1 Label Propagation

Consider a transductive learning setting where we have a set of labelled examples $y_i \in \{-1, +1\}$ for i ranging from 1 to n. We also have a set of t unlabeled examples that we would like to label. While we do not have features for any examples, we are given weights w_{ij} indicating how strongly we prefer unlabeled example i to have the same label as labeled example j, and another set of weights v_{ij} indicating how strongly we prefer unlabeled example i to have the same label as unlabeled example j. We'll assume that $v_{ij} = v_{ji}$ and $v_{ii} = 0$.

To find the labels of the unlabeled examples, a standard label propagation objective is

$$\underset{\hat{y}_1 \in \mathbb{R}, \hat{y}_2 \in \mathbb{R}, \ldots, \hat{y}_t \in R}{\operatorname{argmin}} \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{t} \left[w_{ij} (y_j - \hat{y}_i)^2 \right] + \frac{1}{2} \sum_{i=1}^{t} \sum_{j=i+1}^{t} \left[v_{ij} (\hat{y}_j - \hat{y}_i)^2 \right].$$

Although we can fit this model with gradient descent, a standard approach to fitting it is by cycling through each of the $\hat{y}_i$ and updating them to their optimal value given the values of the remaining $\hat{y}_j$ for $j \neq i$.

(a) Derive the partial derivative of this objective function with respect to a particular $\hat{y}_i$.

(b) Derive the optimal value of a particular $\hat{y}_i$, given the values of the remaining $\hat{y}_j$ for $j \neq i$.

Answer:

(a) We have that

$$\frac{\partial}{\partial \hat{y}_i} = \sum_{j=1}^{n}[-w_{ij}(y_j - \hat{y}_i)] + \sum_{j=1}^{t}[-v_{ij}(\hat{y}_j - \hat{y}_i)],$$

where we've used that $(\hat{y}_i - \hat{y}_i) = 0$ and the symmetry $v_{ij} = v_{ji}$.

(b) We notice that this is a one-dimensional leasts squares problem in disguise. Equating the partial derivative to zero and moving terms not depending on $\hat{y}_i$ to one side we have

$$\sum_{j=1}^{n}[w_{ij}\hat{y}_i] + \sum_{j=1}^{t}[v_{ij}\hat{y}_i] = \sum_{j=1}^{n}[w_{ij}y_j] + \sum_{j=1}^{t}[v_{ij}\hat{y}_j].$$

Taking $\hat{y}_i$ outside the sums on the right and then solving for it gives

$$\hat{y}_i = \frac{\sum_{j=1}^{n}[w_{ij}y_j] + \sum_{j=1}^{t}[v_{ij}\hat{y}_j]}{\sum_{j=1}^{n}[w_{ij}] + \sum_{j=1}^{t}[v_{ij}]}$$

(Basically, we take a weighted combination of our neighbours and normalize by the weights.)

2 Outlierness Ratio

In class we defined an 'outlierness' ratio of an example $x_i \in \mathbb{R}^d$ for $i = 1$ to n. This ratio depends on the k-nearest neighbours, $N_k(x_i)$, and the average distance to these k-nearest neighbours,

$$D_k(x_i) = \frac{1}{k} \sum_{j \in N_k(x_i)} \|x_i - x_j\|.$$

Given these definitions, the 'outlierness' ratio is defined by the quantity

$$O(x_i) = \frac{D_k(x_i)}{\frac{1}{k}\sum_{j \in N_k(x_i)} D_k(x_j)},$$

which roughly measures whether x_i is further away from its neighbours than its neighbours are from their neighbours.

(a) Give pseudo-code describing how to compute this ratio for a single example x_i (we'll assume that no points have the exact same distance from each other).

Answer:

1. Compute the distance of x_i to each other example and store this in a vector $d(j)$:

$$d(j) \leftarrow \|x_i - x_j\|.$$

Find the indices $N_k(x_i)$ of the k smallest elements of $d(j)$. One way to do this is to first find the kth smallest, then going through the elements to find all indices j where $d(j)$ is less than or equal to the kth smallest.

Now that we know the k-nearest neighours, compute the average distance of x_i to its k-nearest neighbours,

$$D_k(x_i) \leftarrow \frac{1}{k} \sum_{j \in N_k(x_i)} \|x_i - x_j\|.$$

2. Now that you know the k-nearest neighbours, repeat step 1 for each of the k-nearest neighbours in order to find $D_k(x_j)$ for each $j \in N_k(x_i)$.

3. With $D_k(x_i)$ and all the $D_k(x_j)$, compute the outlierness ratio,

$$O(x_i) = \frac{D_k(x_i)}{\frac{1}{k}\sum_{j \in N_k(x_j)} D_k(x_j)}.$$

3 Principal Component Analysis

Consider the following dataset, containing 5 examples with 2 features each:

x_1	x_2
-2	-2
-1	-1
0	0
1	1
2	2

(a) What is the first principal component?

(b) What is the (L2-norm) reconstruction error of the point (3,3)?

(c) What is the (L2-norm) reconstruction error of the point (3,4)?

Answer:

(a) First, we see that the mean of both variables is zero so we do not need to center them. Second, we see that all the variables lie along the $x_2 = x_1$. The direction of this line is $(1, 1)$, but since we normalize the principal components to have a distance of one the first PC is $W_1 = (1/\sqrt{(2)}, (1/\sqrt{(2)}))$ so that $\sqrt{W_1} = \sqrt{(1/\sqrt{2})^2 + (1/\sqrt{2})^2} = \sqrt{1/2 + 1/2} = 1$.

(b) To get the low-dimensional representation, we first subtract the means (which are zero) and then multiply by W_1,

$$z = (3 - 0)/\sqrt{2} + (3 - 0)/\sqrt{2} = 6/\sqrt{2}.$$

To go back to the original space, we multiply this by W_1 and add back the means:

$$\hat{x} = \frac{6}{\sqrt{2}}(1/\sqrt{2}, 1/\sqrt{2}) + (0, 0) = (6/2, 6/2) = (3, 3),$$

which is the same as the original point so the reconstruction error is 0.

(c)

$$z = (3 - 0)/\sqrt{2} + (4 - 0)/\sqrt{2} = 7/\sqrt{2}.$$

$$\hat{x} = \frac{7}{\sqrt{2}}(1/\sqrt{2}, 1/\sqrt{2}) + (0, 0) = (7/2, 7/2) = (3.5, 3.5),$$

so the reconstruction error is

$$\sqrt{(3.5 - 3)^2 + (3.5 - 4)^2} = \sqrt{1/4 + 1/4} = 1/\sqrt{2}.$$

4 Laplace Regression

Suppose we have a set of training examples (x_i, y_i) and we want to fit a linear model of the form $y_i \approx w^T x_i$. However, we do not want to use the squared error since we want to be robust to outliers in y_i. So instead we assume that the distribution of y_i follows a Laplace distribution with a mean of $w^T x_i$ and 'diversity' of b,

$$p(y_i|w^T x_i, b) = \frac{1}{2b} \exp\left(-\frac{|y_i - w^T x_i|}{b}\right),$$

where $b > 0$. We want to find the w that maximizes these probabilities assuming that our examples are IID,

$$\operatorname*{argmax}_{w \in \mathbb{R}^d} \prod_{i=1}^{n} p(y_i|w^T x_i, b).$$

Show how finding w corresponds to minimizing an additive loss function,

$$\operatorname*{\mathbf{argmin}}_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f(y_i, w_i^x),$$

and derive the form of this loss function (simplifying as much as possible).

Answer:

We can transform the product into a sum by taking the log, giving

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmax}} \sum_{i=1}^{n} \log p(y_i|w^T x_i, b),$$

and by taking the negative we get a minimization problem

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmin}} - \sum_{i=1}^{n} \log p(y_i|w^T x_i, b),$$

Plugging in the definition of p we get

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmin}} - \sum_{i=1}^{n} (-\log(2b) + \log\left(\exp\left(-\frac{|y_i - w^T x_i|}{b}\right)\right),$$

which can be simplified to

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} \left[\log(2b) + \frac{|y_i - w^T x_i|}{b}\right].$$

Notice that the first term does not depend on w so we can ignore it,

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} \frac{|y_i - w^T x_i|}{b},$$

and notice that we can take b outside the sum

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{b} \sum_{i=1}^{n} |y_i - w^T x_i|.$$

Since b is a positive constant, dividing by b does not change the location of the minimizer and we have

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} |y_i - w^T x_i|,$$

which is the absolute loss.

5 Stochastic Gradient

Using the logistic loss to fit a binary classifier corresponds to solving the optimization problem

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)).$$

Using $f(w)$ to denote the objective function, the gradient of the objective function can be written in the form

$$\nabla f(w) = \sum_{i=1}^{n} g(x_i, w) x_i,$$

for a function g that returns a scalar given the training example x_i and parameter vector w. The cost of computing g is $O(m)$ if x_i has m non-zero values, since it requires multiplying each non-zero element of x_i by the corresponding element of w, so in the worst case computing g costs $O(d)$.

(a) Write pseudo-code doing an iteration of gradient descent on this model with a constant step-size α. What is the cost of performing an iteration of gradient descent in terms of n and d.

(b) Write pseudo-code doing an iteration of stochastic gradient on this model with a constant step-size α. What is the cost of performing an iteration of stochastic gradient in terms of n and d? (You can assume that generating a random number between 1 and n costs $O(1)$.)

(c) How does the cost per iteration in parts (a) and (b) change if each x_i has at most m non-zeroes?

Answer:

(a)

First use the provided provided routine to compute all $g(x_i, w^t)$ from $i = 1$ to n.

Next, use these to compute the gradient,

$$\nabla f(w^t) = \sum_{i=1}^{n} g(x_i, w^t) x_i.$$

Finally, use the gradient to update the parameter based on the constant step-size α,

$$w^{t+1} = w^t - \alpha \nabla f(w^t).$$

The first step requires n calls to g which costs $O(d)$ in the worst case, so it costs $O(nd)$. The second step involves n scalar multiplications (each at a cost of $O(d)$) and n additions (each at a cost of $O(d)$) giving a cost of $O(nd)$. The third step performs a fixed number of scalar multiplications/additions since its cost is $O(d)$, giving a total cost of $O(nd)$.

(b)

First, generate a random integer i between 1 and n.

Next, compute $g(x_i, w^t)$ for the random example i.

Update the parameters based on the gradient of that particular example,

$$w^{t+1} = w^t - \alpha g(x_i, w^t) x_i.$$

The first step costs $O(1)$ by the assumption in the question. The second step costs $O(d)$ in the worst case, and updating the parameter costs $O(d)$ too. This gives a total cost of $O(d)$.

(c)

Gradient descent: Since the cost of computing $g(x_i, w^t)$ is now $O(m)$, the cost of the first step is reduced to $O(mn)$. The cost of computing the gradient is also reduced to $O(nm)$ since each operation only needs to be applied based on the m non-zero values. Since $\nabla f(w^t)$ may be a dense vector, the cost of the third step remains $O(d)$. This gives a total cost of $O(mn + d)$.

Stochastic gradient: The cost of computing $g(x_i, w^t)$ is now $O(m)$. Since we only need to update the elements of w^t corresponding to non-zero values of x_i, the cost of the update is also reduced to $O(m)$. This gives a total cost of $O(m)$.