CS246: Mining Massive Data Sets Winter 2015

Problem Set 4

Due 5:00pm March 5, 2015

Only one late period is allowed for this homework (3/10).

General Instructions

These questions require thought, but do not require long answers. Please be as concise as possible. Please fill the cover sheet and submit it as a front page with your answers. We will **subtract 2 points** for failing to include the cover sheet. Include the names of your collaborators (see the course website for the collaboration or honor code policy).

All students (Regular and SCPD) should submit their answers using Scoryst (see course website FAQ for further instructions). This submission should include the source code you used for the programming questions.

Additionally, all students (Regular and SCPD) should upload all code through a single text file per question using the url below. Please do not use .pdf, .zip or .doc files for your code upload.

Cover Sheet: `http://cs246.stanford.edu/cover.pdf`

Code (Snap) Upload: `http://snap.stanford.edu/submit/`

Assignment (Scoryst) Upload: `https://scoryst.com/course/39/submit/`

Questions

1 Support Vector Machine (40 Points) [Janice, Dima, Peter]

Note: This is a long implementation question. Make sure to start early.

In this problem, we study an approach for implementing and applying Support Vector Machine (SVM) to a real-life dataset. We will start with discovering a few basic facts about SVM and the effects of the regularization parameter. Then, you will get an opportunity to implement a linear SVM solver of your own. Using the solver that you have implemented, you will discover interesting facts about the SVM implementation by varying multiple parameters and with different gradient update techniques.

First consider the following optimization problem representing a linear SVM (with "hard constraints")

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1. \end{aligned}$$

($(\mathbf{x}_i, y_i)$ represents the ith element in the training set where $y_i \in \{-1, 1\}, \mathbf{x}_i, \mathbf{w} \in \mathbb{R}^d$).

This optimization problem will find the maximum margin separator if the dataset is linearly separable. However, not all data is linearly separable. So consider the following variation of SVM by introducing a regularization factor to account for the training mistakes (also called soft margin SVM)

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i \cdot (\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 - \xi_i, \quad \forall i = 1, \ldots, n \\ & \xi_i \geq 0, \quad \forall i = 1, \ldots, n. \end{aligned}$$

($C \in \mathbb{R}$ is the regularization parameter and ξ_i is the slack variable as described in the lecture slides). This enables SVM to be trained to a dataset that is not linearly separable.

(a) [3 Points]

Task: Give a sample training dataset of 5 points with $\mathbf{x}_i \in \mathbb{R}^2$ (*i.e.* an example of 5 points $(x_i^{(1)}, x_i^{(2)})$ with their respective classes $y_i \in \{-1, 1\}$), such that it is infeasible under SVM with hard constraints but is feasible under soft margin SVM.

(b) [3 Points]

A parameter set $(\mathbf{w}, b, \xi_1 \ldots, \xi_n)$ is called feasible if it satisfies all of the inequality constraints in the given optimization problem (*i.e.* $y_i \cdot (\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 - \xi_i$ is satisfied for all $i = 1, \ldots, n$).

Task: Show that there always exists a feasible parameter set to the soft margin SVM optimization problem for any dataset.

You just need to give an example of $(\mathbf{w}, b, \xi_1, \ldots, \xi_n)$ that satisfies the constraints under any $\mathbf{x}$.

(Hint: With what value can you make the left side of the inequality always hold?)

(c) [5 Points]

Let $(\mathbf{w}, b, \xi_1, \ldots, \xi_n)$ be a feasible parameter set in the soft margin SVM optimization problem.

Task: Prove that the sum of the slack variables $\sum_{i=1}^{n} \xi_i$ is an upper bound to the total number of errors made by the linear classification rule represented by $\mathbf{w}$ on the training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$.

An error is made by the classification rule $\mathbf{w}$ on an example $(\mathbf{x}_i, y_i)$ if $y_i \cdot (\mathbf{x}_i \cdot \mathbf{w} + b) < 0$.

Implementation of SVM via Gradient Descent

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

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

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.

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

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 (d) 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 (d))
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 $\eta = \mathbf{0.0001}, \epsilon = \mathbf{0.001}$.

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.

(d) [2 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 $\nabla_{w^{(j)}} f(\mathbf{w}, b)$.)

(e) [15 (for code) + 4 (for plot) 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 vector) 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 updates (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. **Submit your code on SNAP and attach it to your Scoryst submission.**

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.

(f) [8 Points]

Now, we investigate the effects of the regularization parameter (C).

Task: For the values of $C \in \{1, 10, 50, 100, 200, 300, 400, 500\}$, train your soft margin SVM using Stochastic gradient descent on the training data set and compute the percent error on test data set.

The *convergence criteria* is $\Delta_{cost} < \epsilon$, where

$$\Delta_{cost} = 0.5 * \Delta_{cost} + 0.5 * \Delta_{\%cost},$$

and $\Delta_{\%cost}$ is same as above (equation 3).

Calculate $\Delta_{cost}, \Delta_{\%cost}$ at the end of each iteration of the while loop.
We initialize $\Delta_{cost} = 0$, $\mathbf{w} = \mathbf{0}, b = 0$ and compute $f_0(\mathbf{w}, b)$.

For this method, use $\eta = \mathbf{0.0001}, \epsilon = \mathbf{0.001}$.

The percent error is the number of incorrectly classified samples divided by the number of total testing examples.

Task: Plot C versus the percent error. What do you infer from this plot ?

Use `features.train.txt` and `target.train.txt` as the training data set and `features.test.txt` and `target.test.txt` as the test data set for this problem.

What to submit

(a) An example of infeasible point under SVM with hard constraints but feasible under soft margin SVM.

(b) Proof that soft margin SVM is always feasible.

(c) Proof that sum of slack variables is an upper bound to the total number of errors made.

(d) Equation for $\nabla_b f(\mathbf{w}, b)$.

(e) Plot of $f_k(\mathbf{w}, b)$ vs. the number of updates (k). Time for convergence for each of the gradient descent techniques. Interpretation of plot and convergence times. **Submit the code on SNAP and attach it to your Scoryst submission.**

(f) Plot of $\boldsymbol{C}$ versus the percent error. Description of what the plot means.

2 Decision Tree Learning (15 points) [Clifford, Negar]

(Note: This problem has a lot of text, but each of the individual parts is quite easy.)

Goal. We want to construct a decision tree to find out if a person will enjoy beer.

Definitions. Let there be k binary-valued attributes we could use for this inference.

As discussed in class, we pick a decision at each node that reduces the *impurity* the most:

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

where D is the considered input 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.

2.1 Attributes in decision trees

(a) [4pts]

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.
- 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: Which attribute would you use for the decision at the root if we were to use the gini index metric G defined above?

[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 $I(D) = 10 \times (1 - (0.16 + 0.36))$.

(b) [2pts]

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 target value for constructing a binary decision tree be equal to a_1 for 99% of these examples. (It means that 99% of the examples have the same label as the value of a_1. For example, for all objects with $a_1 = 1$, 99% of them are labeled +.)
- Assume that we build a complete binary decision tree (*i.e.*, we use values of all attributes).

Task: In this case, explain how 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.

2.2 Overfitting in decision trees

To combat the overfitting problem in the previous example, there are pruning strategies. (We let nodes correspond to looking up the value of an attribute, and leaves correspond to the final decision reached about target values.)

Strategy. One such strategy is the following:

1. In the first stage, we build a sequence of trees $T_0, T_1, \ldots, T_m$ using the training data. T_0 is the original tree before "pruning" and T_m is the tree consisting of one leaf[2].
2. In the second stage, one of these trees is chosen as the pruned tree, based on its *generalization error* estimation. The generalization error is the number samples the decision tree misclassified, relative to the total number of samples given.

In the first stage, the tree T_{i+1} is obtained by replacing one or more of the subtrees in the previous tree T_i with suitable leaves. The subtrees that are pruned are those that obtain the lowest increase in apparent error rate per pruned node:

$$\alpha = \frac{\text{error}(\text{pruned}(T, t), S) - \text{error}(T, S)}{|\text{nodes}(T)| - |\text{nodes}(\text{pruned}(T, t))|},$$

where:

[2]In the case of the single leaf T_m, a decision is made without consulting any attributes, so we would simply count the total number of positive and negative examples, and pick the majority to be the value to assign to any example.

- error(T, S) indicates the error rate of the tree T over the sample S;
- $|\text{nodes}(T)|$ denotes the number of nodes in T;
- pruned(T, t) denotes the tree obtained by replacing the node t in T with a suitable leaf.

In the second stage, the generalization error of each pruned tree $T_0, T_1, \ldots, T_m$ is estimated. The best pruned tree is then selected.

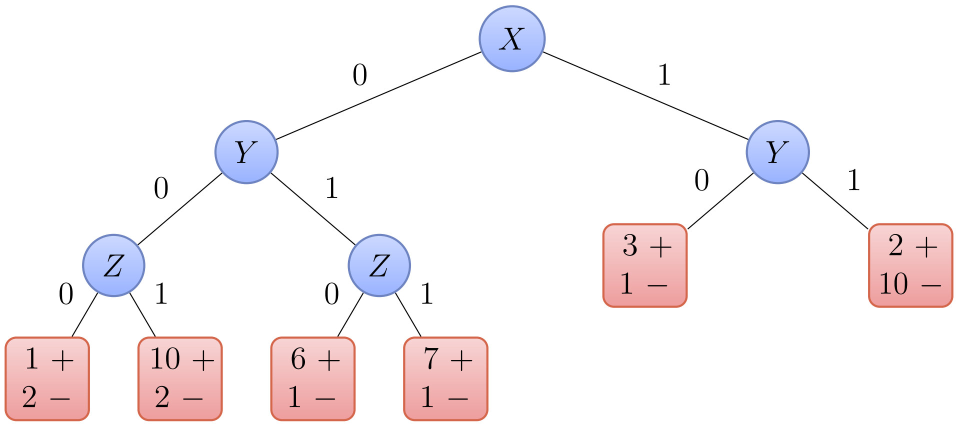

Figure 1: Decision tree T_0.

(c) [6pts]

Consider the tree T_0 on Figure 1. Each node in the tree corresponds to a decision on the labeled attribute, with the left hand side of the branch corresponding to that attribute taking value 0 and the right hand side branch corresponding to the value 1. The final leaves correspond to the number of positive and negative examples.

Task: For the tree T_0 represented in Figure 1, find $T_1, T_2, \ldots, T_m$.

As an example, if we prune the left-most node corresponding to a decision on Z, we will replace it with a leaf containing $1 + 10 = 11$ positive (+) examples and $2 + 2 = 4$ negative (−) examples. So, we would assign positive (+) to any example reaching that point, since it is the majority.

(Hint: For ease of computation, you can prune only one node at a time, i.e *you can prune each internal node individually, calculate the α for each case, and finally choose the node that gives the lowest α.)*

(d) [3pts]

Now to evaluate generalization error. Let us consider a test set containing these examples, of the following form:

- $X = 1$, $Y = 1$, $Z = 1$: answer $= -$;
- $X = 1$, $Y = 0$, $Z = 0$: answer $= -$;
- $X = 0$, $Y = 1$, $Z = 1$: answer $= +$;
- $X = 0$, $Y = 0$, $Z = 0$: answer $= +$.

Task: From among $T_0, T_1, \ldots, T_m$, which tree is has the best generalization error?

What to submit

Points will be taken off for solutions without sufficient work shown.

Sect 2.1 (a) The attribute you would use for the decision at the root.

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

Sect 2.2 (c) The tree sequence $T_1, \ldots, T_m$ after pruning.

(d) The tree with the best generalization error.

3 Clustering Data Streams (15 points) [Nat, James]

Introduction. In this problem, we study an approach for clustering massive data streams. We will study a framework for turning an approximate clustering algorithm into one that can work on data streams, *i.e.*, one which needs a small amount of memory and a small number of (actually, just one) passes over the data. As the instance of the clustering problem, we will focus on the k-means problem.

Definitions. Before going into further details, we need some definitions:

- The function $d : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^+$ denotes the Euclidian distance:
$$d(x, y) = ||x - y||_2.$$
- For any $x \in \mathbb{R}^p$ and $T \subset \mathbb{R}^p$, we define:
$$d(x, T) = \min_{z \in T}\{d(x, z)\}.$$
- Having subsets $S, T \subset \mathbb{R}^p$, and a weight function $w : S \to \mathbb{R}^+$, we define:
$$\text{cost}_w(S, T) = \sum_{x \in S} w(x) d(x, T)^2.$$
- Finally, if for all $x \in S$ we have $w(x) = 1$, we simply denote $\text{cost}_w(S, T)$ by $\text{cost}(S, T)$.

Reminder: k-means clustering. The k-means clustering problem is as follows: given a subset $S \subset \mathbb{R}^p$, and an integer k, find the set T (with $|T| = k$), which minimizes $\text{cost}(S, T)$. If a weight function w is also given, the k-means objective would be to minimize $\text{cost}_w(S, T)$, and we call the problem the weighted k-means problem.

Strategy for clustering data streams. We assume we have an algorithm ALG which is an α-approximate weighted k-means clustering algorithm (for some $\alpha > 1$). In other words, given any $S \subset \mathbb{R}^p$, $k \in \mathbb{N}$, and a weight function w, ALG returns a set $T \subset \mathbb{R}^p$, $|T| = k$, such that:

$$\text{cost}_w(S, T) \leq \alpha \min_{|T'|=k} \{\text{cost}_w(S, T')\}.$$

We will see how we can use ALG as a building block to make an algorithm for the k-means problem on data streams.

The basic idea here is that of divide and conquer: if S is a huge set that does not fit into main memory, we can read a portion of it that does fit into memory, solve the problem on this subset (*i.e.*, do a clustering on this subset), record the result (*i.e.*, the cluster centers and some corresponding weights, as we will see), and then read a next portion of S which is again small enough to fit into memory, solve the problem on this part, record the result, etc. At the end, we will have to combine the results of the partial problems to construct a solution for the main big problem (*i.e.*, clustering S).

To formalize this idea, we consider the following algorithm, which we denote as ALGSTR:

- Partition S into ℓ parts $S_1, \ldots, S_\ell$.
- For each $i = 1$ to ℓ, run ALG on S_i to get a set of k centers $T_i = \{t_{i1}, t_{i2}, \ldots, t_{ik}\}$, and assume $\{S_{i1}, S_{i2}, \ldots, S_{ik}\}$ is the corresponding clustering of S_i (*i.e.*, $S_{ij} = \{x \in S_i |\, d(x, t_{ij}) < d(x, t_{ij'})\ \forall j' \neq j, 1 \leq j' \leq k\}$).
- Let $\widehat{S} = \bigcup_{i=1}^{\ell} T_i$, and define weights $w(t_{ij}) = |S_{ij}|$.
- Run ALG of $\widehat{S}$ with weights w, to get k centers T.
- Return T.

Now, we analyze this algorithm. Assuming $T^* = \{t_1^*, \ldots, t_k^*\}$ to be the optimal k-means solution for S (that is, $T^* = \text{argmin}_{|T'|=k}\{\text{cost}(S, T')\}$), we would like to compare $\text{cost}(S, T)$ (where T is returned by ALGSTR) with $\text{cost}(S, T^*)$.

A small fact might be useful in the analys below: for any $(a, b) \in \mathbb{R}_+$ we have:

$$(a + b)^2 \leq 2a^2 + 2b^2.$$

(a) [5pts]

Next, we show that the cost of the final clustering can be bounded in terms of the total cost of the intermediate clusterings:

Task: Prove that:

$$\operatorname{cost}(S,T) \le 2 \cdot \operatorname{cost}_w(\widehat{S},T) + 2\sum_{i=1}^{\ell} \operatorname{cost}(S_i,T_i).$$

(b) [3pts]

So, to bound the cost of the final clustering, we can bound the terms on the right hand side of the inequality in part (b). Intuitively speaking, we expect the second term to be small compared to $\operatorname{cost}(S,T^*)$, because T^* only uses k centers to represent the data set (S), while the T_i's, in total, use $k\ell$ centers to represent the same data set (and $k\ell$ is potentially much bigger than k). We show this formally:

Task: Prove that:

$$\sum_{i=1}^{\ell} \operatorname{cost}(S_i,T_i) \le \alpha \cdot \operatorname{cost}(S,T^*).$$

(c) [7pt]

Prove that ALGSTR is a $(4\alpha^2 + 6\alpha)$-approximation algorithm for the k-means problem.

Task: Prove that:

$$\operatorname{cost}(S,T) \le (4\alpha^2 + 6\alpha) \cdot \operatorname{cost}(S,T^*).$$

Hint: You might want to first prove two useful facts, which help bound the first term on the right hand side of the inequality in part (a):

$$\operatorname{cost}_w(\widehat{S},T) \le \alpha \cdot \operatorname{cost}_w(\widehat{S},T^*).$$

$$\operatorname{cost}_w(\widehat{S},T^*) \le 2\sum_{i=1}^{\ell} \operatorname{cost}(S_i,T_i) + 2 \cdot \operatorname{cost}(S,T^*).$$

Additional notes: We have shown above that ALGSTR is a $(4\alpha^2 + 6\alpha)$-approximation algorithm for the k-means problem. Clearly, $4\alpha^2 + 6\alpha > \alpha$, so ALGSTR has a somewhat worse approximation guarantee than ALG (with which we started). However, ALGSTR is better suited for the streaming application, as not only it takes just one pass over the data, but also it needs a much smaller amount of memory.

Assuming that ALG needs $\Theta(n)$ memory to work on an input set S of size n (note that just representing S in memory will need $\Omega(n)$ space), if we partitioning S into $\sqrt{n/k}$ equal parts, ALGSTR can work with only $O(\sqrt{nk})$ memory. (Like in the rest of the problem, k represents the number of clusters per partition.)

Note that for typical values of n and k, assuming $k \ll n$, we have $\sqrt{nk} \ll n$. For instance, with $n = 10^6$, and $k = 100$, we have $\sqrt{nk} = 10^4$, which is 100 times smaller than n.

What to submit

(a) Proof that $\text{cost}(S, T) \leq 2 \cdot \text{cost}_w(\widehat{S}, T) + 2\sum_{i=1}^{\ell} \text{cost}(S_i, T_i)$.

(b) Proof that $\sum_{i=1}^{\ell} \text{cost}(S_i, T_i) \leq \alpha \cdot \text{cost}(S, T^*)$.

(c) Proof that $\text{cost}(S, T) \leq (4\alpha^2 + 6\alpha) \cdot \text{cost}(S, T^*)$.

4 Data Streams (30 points) [Clement, Hristo, Jean-Yves]

Introduction. In this problem, we study an approach to approximate the occurrence frequencies 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.

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 of $F[i]$.

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

(a) [3pts]

Task: Prove that for any $1 \le i \le n$:

$$\tilde{F}[i] \ge F[i].$$

(b) [6pts]

Task: Prove that for any $1 \le i \le n$ and $1 \le j \le \lceil \log(\frac{1}{\delta}) \rceil$:

$$\mathsf{E}\left[c_{j,h_j(i)}\right] \le F[i] + \frac{\epsilon}{e}(t - F[i]).$$

(c) [6pts]

Task: Prove that:

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

Note that parts (a) and (c) together show that with high probability (at least $1 - \delta$), $\tilde{F}[i]$ 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.

(d) [15pts]

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

(a) Proof that $\tilde{F}[i] \geq F[i]$.

(b) Proof that $\mathsf{E}\left[c_{j,h_j(i)}\right] \leq F[i] + \frac{\epsilon}{e}(t - F[i])$.

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

(d) Log-log plot of the relative error as a function of the frequency. Answer: for which word frequencies is the relative error below 1. Upload your code on Snap and include it with your Scoryst submission.