Statistical Machine Learning GU4241/GR5241
Spring 2017
https://courseworks.columbia.edu/

Linxi Liu
ll3098@columbia.edu
Shuaiwen Wang
sw2853@columbia.edu
Haolei Weng
hw2375@columbia.edu

Homework 6

Due: Thursday, April. 27th, 2017

Homework submission: We will collect your homework **at the beginning of class** on the due date. You also need to submit **your code** for all problems through Canvas. If you cannot attend class that day, you can leave your solution in the homework dropbox 904 at the Department of Statistics, 9th floor SSW, at any time before then.

Problem 1 (Implementing PageRank, 20 points)

In this problem, you will implement the PageRank algorithm. You will be experimenting with a small randomly generated graph (assume graph has no dead ends) which can be downloaded on Canvas.

It has $n = 100$ nodes (numbered $1, 2, \ldots, 100$), and $m = 1024$ edges, 100 of which form a directed cycle (through all the nodes) which ensures that the graph is connected. There may be multiple edges between a pair of nodes, your code should handle these instead of ignoring them. The first column in `graph.txt` refers to the source node, and the second column refers to the destination node.

Assume the directed graph $G = (V, E)$ has n nodes (numbered $1, 2, \ldots, n$) and m edges, all nodes have positive out-degree, and $A = (A_{ji})_{n\times n}$ is an $n \times n$ matrix as defined in class such that for any $i, j \in \{1, 2, \ldots, n\}$:

$$A_{ji} = \begin{cases} \frac{1}{\deg(i)} & \text{if } (i \to j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Here, $\deg(i)$ is the number of outgoing edges of node i in G. Denote the PageRank vector by the column vector r. By the definition of PageRank, we have the following equation:

$$r = \frac{\alpha}{n}\mathbf{1} + (1-\alpha)Ar, \tag{1}$$

where $\mathbf{1}$ is the vector of length n with all entries equal to 1.

Based on this equation, the iterative procedure to compute PageRank works as follows:

- Initialize: $r^{(0)} = \frac{1}{n}\mathbf{1}$.
- For i from 1 to k, iterate: $r^{(i)} = \frac{\alpha}{n}\mathbf{1} + (1-\alpha)Ar^{(i-1)}$.

Run the aforementioned iterative process for 40 iterations, assume $\alpha = 0.2$ and obtain the PageRank vector r. Compute the following:

- List the top 5 nodes ids with the highest PageRank scores (submit both ids and scores).
- List the bottom 5 node ids with the lowest PageRank scores (submit both ids and scores).

Problem 2 (Bayesian inference, 20 points)

Suppose observations $X_1, X_2, \ldots$ are recorded. We assume these to be conditionally independent and exponentially distributed given a parameter θ:

$$X_i \sim \text{Exponential}(\theta),$$

for all $i = 1, \ldots, n$. The exponential distribution is controlled by one *rate parameter* $\theta > 0$, and its density is

$$p(x; \theta) = \theta e^{-\theta x}$$

for $x \in \mathbb{R}_+$.

1. Plot the graph of $p(x; \theta)$ for $\theta = 1$ in the interval $x \in [0, 4]$.
2. What is the visual representation of the likelihood of individual data points? Draw it on the graph above for the samples in a toy dataset $\mathcal{X} = \{1, 2, 4\}$ and $\theta = 1$.
3. Would a higher rate (e.g. $\theta = 2$) increase or decrease the likelihood of each sample in this toy data set?

We introduce a prior distribution $q(\theta)$ for the parameter. Our objective is to compute the posterior. In general, that requires computation of the evidence as the integral

$$p(x_1, \ldots, x_n) = \int_{\mathbb{R}_+} \Big(\prod_{i=1}^{n} p(x_i|\theta) \Big) q(\theta) d\theta \ .$$

We will not have to compute the integral in the following, since we choose a prior that is conjugate to the exponential.

The natural conjugate prior for the exponential distribution is the gamma distribution:

$$q(\theta|\alpha, \beta) = \theta^{\alpha-1} \frac{\beta^\alpha e^{-\beta\theta}}{\Gamma(\alpha)}$$

for $\theta \geq 0$ and $\alpha, \beta > 0$. We have already encountered this distribution in an earlier homework problem (where we computed its maximum likelihood estimator), and you will notice that we are using a different parametrization of the gamma density here.

Question 1. Take a moment to convince yourself that the exponential and gamma distributions are exponential family models. Show that, if the data is exponentially distributed as above with a gamma prior

$$q(\theta) = \text{Gamma}(\alpha_0, \beta_0) \ ,$$

the posterior is again a gamma, and find the formula for the posterior parameters. (In other words, adapt the computation we performed in class for general exponential families to the specific case of the exponential/gamma model.) In detail:

- Ignore multiplicative constants and normalization terms, such as the evidence term in Bayes' formula.
- Show that the posterior is proportional to a gamma distribution.
- Deduce the parameters by comparing your result for the posterior to the definition of the gamma distribution.

Machine learning problems are often *online problems*, where each data point has to be processed immediately when it is recorded (as opposed to *batch problems*, where the entire data set is recorded first and then processed as a whole). Conjugate priors are particularly useful for online problems, since, roughly speaking, the posterior given the first $(n-1)$ observations can be used as a prior for processing the nth observation:

Question 2.

a. Show that, if $p(x|\theta)$ is an exponential family model and $q(\theta)$ its natural conjugate prior, the posterior $\Pi(\theta|x_{1:n})$ under n observations can be computed as the posterior given a single observation x_n using the prior $\tilde{q}(\theta) := \Pi(\theta|x_{1:n-1})$.

b. For the specific case of the exponential/gamma model, give the formula for the parameters (α_n, β_n) of the posterior $\Pi(\theta|x_{1:n}, \alpha_0, \beta_0)$ as a function of $(\alpha_{n-1}, \beta_{n-1})$.

c. Visualize the gradual change of shape of the posterior $\Pi(\theta|x_{1:n}, \alpha_0, \beta_0)$ with increasing n:

- Generate $n = 256$ exponentially distributed samples with parameter $\theta = 1$.
- Use the values $\alpha_0 = 2, \beta_0 = 0.2$ for the hyperparameters of the prior.
- Visualize the updated posterior distribution after $n = \{4, 8, 16, 256\}$, in the range $\theta \in [0, 4]$. Plot all curves into the same figure and label each curve.
  Hint: The gamma function Γ, which occurs in the definition of the gamma density, is implemented in R as `gamma`. When you have to compute a product over several data points, you might run into numerical problems with this function. One possible workaround to first compute the log-likelihood and then take its exponential $\exp(\log(p(x_{1:n}; \alpha, \beta)))$. The logarithm of the gamma function is implemented in R as a separate function `lgamma`.
- Comment on the behavior of the posterior distribution as n increases.

Problem 3 (Implementation of SVM via Gradient Descent, 6 extra points)

In the problem, you will implement the soft margin SVM using different gradient descent methods. You can use either R or Python for this problem. Our training data consists of n pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$. Define a hyperplane by

$$\{x : f(x) = x^T \mathbf{w} + b = 0\}.$$

A classification rule induced by $f(x)$ is

$$G(x) = \text{sign}(x^T \mathbf{w} + b).$$

To recap, to estimate the $\mathbf{w}$, b of the soft margin SVM, we can minimize the cost:

$$f(\mathbf{w}, b) = \frac{1}{2} \sum_{j=1}^{d} (w^{(j)})^2 + C \sum_{i=1}^{n} \max \left\{ 0, 1 - y_i \left(\sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b \right) \right\}. \tag{2}$$

Define $L(\mathbf{w}, b; x_i, y_i) = \max\{0, 1 - y_i(\sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b)\}$. In order to minimize the cost function, we first obtain the gradient with respect to $w^{(j)}$, the jth item in the vector $\mathbf{w}$, and b as follows:

$$\begin{aligned} \nabla_{w^{(j)}} f(\mathbf{w}, b) &= \frac{\partial f(\mathbf{w}, b)}{\partial w^{(j)}} = w_j + C \sum_{i=1}^{n} \frac{\partial L(\mathbf{w}, b; x_i, y_i)}{\partial w^{(j)}}, \\ \nabla_b f(\mathbf{w}, b) &= \frac{\partial f(\mathbf{w}, b)}{\partial b} = C \sum_{i=1}^{n} \frac{\partial L(\mathbf{w}, b; x_i, y_i)}{\partial b}, \end{aligned} \tag{3}$$

where

$$\begin{aligned} \frac{\partial L(\mathbf{w}, b; x_i, y_i)}{\partial w^{(j)}} &= \begin{cases} 0 & \text{if } y_i(x_i^T \mathbf{w} + b) \geq 1 \\ -y_i x_i^{(j)} & \text{otherwise.} \end{cases} \\ \frac{\partial L(\mathbf{w}, b; x_i, y_i)}{\partial b} &= \begin{cases} 0 & \text{if } y_i(x_i^T \mathbf{w} + b) \geq 1 \\ -y_i & \text{otherwise.} \end{cases} \end{aligned}$$

Now, we will implement and compare the following gradient descent techniques:

- **Batch gradient descent:** Iterate through the entire dataset and update the parameters as follows:

  $k = 0$

while convergence criteria not reached **do**
  for $j = 1, \ldots, d$ **do**
    Update $w^{(j)} \leftarrow w^{(j)} - \eta \nabla_{w^{(j)}} f(\mathbf{w}, b)$
  end for
  Update $b \leftarrow b - \eta \nabla_b f(\mathbf{w}, b)$
  Update $k \leftarrow k + 1$
end while

where,
n is the number of samples in the training data,
d is the dimensions of $\mathbf{w}$,
η is the learning rate of the gradient descent, and
$\nabla_{w^{(j)}} f(\mathbf{w}, b)$ and $\nabla_b f(\mathbf{w}, b)$ are the values compuated from equation (3).

The *convergence criteria* for the above algorithm is $\Delta_{\%cost} < \epsilon$, where

$$\Delta_{\%cost} = \frac{|f_{k-1}(\mathbf{w}, b) - f_k(\mathbf{w}, b)| \times 100}{f_{k-1}(\mathbf{w}, b)}. \tag{4}$$

Here,
$f_k(\mathbf{w}, b)$ is the value of equation (2) at kth iteration,
$\Delta_{\%cost}$ is computed at the end of each iteration of the while loop.
Initialize $\mathbf{w} = 0$, $b = 0$ and compute $f_0(\mathbf{w}, b)$ with these values.
For this method, use $\eta = 0.0000003$, $\epsilon = 0.25$.

- **Stochastic gradient descent:** Go through the dataset and update the parameters, one training sample at a time, as follows:

  Randomly shuffle the training data
  $i = 1$, $k = 0$
  while convergence criteria not reached **do**
    for $j = 1, \ldots, d$ **do**
      Update $w^{(j)} \leftarrow w^{(j)} - \eta \nabla_{w^{(j)}} f_i(\mathbf{w}, b)$
    end for
    Update $b \leftarrow b - \eta \nabla_b f_i(\mathbf{w}, b)$
    Update $i \leftarrow (i \bmod n) + 1$
    Update $k \leftarrow k + 1$
  end while

  where,
  n is the number of samples in the training data,
  d is the dimension of $\mathbf{w}$,
  η is the learning rate and
  $\nabla_{w^{(j)}} f_i(\mathbf{w}, b)$ is defined for a single training sample as follows:

$$\nabla_{w^{(j)}} f_i(\mathbf{w}, b) = \frac{\partial f_i(\mathbf{w}, b)}{\partial w^{(j)}} = w_j + C \frac{\partial L(\mathbf{w}, b; x_i, y_i)}{\partial w^{(j)}}$$

  $\nabla_b f_i(\mathbf{w}, b)$ is similar.

  The *convergence criteria* here is $\Delta_{cost}^{(k)} < \epsilon$, where

$$\Delta_{cost}^{(k)} = 0.5 \Delta_{cost}^{(k-1)} + 0.5 \Delta_{\%cost},$$

  where,
  k is the iteration number, and
  $\Delta_{\%cost}$ is the same as above (from equation 4).
  Calcuate Δ_{cost}, $\Delta_{\%cost}$ at the end of each iteration of the while loop.
  Initialize $\Delta_{cost} = 0$, $\mathbf{w} = 0$, $b = 0$ and compute $f_0(\mathbf{w}, b)$ with these values.
  For this method, use $\eta = 0.0001, \epsilon = 0.001$.

- **Mini batch gradient descent:** Go through the dataset in batches of predetermined size and update the parameters, one training sample at a time, as follows:

  Randomly shuffle the training data
  $l = 1$, $k = 0$
  while convergence criteria not reached **do**
    for $j = 1, \ldots, d$ **do**
      Update $w^{(j)} \leftarrow w^{(j)} - \eta \nabla_{w^{(j)}} f_l(\mathbf{w}, b)$
    end for
    Update $b \leftarrow b - \eta \nabla_b f_l(\mathbf{w}, b)$
    Update $l \leftarrow (l + 1)$ mod $((n + batch\_size - 1)/batch\_size)$
    Update $k \leftarrow k + 1$
  end while

  where,
  n is the number of samples in the training data,
  d is the dimension of $\mathbf{w}$,
  η is the learning rate and
  batch_size is the number of training samples considered in each batch, and $\nabla_{w^{(j)}} f_l(\mathbf{w}, b)$ is defined for a batch of training sample as follows:

$$\nabla_{w^{(j)}} f_l(\mathbf{w}, b) = \frac{\partial f_l(\mathbf{w}, b)}{\partial w^{(j)}} = w_j + C \sum_{i=l*batch\_size+1}^{\min\{n,(l+1)*batch\_size\}} \frac{\partial L(\mathbf{w}, b; x_i, y_i)}{\partial w^{(j)}}$$

  The *convergence criteria* here is $\Delta_{cost}^{(k)} < \epsilon$, where

$$\Delta_{cost}^{(k)} = 0.5\Delta_{cost}^{(k-1)} + 0.5\Delta_{\%cost},$$

  where,
  k is the iteration number, and
  $\Delta_{\%cost}$ is the same as above (equation 4).
  Calcuate Δ_{cost}, $\Delta_{\%cost}$ at the end of each iteration of the while loop.
  Initialize $\Delta_{cost} = 0$, $\mathbf{w} = 0$, $b = 0$ and compute $f_0(\mathbf{w}, b)$ with these values.
  For this method, use $\eta = 0.00001, \epsilon = 0.01$, batch_size = 20.

Homework problems: Implement the SVM algorithm for all the the above mentioned gradient descent techniques.

Use $C = 100$ for all the techniques. For all other parameters, use the values specified in the description of the technique. **Note:** update w in iteration $k + 1$ using the values computed in iteration k. Do not update using values computed in the current iteration!

Run you implementation on the data set under the folder `HW6_SVM`. The data set contains the following files:

1. `features.txt`: Each line contains features (comma-separated values) for a single datapoint. It have 6414 datapoints (rows) and 122 features (columns).
2. `target.txt`: Each line contains the response variable (y = -1 or 1) for the corresponding row in `features.txt`.

Plot the value of the cost function $f(\mathbf{w}, b)$ after each iteration vs. the number of iteration (k). Report the total time taken for convergence by each of the gradient descent techniques. What do you infer from the plots and the time for convergence?

The diagram should have graphs from all the three techniques on the same plot.