CS246: Mining Massive Data Sets **Winter 2016**

Problem Set 4

Due 11:59pm March 3, 2016

Only one late period is allowed for this homework (11:59pm 3/8).

General Instructions

Submission instructions: These questions require thought but do not require long answers. Please be as concise as possible. You should submit your answers as a writeup in PDF format via GradeScope and code via the Snap submission site.

Submitting writeup: Prepare answers to the homework questions into a single PDF file and submit it via http://gradescope.com. Make sure that the answer to each question is on a *separate page*. On top of each page write the number of the question you are answering. Please find the cover sheet and the recommended templates located here:
http://web.stanford.edu/class/cs246/homeworks/hw4/submission_template_hw4.tex
http://web.stanford.edu/class/cs246/homeworks/hw4/submission_template_hw4.pdf
http://web.stanford.edu/class/cs246/homeworks/hw4/submission_template_hw4.docx

Not including the cover sheet in your submission will result in a 2-point penalty. It is also important to tag your answers correctly on Gradescope. We will deduct 5/N points for each incorrectly tagged subproblem (where N is the number of subproblems). This means you can lose up to 5 points for incorrect tagging.

Submitting code: Upload your code at http://snap.stanford.edu/submit. Put all the code for a single question into a single file and upload it.

Questions

1 Perceptron Learning and Linear Separability (20 points)

You are given information about five patients who are being tested for diabetes. The results of Fasting Plasma Glucose (FPG) test and the Oral Glucose Tolerance (OGT) test are provided for each of these patients. The outcome variable indicating if each patient has been diagnosed with diabetes (+1) or not (-1) is also available in the data.

We already generated features for each patient x as follows:

- $\phi_1(x) = 1$ if FPG test is positive, otherwise it is set to 0.
- $\phi_2(x) = 1$ if OGT test is positive, otherwise it is set to 0.
- $\phi_3(x) = -1$, a bias term

Given a weight vector $w = (w1, w2, w3)$, our classifier returns $+1$ if $w_1\phi_1(x) + w_2\phi_2(x) + w_3\phi_3(x) > 0$ and -1 otherwise.

Our training set comprises of the following features and labels:

PatientID	ϕ_1	ϕ_2	ϕ_3	Label
1	0	0	-1	+1
2	1	1	-1	+1
3	1	1	-1	+1
4	0	1	-1	-1
5	1	0	-1	-1

Table 1: Diabetes Dataset

(a) [6 Points]

Compute the first four updates of the Perceptron training algorithm [Slides 9-11 of the lecture slide deck machine-learning-2] using the diabetes data provided in Table 1. Fill in the following table, using the given intial Perceptron weights $w = (w_1, w_2, w_3) = (0, 0, 0)$, $\eta = 1/5$.

w	w_1	w_2	w_3
Initial	0	0	0
After Observing PatientID = 1			
After Observing PatientID = 2			
After Observing PatientID = 3			
After Observing PatientID = 4			

(b) [4 Points]

Will the Perceptron algorithm return a solution for the diabetes dataset shown in Table 1? Why?

(c) [10 Points]

Linear classifiers such as Perceptrons are often insufficient to represent a dataset using a given set of features. However, it is often possible to find new features using nonlinear functions of our existing features which do allow linear classifiers to separate the data. Nonlinear features result in more expressive linear classifiers. For example, consider the following data set, where +s represent positive examples and −s represent negative examples.

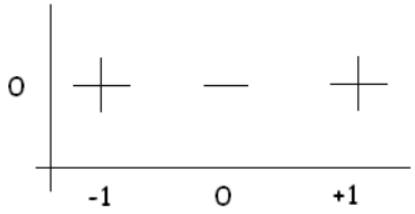

Figure 1: The original data points are not linearly separable

Let x_1 and x_2 denote the two dimensions of the data shown in Figure 1. No linear classifier can separate the positive examples $(-1, 0)$ and $(1, 0)$ from the negative example $(0, 0)$ in the dataset shown in Figure 1. Rather than using a single feature, if we perform a nonlinear mapping or transformation $\psi = ({x_1}^2, x_2+1)$, the positive examples are both mapped to $(1, 1)$ and the negative example is mapped to $(0, 1)$, and we see the data is now linearly separable (Figure 2).

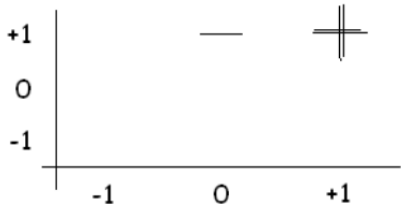

Figure 2: After the non-linear transformation the data becomes linearly separable

Which of the following transformations linearly separate the diabetes dataset shown in Table 1? Justify your answer about each transformation briefly. If a particular transformation linearly separates the data, also provide the value of w that separates the two classes in the data?

(i) $\psi = (\phi_1^2, \phi_1\phi_2, -1)$

(ii) $\psi = ((\phi_1 \text{ xor } \phi_2), \phi_2, -1)$ where a xor b is 1 if either $a = 1$ or $b = 1$ but not both.

(iii) $\psi = (\phi_1 - \phi_2, \phi_1 + \phi_2, -1)$

What to submit

(i) Table with entries filled in for w_1, w_2 and w_3. [1(a)]

(ii) Answer with explanation. [1(b)]

(iii) For each transformation, answer if the transformation makes the data linearly separable. Provide brief justifications for your answers. Also, give the value of w for those transformations which linearly separate the data. [1(c)]

2 Implementation of SVM via Gradient Descent (30 points)

Here, you will implement the soft margin SVM using different gradient descent methods as described in the section 12.3.4 of the textbook. To recap, to estimate the $\mathbf{w}, b$ of the soft margin SVM, we can minimize the cost:

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

In order to minimize the function, we first obtain the gradient with respect to $w^{(j)}$, the jth item in the vector $\mathbf{w}$, as follows.

$$\nabla_{w^{(j)}} f(\mathbf{w}, b) = \frac{\partial f(\mathbf{w}, b)}{\partial w^{(j)}} = w_j + C\sum_{i=1}^{n}\frac{\partial L(x_i, y_i)}{\partial w^{(j)}}, \quad (2)$$

where:

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

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

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

   k = 0
   while convergence criteria not reached **do**
     for $j = 1, ..., d$ **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
$\nabla_{w^{(j)}} f(\mathbf{w}, b)$ is the value computed from computing equation (2) above and $\nabla_b f(\mathbf{w}, b)$ is the value computed from your answer in question (a) below.

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

$$\Delta_{\%cost} = \frac{|f_{k-1}(\mathbf{w}, b) - f_k(\mathbf{w}, b)| \times 100}{f_{k-1}(\mathbf{w}, b)}. \tag{3}$$

where,
$f_k(\mathbf{w}, b)$ is the value of equation (1) at kth iteration,
$\Delta_{\%cost}$ is computed at the end of each iteration of the while loop.
Initialize $\mathbf{w} = \mathbf{0}, b = 0$ and compute $f_0(\mathbf{w}, b)$ with these values.
For this method, use $\eta = \mathbf{0.0000003}, \epsilon = \mathbf{0.25}$

2. **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 $j = 1, ..., d$ **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 $i \leftarrow (i \mod n) + 1$
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 it the learning rate and
$\nabla_{w^{(j)}} f_i(\mathbf{w}, b)$ is defined for a single training sample as follows:

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

(Note that you will also have to derive $\nabla_b f_i(\mathbf{w}, b)$, but it should be similar to your solution to question (a) below.
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 = iteration number, and
$\Delta_{\%cost}$ is same as above (equation 3).

Calculate $\Delta_{cost}, \Delta_{\%cost}$ at the end of each iteration of the while loop.
Initialize $\Delta_{cost} = 0$, $\mathbf{w} = \mathbf{0}, b = 0$ and compute $f_0(\mathbf{w}, b)$ with these values.
For this method, use $\eta = 0.0001, \epsilon = 0.001$.

3. **Mini batch gradient descent**: Go through the dataset in batches of predetermined size and update the parameters as follows:

   Randomly shuffle the training data
   $l = 0, k = 0$
   while convergence criteria not reached **do**
     for $j = 1, ..., d$ **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 dimensions of $\mathbf{w}$,
   η is the learning rate,
   $batch\_size$ 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 samples as follows:

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

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

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

   k = iteration number,
   and $\Delta_{\%cost}$ is same as above (equation 3).

   Calculate $\Delta_{cost}, \Delta_{\%cost}$ at the end of each iteration of the while loop.
   Initialize $\Delta_{cost} = 0$, $\mathbf{w} = \mathbf{0}, b = 0$ and compute $f_0(\mathbf{w}, b)$ with these values.
   For this method, use $\eta = 0.00001, \epsilon = 0.01$, batch_size = 20.

(a) [5 Points]

Notice that we have not given you the equation for, $\nabla_b f(\mathbf{w}, b)$.

Task: What is $\nabla_b f(\mathbf{w}, b)$ used for the Batch Gradient Descent Algorithm?

(Hint: It should be very similar to $\nabla_{w^{(j)}} f(\mathbf{w}, b)$.)

(b) [25 Points]

Task: Implement the SVM algorithm for all of the above mentioned gradient descent techniques in any choice of programming language you prefer.

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 $i + 1$ using the values computed in iteration i. Do not update using values computed in the current iteration!

Run your implementation on the data set at http://snap.stanford.edu/class/cs246-data/HW4-q1.zip. The data set contains the following files :

1. `features.txt` : Each line contains features (comma-separated values) for a single datapoint. It has 6414 datapoints (rows) and 122 features (columns).
2. `target.txt` : Each line contains the target variable (y = -1 or 1) for the corresponding row in `features.txt`.
3. `features.train.txt` : Each line contains features (comma-separated values) for a single datapoint. It has 6000 datapoints (rows) and 122 features (columns).
4. `target.train.txt` : Each line contains the target variable (y vector) for the corresponding row in `features.train.txt`.
5. `features.test.txt` : Each line contains features (comma-separated values) for a single datapoint. It has 414 datapoints (rows) and 122 features (columns).
6. `target.test.txt` : Each line contains the target variable (y vector) for the corresponding row in `features.test.txt`.

Use `features.txt` and `target.txt` as inputs for this problem.

Task: Plot the value of the cost function $f_k(\mathbf{w}, b)$ vs. the number of iterations (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.

As a sanity check, Batch GD should converge in 10-300 iterations and SGD between 500-3000 iterations with Mini Batch GD somewhere in-between. However, the number of iterations may vary greatly due to randomness. If your implementation consistently takes longer though, you may have a bug.

What to submit

(i) Equation for $\nabla_b f(\mathbf{w}, b)$. [2(a)]

(ii) Plot of $f_k(\mathbf{w}, b)$ vs. the number of updates (k). Interpretation of plot and convergence times. [2(b)]

(iii) Submit the code on snap submission website. [2(b)]

3 Decision Tree Learning (20 points)

In this problem, we want to construct a decision tree to find out if a person will enjoy beer.

Definitions. Let there be k binary-valued attributes in the data.

We pick an attribute that maximizes the gain at each node:

$$G = \max[I(D) - (I(D_L) + I(D_R))]; \tag{4}$$

where D is the given dataset, and D_L and D_R are the sets on left and right hand-side branches after division. Ties may be broken arbitrarily.

We define $I(D)$ as follows[1]:

$$I(D) = |D| \times \left(1 - \sum_i p_i^2\right),$$

where:

- $|D|$ is the number of items in D;
- $1 - \sum_i p_i^2$ is the gini index;
- p_i is the probability distribution of the items in D, or in other words, p_i is the fraction of items that take value $i \in \{+, -\}$. Put differently, p_+ is the fraction of positive items and p_- is the fraction of negative items in D.

Note that this intuitively has the feel that the more evenly-distributed the numbers are, the lower the $\sum_i p_i^2$, and the larger the impurity.

(a) [10 Points]

Let $k = 3$. We have three binary attributes that we could use: "likes wine", "likes running" and "likes pizza". Suppose the following:

- There are 100 people in sample set, 60 of whom like beer and 40 who don't.

[1] As an example, if D has 10 items, with 4 positive items (*i.e.* 4 people who enjoy beer), and 6 negative items (*i.e.* 6 who do not), we have $I(D) = 10 \times (1 - (0.16 + 0.36))$.

- Out of the 100 people, 50 like wine; out of those 50 people who like wine, 30 like beer.
- Out of the 100 people, 30 like running; out of those 30 people who like running, 20 like beer.
- Out of the 100 people, 80 like pizza; out of those 80 people who like pizza, 50 like beer.

Task: What are the values of G (defined in Equation 4) for wine, running and pizza attributes? Which attribute would you use to split the data at the root if you were to maximize the gain G using the gini index metric defined above?

(b) [10 Points]

Let's consider the following example:

- There are 100 attributes with binary values $a_1, a_2, a_3, \ldots, a_{100}$.
- Let there be one example corresponding to each possible assignment of 0's and 1's to the values $a_1, a_2, a_3 \ldots, a_{100}$. (Note that this gives us 2^{100} training examples.)
- Let the values taken by the target variable y depend on the values of a_1 for 99% of the datapoints. More specifically, of all the datapoints where $a_1 = 1$, let 99% of them are labeled $+$. Similarly, of all the datapoints where $a_1 = 0$, let 99% of them are labeled with $-$.
- Assume that we build a complete binary decision tree (*i.e.*, we use values of all attributes).

Task: In this case, explain what the decision tree will look like. (A one line explanation will suffice.) Also, identify what the desired decision tree for this situation should look like to avoid overfitting, and why.

What to submit

(i) Values of G for wine, running and pizza attributes. [3(a)]

(ii) The attribute you would use for splitting the data at the root. [3(a)]

(iii) Explain what the decision tree looks like in the described setting. Explain how a decision tree should look like to avoid overfitting. (1-2 lines each) [3(b)]

4 Data Streams (30 points)

In this problem, we study an approach to approximating the frequency of occurrences of different items in a data stream. Assume $S = \langle a_1, a_2, \ldots, a_t \rangle$ is a data stream of items from the set $\{1, 2, \ldots, n\}$. Assume for any $1 \leq i \leq n$, $F[i]$ is the number of times i has appeared in S. We would like to have good approximations of the values $F[i]$ $(1 \leq i \leq n)$ at all times.

A simple way to do this is to just keep the counts for each item $1 \leq i \leq n$ separately. However, this will require $\mathcal{O}(n)$ space, and in many applications (e.g., think online advertising and counts of user's clicks on ads) this can be prohibitively large. We see in this problem that it is possible to approximate these counts using a much smaller amount of space. To do so, we consider the algorithm explained below.

Strategy. The algorithm has two parameters $\delta, \epsilon > 0$. It picks $\lceil \log \frac{1}{\delta} \rceil$ independent hash functions:

$$\forall j \in \left[\!\left[1; \left\lceil \log \frac{1}{\delta} \right\rceil \right]\!\right], \quad h_j : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, \lceil \frac{e}{\epsilon} \rceil\},$$

where log denotes natural logarithm. Also, it associates a count $c_{j,x}$ to any $1 \leq j \leq \lceil \log \frac{1}{\delta} \rceil$ and $1 \leq x \leq \lceil \frac{e}{\epsilon} \rceil$. In the beginning of the stream, all these counts are initialized to 0. Then, upon arrival of each a_k $(1 \leq k \leq t)$, each of the counts $c_{j,h_j(a_k)}$ $(1 \leq j \leq \lceil \log \frac{1}{\delta} \rceil)$ is incremented by 1.

For any $1 \leq i \leq n$, we define $\tilde{F}[i] = \min_j \{c_{j,h_j(i)}\}$. We will show that $\tilde{F}[i]$ provides a good approximation to $F[i]$.

Memory cost. Note that this algorithm only uses $\mathcal{O}\left(\frac{1}{\epsilon} \log \frac{1}{\delta}\right)$ space.

Properties. Few important properties of the algorithm presented above:

- For any $1 \leq i \leq n$:
$$\tilde{F}[i] \geq F[i].$$
- For any $1 \leq i \leq n$ and $1 \leq j \leq \lceil \log(\frac{1}{\delta}) \rceil$:
$$\mathsf{E}\left[c_{j,h_j(i)}\right] \leq F[i] + \frac{\epsilon}{e}(t - F[i]).$$

(a) [10 Points]

Prove that:

$$\mathsf{Pr}\left[\tilde{F}[i] \leq F[i] + \epsilon t\right] \geq 1 - \delta.$$

Hint: Use Markov inequality and the property of independence of hash functions.

Based on the proof in part (a) and the properties presented earlier, it can be inferred that $\tilde{F}[i]$ is a good approximation of $F[i]$ for any item i such that $F[i]$ is not very small (compared to t). In many applications (*e.g.*, when the values $F[i]$ have a heavy-tail distribution), we are indeed only interested in approximating the frequencies for items which are not too infrequent. We next consider one such application.

(b) [20 Points]

Warning. This implementation question requires substantial computation time - python / java / c(++) implementations will be faster. (Python implementation reported to take 15min - 1 hour). In any case, we advise you to start early.

Dataset. http://snap.stanford.edu/class/cs246-data/HW4-q4.zip The dataset contains the following files:

1. `words_stream.txt` Each line of this file is a number, corresponding to the ID of a word in the stream.
2. `counts.txt` Each line is a pair of numbers separated by a tab. First number is an ID word and the second number is its associated exact frequency count in the stream.
3. `words_stream_tiny.txt` and `counts_tiny.txt` are smaller versions of the dataset above that you can use for debugging your implementation.
4. `hash_params.txt` Each line is a pair of numbers separated by a tab, corresponding to parameters a and b which you may use to define your own hash functions (See explanation below).

Instructions. Implement the algorithm and run it on the dataset with parameters $\delta = e^{-5}, \epsilon = e \times 10^{-4}$. (Note: with this choice of δ you will be using 5 hash functions - the 5 pairs (a, b) that you'll need for the hash functions are in `hash_params.txt`). Then for each distinct word i in the dataset, compute the relative error $E_r[i] = \frac{\tilde{F}[i]-F[i]}{F[i]}$ and plot these values as a function of the exact word frequency $\frac{F[i]}{t}$.

The plot should use a logarithm scale both for the x and the y axes, and there should be ticks to allow reading the powers of 10 (e.g. 10^{-1}, 10^0, 10^1 etc...). The plot should have a title, as well as the x and y axes. The exact frequencies $F[i]$ should be read from the counts file. Note that words of low frequency can have a very large relative error, that is not a bug in your implementation, just a consequence of the bound we proved in question c.

Answer the following question by reading values from your plot: What is an approximate condition on a word frequency in the document to have a relative error below $1 = 10^0$?

Hash functions. You may use the following hash function (see example pseudo-code), with $p = 123457$, a and b values provided in the hash params file and n_buckets chosen according to the specification of the algorithm. In the provided file, each line gives you a, b values to create one hash function.

```
# Returns hash(x) for hash function given by parameters a, b, p and n_buckets
def hash_fun(a, b, p, n_buckets, x)
{
y = x [modulo] p
hash_val = (a*y + b) [modulo] p
return hash_val [modulo] n_buckets
}
```

What to submit

(i) Proof that $\mathsf{Pr}\left[\tilde{F}[i] \leq F[i] + \epsilon t\right] \geq 1 - \delta$. [4(a)]

(ii) Log-log plot of the relative error as a function of the frequency. Answer for which word frequencies is the relative error below 1. [4(b)]

(iii) Submit the code on snap submission site. [4(b)]