() < 0$ for $y = -1$, the product $-h(\mathbf{x}_i; \mathbf{w}, w_0)y_i > 0$ when there is a classification error. The perceptron loss function is defined as:

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} ReLU(-h(\mathbf{x}_i; \mathbf{w}, w_0)y_i) \tag{15}$$

$$= - \sum_{i=1:y_i \neq \hat{y}_i}^{n} (\mathbf{w}^\top \mathbf{x}_i + w_0)y_i \tag{16}$$

The first term takes the sum over all training examples of the ReLU function applied to $-h(\mathbf{x}_i; \mathbf{w}, w_0)y_i$. When there is a misclassified example, then this value is positive and it counts as a loss. Equivalently, we can simply write this as the negated sum over all misclassified examples of $h(\mathbf{x}_i; \mathbf{w}, w_0)y_i$.

This loss function has a gradient that is easier to work with than if we had used a $0/1$ error function, and we can now apply stochastic gradient descent. The change in weight vector from step t to $t+1$ is given by the following iteration on an incorrect example:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial}{\partial \mathbf{w}} \mathcal{L}^{(i)}(\mathbf{w}) = \mathbf{w}^{(t)} + \eta y_i \mathbf{x}_i, \tag{17}$$

where η is the learning rate parameter. Note that as the weight vector evolves during training, the set of examples that are misclassified will also change.

4 Practice Questions

1. **Posterior Weight Distribution By Completing the Square (Bishop 3.7)**

We know from (3.10) in Bishop that the likelihood can be written as

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{i=1}^{n} \mathcal{N}(y_i|\mathbf{w}^\top \mathbf{x}_i, \beta^{-1})$$
$$\propto \exp\left(-\frac{\beta}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})\right)$$

where precision $\beta = \frac{1}{\sigma^2}$ and in the second line above we have ignored the Gaussian normalization constants. By completing the square, show that with a prior distribution on $\mathbf{w}$ given by $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_0, \mathbf{S}_0)$, the posterior distribution $p(\mathbf{w}|D)$ is given by

$$p(\mathbf{w}|D) = \mathcal{N}(\mathbf{w}|\mathbf{m}_n, \mathbf{S}_n)$$

where

$$\mathbf{m}_n = \mathbf{S}_n(\mathbf{S}_0^{-1}\mathbf{m}_0 + \beta\mathbf{X}^\top\mathbf{y})$$
$$\mathbf{S}_n = \left(\mathbf{S}_0^{-1} + \beta\mathbf{X}^\top\mathbf{X}\right)^{-1}$$

Here's the first step. Take ln[(likelihood)(prior)] and collect normalization terms that don't depend on $\mathbf{w}$:

$$\ln p(\mathbf{w}|D) \propto \ln p(\mathbf{y}|\mathbf{X}, \mathbf{w}) + \ln p(\mathbf{w})$$
$$= const - \frac{\beta}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) - \frac{1}{2}(\mathbf{w} - \mathbf{m}_0)^\top\mathbf{S}_0^{-1}(\mathbf{w} - \mathbf{m}_0)$$

Hint: Remember, you already know what the posterior should look like. Once you simplify your expression enough, try foiling the posterior in terms of $\mathbf{m}_n$ and $\mathbf{S}_n^{-1}$ and see if you can see the relationship between your expression and this posterior.

Solution:

2. **Properties of Softmax**

Consider a K-class classification problem. Let $\{\mathbf{w}_k\}_{k=1}^K$ be defined such that for some data point $\mathbf{x}$, $z_k = \mathbf{w}_k^\top \mathbf{x}$ can be interpretted as a score for $\mathbf{x}$ belonging to class k. Multi-class Logistic Regression (LR) with a trained set of weights assigns $\mathbf{x}$ the class k for which it has the highest such score. The **softmax transformation** takes as input a vector, and outputs a transformed vector of the same size.

$$\text{softmax}(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{\ell=1}^K \exp(z_\ell)}, \text{ for all } k$$

LR uses the softmax over a vector of K scores $\mathbf{z} = [\mathbf{w}_1^\top \mathbf{x}, \ldots, \mathbf{w}_K^\top \mathbf{x}]$ so that it can be normalized and intepretted as a vector of *probabilities*.

$$p(\mathbf{y} = C_k \mid \mathbf{x}; \{\mathbf{w}_\ell\}_{\ell=1}^K) = \text{softmax}([\mathbf{w}_1^\top \mathbf{x} \ldots \mathbf{w}_K^\top \mathbf{x}]^\top)_k = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{\ell=1}^K \exp(\mathbf{w}_\ell^\top \mathbf{x})}.$$

where C_k is a *one-hot* vector with a 1 in coordinate k and 0s elsewhere.
Assuming data $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$, the negative log-likelihood can be written as:

$$\mathcal{L}(\{\mathbf{w}_\ell\}) = -\sum_{i=1}^N \ln p(\mathbf{y}_i \mid \mathbf{x}_i; \{\mathbf{w}_\ell\})$$

The softmax is an important function and you will see it again in other models, such as neural networks. In this problem, we aim to gain intuitions into the properties of softmax and multiclass logistic regression. In the section note solutions, we provide arguments for facts (a), (b), (c), and (d) (make sure you verify these on your own time). Using (d), show that (e) holds:

(a) The output of the softmax is a vector with non-negative components that are at most 1.
(b) The output of the softmax defines a distribution, so the components sum to 1.
(c) Softmax preserves order. This means that if elements $z_k < z_\ell$ in $\mathbf{z}$, then $\text{softmax}(\mathbf{z})_k < \text{softmax}(\mathbf{z})_\ell$ for any k, ℓ.
(d)
$$\frac{\partial \text{softmax}(\mathbf{z})_k}{\partial z_j} = \text{softmax}(\mathbf{z})_k (I_{kj} - \text{softmax}(\mathbf{z})_j) \text{ for any } k, j$$
where indicator $I_{kj} = 1$ if $k = j$ and $I_{kj} = 0$ otherwise.
(e) Using (d), show that:
$$\frac{\partial}{\partial \mathbf{w}_j} \mathcal{L}(\{\mathbf{w}_\ell\}) = \sum_{i=1}^N \left[p(\mathbf{y}_i = C_j | \mathbf{x}_i; \{\mathbf{w}_\ell\}) - y_{ij}\right] \mathbf{x}_i$$

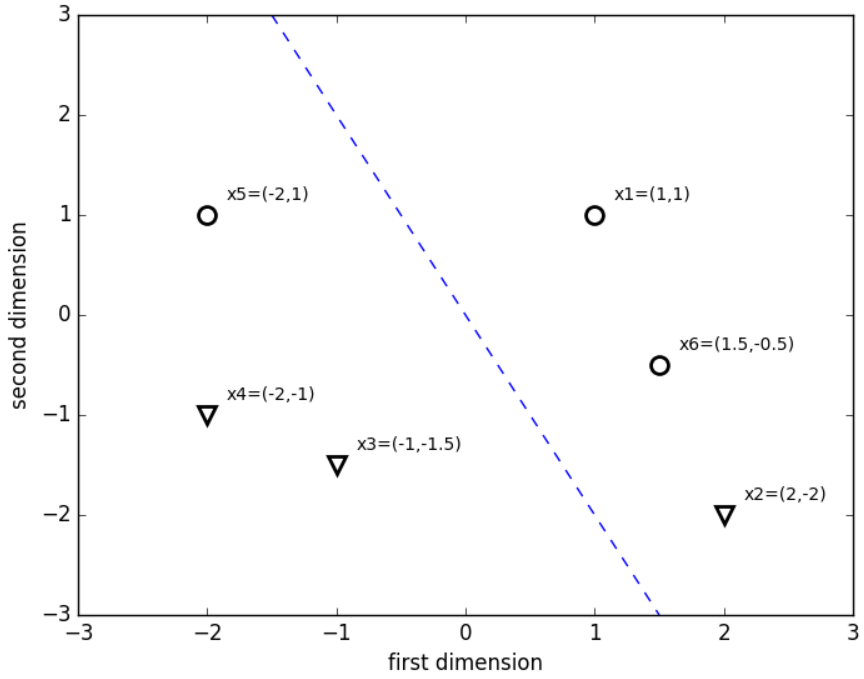
Solution:

3. **Small Perceptron Example**

Let's train a perceptron on a small data set. Consider data $\{\mathbf{x}_i\}_{i=1}^n, \mathbf{x}_i \in \mathbb{R}^2$. Let the learning rate $\eta = 0.2$ and let the weights be initialized as:

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}, w_0 = 0$$

Let the circles have $y_i = 1$ and the triangles $y_i = -1$. The data and initial separation boundary (determined by $\mathbf{w}$) is illustrated below.



Proceed by iterating over each example until there are no more classification errors. When in doubt, refer to the notes above. We know a priori that we will be able to train the classifier and have no classification errors because one can see visually that the data is linearly separable (note: as mentioned above, if the data were not so obviously linearly separable, a new basis could make it so). How many updates do you have to make? Is this surprising?

Solution: