Homework 3

10-605/805: Machine Learning with Large Datasets

Due Wednesday, October 4th at 11:59 PM Eastern Time

Submit your solutions via Gradescope, **with your solution to each subproblem on a separate page**, i.e., following the template below.

IMPORTANT: Be sure to highlight where your solutions are for each question when submitting to Gradescope otherwise you will be marked 0 and will need to submit regrade request for every solution un-highlighted in order for fix it!

Note that Homework 3 consists of two parts: this written assignment, and a programming assignment. Remember to fill out the collaboration section found at the end of this homework as per the course policy.

Programming: The programming in this homework is **NOT** autograded however you are required to upload your completed notebooks to Gradescope, otherwise you will not receive credit for the programming sections.

1 Written Section [60 Points]

1.1 Count-Min Sketch Basics (15 pts)

(a) Let's use count-min sketch to estimate the counts of a stream of 5 items: x_1, x_2, x_3, x_4, x_5, using 3 hash functions: h_1, h_2, h_3, each with 3 buckets. The hash functions have the following collision matrix:

collision matrix			
h_1	x_1, x_3	x_4, x_5	x_2
h_2	x_2, x_5	x_3	x_1, x_4
h_3	x_1	x_2, x_3	x_4, x_5

Suppose you see the following stream of data: $\{x_1, x_2, x_5, x_3, x_3, x_4, x_4, x_4\}$.

i. *[2 points]* What is the value of $\hat{c}_1$, the approximate count of item x_1, based on this sketch?
$c_1 \approx min\{3, 4, 1\} = 1$

ii. *[2 points]* What is the value of $\hat{c}_4$, the approximate count of item x_4, based on this sketch?
$c_4 \approx min\{4, 4, 4\} = 4$

iii. *[5 points]* Let c_s be the true count of item x_s. What is the average absolute error of approximation for all items, i.e. $\frac{1}{5}\sum_{s=1}^{5} |\hat{c}_s - c_s|$? Actual:

$$c_1 = 1 \quad c_2 = 1 \quad c_3 = 2 \quad c_4 = 3 \quad c_5 = 1 \tag{1}$$

Approximations:

$$c_1 \approx min\{3, 4, 1\} = 1 c_2 \approx min\{1, 2, 3\} = 1 c_3 \approx min\{3, 2, 3\} = 2 c_4 \approx min\{4, 4, 4\} = 4 c_5 \approx min\{4, 2, 4\} = 2$$

$$\frac{1}{5}(0 + 0 + 0 + 1 + 1) = \frac{2}{5} \tag{2}$$

(b) Consider the approximation guarantee for count-min sketch presented in lecture: given $r > \log(\frac{1}{\delta})$ hash functions, each with $m > \frac{e}{\epsilon}$ buckets, then

$$P\left(\hat{c}_s \geq c_s + \epsilon \|c\|_1\right) \leq \delta$$

i. *[3 points]* How would the number of required buckets change if our tolerance on the discrepancy ϵ is halved?
If ϵ is halved then the number of buckets required will double.

ii. *[3 points]* How would the number of hash functions change if we allowed the probability δ to be twice as large?
The number of hash functions decreases by $\log(2) = 1$.

1.2 An Unbiased Sketch (30 pts)

The count-min sketch presented in lecture is biased, as the estimated count $\hat{c}_s$ for item $s \in \{1, \ldots, k\}$ is always higher than (or equal to) the true count c_s. In this problem, we will explore an *unbiased* sketch.

Let g be a hash function chosen from a family G of independent hashes, such that g maps each item s to either $+1$ or -1 with equal probability:

$$P(g(s) = +1) = P(g(s) = -1) = 1/2.$$

Let's start with a simple version of the sketch, which uses one additional hash function, h, which maps each item s to one of m buckets. Assume that g and h are independent. When we observe an item s in the sequence, we simply update the sketch counter, C, as:

$$C[h(s)] = C[h(s)] + g(s).$$

Then, if we would like to predict the count for item s, we return:

$$\hat{c}_s = C[h(s)]\ g(s).$$

Given this sketch, please answer the following questions:

a) *[8 points]* First, show that this simple sketch is unbiased, i.e., $\mathbb{E}[\hat{c}_s] = c_s$. (*Hint: Find an expression for $\hat{c}_s$ in terms of $g(s)$ and c_s, and use linearity of expectation.*)

$$C[y] = \sum_{i=1}^{k} I(h(i) = h(y)) c_i g(i)$$

$$\begin{aligned}
\hat{c}_s &= C[h(s)]g(s) \\
&= g(s) \sum_{i=1}^{k} I(h(i) = h(s)) c_i g(i) \\
&= g(s)(c_s g(s) + \sum_{i=1, i\neq s}^{k} I(h(i) = h(s)) c_i g(i)) \\
&= c_s + \sum_{i=1, i\neq s}^{k} I(h(i) = h(s)) c_i g(i) g(j)
\end{aligned}$$

$$\begin{aligned}
E[\hat{c}_s] &= E[c_s g_s^2 + \sum_{i=1, i\neq s}^{k} I(h(i) = h(s)) c_i g(i) g(s)] \\
&= E[c_s] + \sum_{i=1, i\neq s}^{k} E[I(h(i) = h(s)) c_i g(i) g(s)] \quad h \text{ independent of } g \\
&= E[c_s] + \sum_{i=1, i\neq s}^{k} E[I(h(i) = h(s)) c_i g(i)] E[g(s)] \\
&= E[c_s] + \sum_{i=1, i\neq s}^{k} E[I(h(i) = h(s)) c_i g(i)] \times 0 \\
&= E[c_s] \\
&= c_s
\end{aligned}$$

b) *[8 points]* Let's understand how much the estimates vary. Prove that: $Var(\hat{c}_s) = \frac{1}{m}\sum_{j\in\{1,\ldots,k\},j\neq s} c_j^2$, where m is the number of buckets for the hash function h.

$$Var[X] = E[X^2] - E[X]^2$$

$$Var(\hat{c}_s) = E[(c_s + \sum_{i=1,i\neq s}^{k} I(h(i) = h(s))c_i g(i)g(s))^2] - c_s^2$$

$$= E[c_s^2 + 2c_s \sum_{i=1,i\neq s}^{k} I(h(i) = h(s))c_i g(i)g(s) + (\sum_{i=1,i\neq s}^{k} I(h(i) = h(s))c_i g(i)g(s))^2] - c_s^2$$

$$= E[c_s^2] + E[2c_s \sum_{i=1,i\neq s}^{k} I(h(i) = h(s))c_i g(i)g(s)] + E[(\sum_{i=1,i\neq s}^{k} I(h(i) = h(s))c_i g(i)g(s))^2] - c_s^2$$

$$= E[2c_s \sum_{i=1,i\neq s}^{k} I(h(i) = h(s))c_i g(i)g(s)] + E[(\sum_{i=1,i\neq s}^{k} I(h(i) = h(s))c_i g(i)g(s))^2]$$

$$= 0 + E[(\sum_{i=1,i\neq s}^{k} I(h(i) = h(s))c_i g(i)g(s))^2] \qquad E[g(s)] = 0$$

$$= E[(\sum_{i=1,i\neq s}^{k} I(h(i) = h(s))c_i g(i)g(s))^2]$$

$$= E[\sum_{i=1,i\neq s}^{k}\sum_{j=1,j=i}^{k} I(h(i) = h(j))c_i g(i)g(s)I(h(j) = h(s))c_j g(j)g(s)$$

$$+ \sum_{i=1,i\neq s}^{k}\sum_{j=1,j\neq i}^{k} I(h(i) = h(s))c_i g(i)g(s)I(h(j) = h(s))c_j g(j)g(s)]$$

$$= \frac{1}{m}\sum_{i=1,i\neq s}^{k} E[(c_i g(i)g(s))^2] + \sum_{i=1,i\neq s}^{k}\sum_{j=1,j\neq i}^{k} E[I(h(i) = h(s))c_i g(i)g(s)I(h(j) = h(s))c_j g(j)g(s)]$$

$$= \frac{1}{m}\sum_{i=1,i\neq s}^{k} E[c_i^2] + 0$$

$$= \frac{1}{m}\sum_{i=1,i\neq s}^{k} c_i^2$$

c) *[6 points]* We will now bound the probability of getting a bad estimate $\hat{c}_s$, i.e., one that differs substantially from the true count c_s. Show that:

$$P(|\hat{c}_s - c_s| \geq \epsilon \|\mathbf{c}_{-s}\|_2) \leq \frac{1}{\epsilon^2 m},$$

where $\mathbf{c}_{-s}$ is a vector of length $(k-1)$ containing the counts for all items except s. *(Hint: Consider Chebyshev's inequality.)*

Chebyshev's inequality is:

$$\Pr(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}.$$

Let $X = \hat{c}_s$, then $\mu = c_s$ and $\sigma = \sqrt{\frac{1}{m}\sum_{i=1, i\neq s}^{k} c_i^2}$

$$\Pr(|\hat{c}_s - c_s| \geq k\sqrt{\frac{1}{m}\sum_{i=1, i\neq s}^{k} c_i^2}) \leq \frac{1}{k^2}.$$

$$\Pr(|\hat{c}_s - c_s| \geq k\sqrt{\frac{1}{m}}\|\mathbf{c}_{-s}\|_2) \leq \frac{1}{k^2}.$$

To make the expression $k\sqrt{\frac{1}{m}}\|\mathbf{c}_{-s}\|_2 = \epsilon\|\mathbf{c}_{-s}\|_2$ let $k = \epsilon\sqrt{m}$.

Then plugging in we get:

$$P(|\hat{c}_s - c_s| \geq \epsilon \|\mathbf{c}_{-s}\|_2) \leq \frac{1}{\epsilon^2 m}$$

d) Similar to count-min sketch, we can improve the results from (c) by using r hash functions for mapping the items to buckets $(h_1, \ldots, h_r)$, and r hash functions for assigning signs $(g_1, \ldots g_r)$. In particular, setting $m > \frac{3}{\epsilon^2}$ and $r > (log(1/\delta))$, we can guarantee:

$$P(|\hat{c}_s - c_s| \geq \epsilon \|\mathbf{c}_{-s}\|_2) \leq \delta \,.$$

Note that while this result appears similar to the one we had for count-min sketch, it requires more buckets, m. Recall that for count-min sketch with $m > \frac{e}{\epsilon}$ and $r > (log(1/\delta))$, we had the bound:

$$P(|\hat{c}_s - c_s| \geq \epsilon \|\mathbf{c}\|_1) \leq \delta \,.$$

With these results in mind, answer the questions below.

i. *[4 points]* Explain a scenario where count-min sketch would be preferable to the unbiased sketch we derived (using the above bounds).

A count-min sketch would be good when there is a smaller number of unique items that need to be hashed. Less buckets means that the hash function can work faster and takes less storage space. Also since there are not too many unique items, there will be few collisions despite there being fewer buckets.

ii. *[4 points]* Explain a scenario where the unbiased sketch we derived would be preferable to count-min sketch (using the above bounds).

Use an unbiased sketch if the data set contains a lot of unique items and more buckets are needed to prevent excessive collisions. However there will be more buckets compared to the count-min sketch which increases the runtime and storage space.

1.3 Locality-Sensitive Hashing (15 pts)

In lecture, we saw that LSH can be used for applications such as nearest neighbors classification and clustering. We mentioned that if two points $\mathbf{x}$ and $\mathbf{y}$ have small *cosine distance*, then the resulting bit vectors formed by using LSH, $\mathbf{x}'$ and $\mathbf{y}'$, will have small *Hamming distance*. In this problem, we will explore this result in more detail.

First, let's recall the definitions of cosine similarity, cosine distance, and Hamming distance.

Cosine similarity measures the cosine of the angle between two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^k$, and can be calculated as:

$$s_{cos}(\mathbf{x}, \mathbf{y}) = \cos(\theta(\mathbf{x}, \mathbf{y})) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2}$$

Cosine distance is measured as $d_{cos}(\mathbf{x}, \mathbf{y}) = 1 - s_{cos}(\mathbf{x}, \mathbf{y})$.

Hamming distance between two *binary* (0-1) vectors $\mathbf{x}', \mathbf{y}'$ of length k measures the number of differing bits, and is calculated as:

$$d_h(\mathbf{x}', \mathbf{y}') = \sum_{i=1}^{k} |\mathbf{x}'_i - \mathbf{y}'_i|$$

(a) *[1 points]* Define $\mathbf{a} = [0, 0, 1, 1, 0]$, $\mathbf{b} = [0, 1, 0, 0, 1]$. What's the *cosine similarity* between $\mathbf{a}$ and $\mathbf{b}$?

$$[0, 0, 1, 1, 0] \cdot [0, 1, 0, 0, 1] = 0$$

(b) *[1 points]* Define $\mathbf{a} = [0, 0, 1, 1, 0]$, $\mathbf{b} = [0, 1, 0, 0, 1]$. What's the *Hamming distance* between $\mathbf{a}$ and $\mathbf{b}$?

$$\|[0, 0, 1, 1, 0] - [0, 1, 0, 0, 1]\|_1 = 4$$

(c) *[3 points]* For a random hyperplane, $\mathbf{z}$, define the following hash function:

$$h_z(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad \mathbf{z} \cdot \mathbf{x} \geq 0 \\ 0 & \text{if} \quad \mathbf{z} \cdot \mathbf{x} < 0 \end{cases}$$

For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^k$, prove that:

$$P[h_z(\mathbf{x}) = h_z(\mathbf{y})] = 1 - \frac{\theta(\mathbf{x}, \mathbf{y})}{\pi},$$

where $\theta(\mathbf{x}, \mathbf{y})$ is the angle between vectors $\mathbf{x}, \mathbf{y}$ measured in radians.

To prove this, use the fact that $P[\mathbf{z} \cdot \mathbf{x} \geq 0, \mathbf{z} \cdot \mathbf{y} < 0] = \frac{\theta(\mathbf{x}, \mathbf{y})}{2\pi}$, i.e., the probability that a random hyperplane separates two vectors is proportional to the angle between them.

$$\begin{aligned} P[h_z(\mathbf{x}) = h_z(\mathbf{y})] &= P[h_z(\mathbf{x}) = 0, h_z(\mathbf{y}) = 0] + P[h_z(\mathbf{x}) = 1, h_z(\mathbf{y}) = 1] \\ &= 1 - P[h_z(\mathbf{x}) = 1, h_z(\mathbf{y}) = 0] + P[h_z(\mathbf{x}) = 0, h_z(\mathbf{y}) = 1] \\ &= 1 - \frac{\theta(\mathbf{x}, \mathbf{y})}{2\pi} - \frac{\theta(\mathbf{x}, \mathbf{y})}{2\pi} \\ &= 1 - \frac{\theta(\mathbf{x}, \mathbf{y})}{\pi} \end{aligned}$$

(d) *[6 points]* In LSH, we estimate $P[h_z(\mathbf{x}) = h_z(\mathbf{y})]$ by selecting b random hyperplanes, and calculating bit vectors $\mathbf{x}'_i = h_i(\mathbf{x})$ and $\mathbf{y}'_i = h_i(\mathbf{y})$ for $i = 1, \ldots, b$. Suppose that after running this procedure, we compute the Hamming distance and find: $d_h(\mathbf{x}', \mathbf{y}') = h$. Using this estimate and part (c), show that:

$$d_{cos}(\mathbf{x}, \mathbf{y}) \approx 1 - \cos\left(\frac{h}{b}\pi\right)$$

Note that this equation shows that if $h = b$, i.e., the bit vectors have a large hamming distance, the cosine distance will also be large ($d_{cos}(\mathbf{x}, \mathbf{y}) = 2$). Similarly, for $h = 0$, we will have $d_{cos}(\mathbf{x}, \mathbf{y}) = 0$. (*Hint: The proof should only be a few lines.*)

$$P[h_z(\mathbf{x}) = h_z(\mathbf{y})] = 1 - \frac{\theta(\mathbf{x}, \mathbf{y})}{\pi} \approx 1 - \frac{h}{b}$$

$$\frac{\theta(\mathbf{x}, \mathbf{y})}{\pi} \approx \frac{h}{b}$$

$$\theta(\mathbf{x}, \mathbf{y}) \approx \frac{h}{b}\pi$$

$$d_{cos}(\mathbf{x}, \mathbf{y}) = 1 - \cos(\theta(x, y)) \approx 1 - \cos\left(\frac{h}{b}\pi\right)$$

(e) Let's go through a simple example using LSH. Suppose we have two vectors, $\mathbf{x} = [3, 4, 5, 6]$ and $\mathbf{y} = [4, 3, 2, 1]$.

i. *[1 points]* Use one random vector: $\mathbf{z}_1 = [0.16, -0.26, -1.12, 0.01]$ to perform LSH. Compute the error of the resulting estimated cosine distance, i.e., $|1 - \cos\left(\frac{h}{b}\pi\right) - d_{cos}(\mathbf{x}, \mathbf{y})|$.

$$\mathbf{x} \cdot \mathbf{z}_1 = -6.1 \quad \mathbf{y} \cdot \mathbf{z}_1 = -2.37$$

$$h_{zv}(\mathbf{x}) = 0 \quad h_{zv}(\mathbf{y}) = 0$$

$$|1 - \cos\left(\frac{h}{b}\pi\right) - d_{cos}(\mathbf{x}, \mathbf{y})| = |1 - \cos\left(\frac{0}{1}\pi\right) - 0.21250076904184234|$$

$$0.21250076904184234$$

ii. *[1 points]* Now let's try with more vectors. Use four random vectors: $\mathbf{z}_1 = [0.16, -0.26, -1.12, 0.01]$, $\mathbf{z}_2 = [1.08, -1.14, 0.8, -0.8]$, $\mathbf{z}_3 = [1.94, 0.08, 1.83, 0.17]$, $\mathbf{z}_4 = [-0.88, -1.01, 1.36, -0.07]$ to perform LSH. Compute the error of the estimated cosine distance.

$$x' = [0, 0, 1, 0] \quad y' = [0, 1, 1, 0]$$

$$|1 - \cos\left(\frac{h}{b}\pi\right) - d_{cos}(\mathbf{x}, \mathbf{y})| = |1 - \cos\left(\frac{1}{4}\pi\right) - 0.21250076904184234|$$

$$0.08039244977161009$$

iii. *[2 points]* In practice, it can be sufficient to consider random vectors only consisting of $\{-1, +1\}$ when performing LSH. Use the signs of the previous random vectors, i.e., $\mathbf{z}_1 = [+1, -1, -1, +1]$, $\mathbf{z}_2 = [+1, -1, +1, -1]$, $\mathbf{z}_3 = [+1, +1, +1, +1]$, $\mathbf{z}_4 = [-1, -1, +1, -1]$, to perform LSH. Compute the error of the estimated cosine distance.

$$x' = [1, 0, 1, 0] \quad y' = [1, 1, 1, 0]$$

$$|1 - \cos\left(\frac{h}{b}\pi\right) - d_{cos}(\mathbf{x}, \mathbf{y})| = |1 - \cos\left(\frac{1}{4}\pi\right) - 0.21250076904184234|$$

$$0.08039244977161009$$

2 Programming Section [40 Points]

2.1 Introduction

In this coding section, we will explore a variant of approximate nearest neighbors (ANN) search for images based on locality-sensitive hashing (LSH). As discussed in class, LSH consists of a hashing function which encourages collisions between nearby points in the original space. Choosing the right hashing function is somewhat of an art and depends largely on the domain. In this assignment, we will develop the *cross-polytope hash* for image data. Roughly, the hash first projects an input sample to a lower dimensional space (by using a JL style projection) and then selects the coordinate with the highest magnitude in the resulting projected vector. The **index** of this coordinate, along with its sign, serves as a hash for the given input. One could then perform multiple such projections and stack all the individual hash values to form a **fingerprint**. The number of projections and their dimensionality is a hyperparameter that we need to tune.

3 Logistics and AWS

We provide the code template for this assignment is *one* Jupyter notebook. What you need to do is to follow the instructions in the notebook and implement the missing parts marked with '`<FILL_IN>`' or '`# YOUR CODE HERE`'.

For this homework you will be diving into using AWS's Elastic MapReduce (EMR) systems. As such we strongly recommend that you start this homework early, as learning this is not something you will want to put off until the last moment. For setting up these services we hosted a lecture on September 22nd, which guided you through the set up of everything you need to get started with AWS, we recommend going through all steps of this to familiarize yourself with the service. You can find the slides here: AWS Guide

If you are enrolled in the course and applied for them via the Google form, you should have $150 AWS credits applied to your account. While we expect that all AWS based homeworsk will cost you $60 in total, this extra amount is there in case it takes you more. If you use up all of your $150 your credit card which you opened your AWS account with will be charged! Please be sure to set up usage warnings to make sure that you are not overspending. Here is a list of everything you will create which is chargeable in AWS:

1. EC2 Instances
2. EBS Volumes
3. EMR Instances
4. S3 Bucket Usage - Very small cost

When you are not working on the homework we strongly recommend you terminate/delete instances/volumes as the cost for leaving these open, even idle, can build up fast. The only thing we suggest keeping open is the S3 bucket until you are completely finished with HW4, since this costs very little. You should plan to run cheap instances while developing your code on a subset of data and only run it on something larger (and more expensive) once you have everything ready.

3.1 HW3 AWS Requirements and Getting lab files

For HW3 you will need to use a S3 bucket and EMR Instances. We recommend writing your code using one m4.xlarge 'worker' instance using a subset of the data and using two m4.xlarge 'worker' instances when you have finished your notebook and need to run on the full dataset. **Make sure that your AWS is current set to N.Virginia US-EAST-1.**

You can obtain the notebooks `'HW3-LSH.ipynb'` in the homework 3 handout .zip file. You should upload this to your Jupyterlab hosted on your AWS EMR cluster. **Be sure to download your completed notebook before you terminate your AWS cluster, otherwise you will lose all completed data.**

You will need to set up an EMR cluster with the appropriate following the instructions in the AWS Guide, you can also review the lecture recording of the class hosted on Friday 9/22. It is important that you add a 'Step' to your EMR cluster which will get the data from the public S3 bucket. Then connect to your JupyterHub application connected to your EMR cluster.

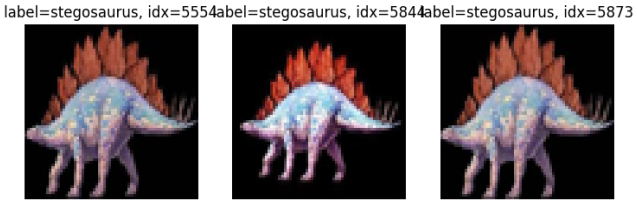
3.2 Preparing for submission

We provide several public tests via `assert` in the notebook, which will help you know if you are completing the notebook correctly. You may want to pass all those tests before submitting your homework. There is no Autograder for this homework, however in order to be given any credit for your programming section, you **MUST** upload your completed notebook to the HW3 Programming submission slot on Gradescope.

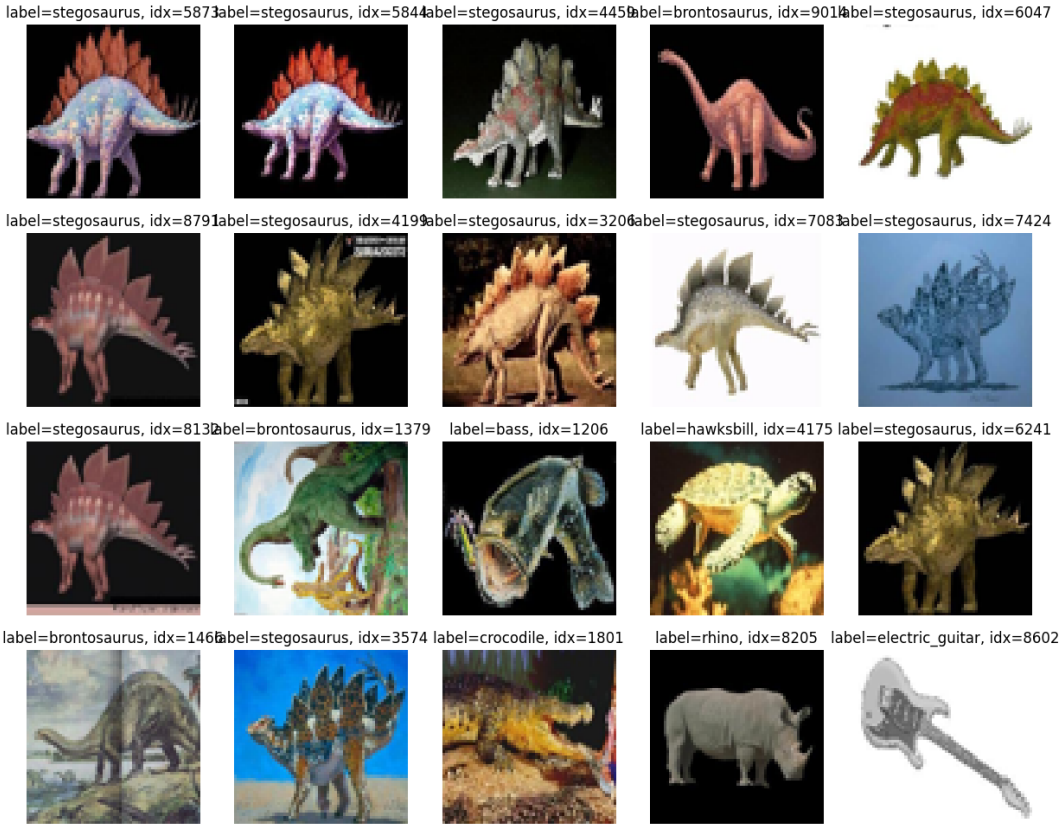
3.3 Deliverables

Include the query image in the neighbor set for questions (a)-(d) below.

(a) *[8 points]* Visualize up to 20 LSH neighbors for the test image 'stegosaurus/image_0043.jpg'.

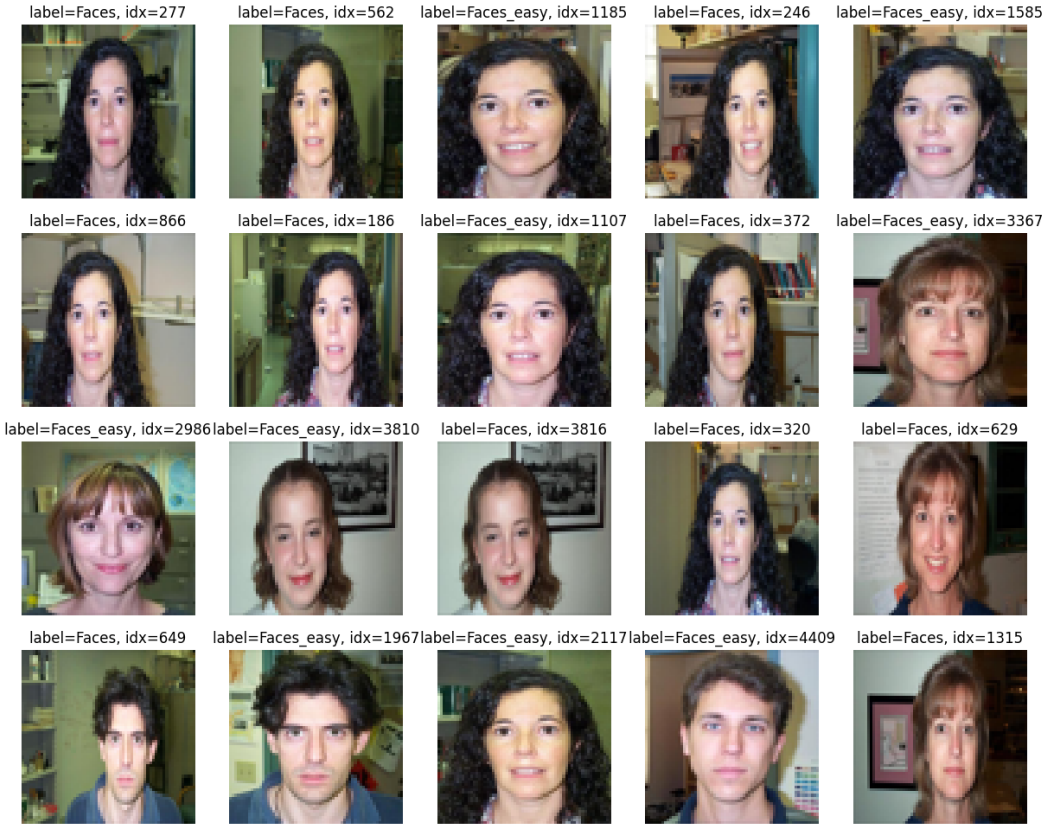


(b) *[8 points]* Visualize up to 20 kNN neighbors for the test image 'stegosaurus/image_0043.jpg' using VGG embeddings.



(c) *[8 points]* Visualize up to 20 LSH neighbors for the test image 'Faces/image_0334.jpg'.

(d) *[8 points]* Visualize up to 20 kNN neighbors for the test image 'Faces/image_0334.jpg' using VGG embeddings.



(e) *[4 points]* What can you say about the neighbors returned by LSH versus those returned by kNN+VGG? Which one returns more "semantically" similar neighbors (i.e., the content of the images is the same), and which one returns "pixel"-similar neighbors (i.e., corresponding pixel values are similar)?

kNN+VGG returns more semantically similar neighbors (there is a picture of a flamingo in the LSH neighbors for the face pictures). The LSH algorithm returns more pixel-similar neighbors as all the images it responds tend to have similar color palette.

(f) *[4 points]* Assume we are performing millions of queries. What can you say about the runtime of each method (LSH vs kNN)? Include any preprocessing steps in your calculation.

LSH is much faster than kNN if processing millions of queries.

The runtime of the kNN method for each query is roughly $O(nm)$ where n is the number of images and m is the number of features for each image because it's necessary to calculate the distance between the desired image and all the other images. Computing the distance between two images takes $O(m)$ time where m is the number of "features" of that image. Hence the runtime for computing Q queries is $O(Qnm)$ The runtime of the LSH method is $O(nmd)$ to preprocess the images, where d is the reduced number of dimensions. Comparing two pictures with LSH is simply comparing the d fingerprints for each picture so bucketing the images takes $O(nd)$ time. Lastly querying the images takes $O(1)$ time since all that needs to be done is pulling the fingerprint. Thus overall the runtime is $O(nmd)$

4 Collaboration Questions

1. (a) Did you receive any help whatsoever from anyone in solving this assignment?

   Gabriel Fonseca

   (b) If you answered 'yes', give full details (e.g. "Jane Doe explained to me what is asked in Question 3.4")
   Helped me set up AWS.

2. (a) Did you give any help whatsoever to anyone in solving this assignment?

   Gabriel Fonseca

   (b) If you answered 'yes', give full details (e.g. "I pointed Joe Smith to section 2.3 since he didn't know how to proceed with Question 2")
   Explained the theory behind some of the programing.

3. (a) Did you find or come across code that implements any part of this assignment?
   No

   (b) If you answered 'yes', give full details (book & page, URL & location within the page, etc.).