Assignment 3
CSE 547/Stat 548: ML for Big Data

University of Washington

0 Certify you read the HW Policies [0 points]

Please explicitly answer the three questions below and include your answers marked in a "problem 0" in your solution set. *If you have not included these answers, your assignment will not be graded.*

Readings: Read the required material.

Submission format: Submit your HW as a *single* typed pdf document; this document must contain all plots. Handwritten solutions (even if they are scanned in to the pdf) will *not* be accepted. Submit all your code in a gzipped tarball named `code.tgz`. It is encouraged that students use latex, though another comparable typesetting method is also acceptable. Some free tools that might help: ShareLaTeX (`www.sharelatex.com`), TexStudio (Windows), MacTex (Mac), TexMaker (cross-platform), and Detexify[2] (online). If you want to type, but don't know (and don't want to learn) LaTeX, consider using a markdown editor with real-time preview and equation editing (e.g., `stackedit.io`, `marxi.co`).

Written work: Please provide succinct answers *along with succinct reasoning for all your answers*. Points may be deducted if long answers demonstrate a lack of clarity. Similarly, when discussing the experimental results, concisely create tables and figures to organize the experimental results. In other words, all your explanations, tables, and figures for any particular part of a question must be grouped together.

Python source code: for the programming assignment. Please note that we will not accept Jupyter notebooks. Submit your code together with a neatly written README file to instruct how to run your code with different settings (if applicable). We assume that you always follow good practice of coding (commenting, structuring); these factors are not central to your grade.

Coding policies: You must write your own code. You are welcome to use any Python libraries for data munging, visualization, and numerical linear algebra. Examples includes Numpy, Pandas, and Matplotlib. If you use TensorFlow or PyTorch, you may *not* use any functions which define a neural network for you, e.g. no Keras is allowed. In PyTorch, you may *not* use the torch.nn.module (or any of the inherited functions). Basically, this means that you are not allowed to define any sort of neural network through the library: you must just write out the "forward pass" yourself; you may not use any built in functions/libraries which specify the number of nodes/layers in your network; the ML library you are using should not ever know you are coding up a neural net (it should be building computation graphs for the computations you specify). You are, however, allowed to use the torch.nn.functional interface provided by PyTorch.

On some questions, we will explicitly not allow ML packages, like Scikit-Learn, TensorFlow, or PyTorch. If in doubt, post to the message boards or email the instructors.

Collaboration: It is acceptable for you to discuss problems with other students; it is not acceptable for students to look at another students written answers. It is acceptable for you to discuss coding questions with others; it is not acceptable for students to look at another students code. Each student must understand, write, and hand in their own answers. In addition, each student must write and submit their own code in the programming part of the assignment.

Acknowledgments: We expect the students not to refer to or seek out solutions in published material from previous years, on the web, or from other textbooks. Students are certainly encouraged to read extra material for a deeper understanding.

0.1 List of Collaborators

List the names of all people you have collaborated with and for which question(s).

0.2 List of Acknowledgements

If you do inadvertently find an assignment's answer, acknowledge for which question and provide an appropriate citation (there is no penalty, provided you include the acknowledgement). If not, then write "none".

0.3 Certify that you have read the instructions

Please make sure to read and follow these instructions. Write "I have read and understood these policies" to certify this. If you do not yet understand what it means not to use the PyTorch "torch.nn.module", please state the you will figure out how to avoid using the nn.module class (e.g. by making sure you post questions on the discussion board/going to office hours/websearch/etc if in doubt). It is fair game to use the "torch.nn.functional" mode.

1 Gaussian Random Projections and Inner Products [15 Points]

In this problem, you will show that inner products are approximately preserved using random projections. Let $\phi(x) = \frac{1}{\sqrt{m}}Ax$ represent our random projection of $x \in \mathbb{R}^d$, with A an $m \times d$ projection matrix with each entry sampled i.i.d from $N(0,1)$. (Note that each row of A is a random projection vector, $v^{(i)}$.)

The *norm preservation theorem* states that for all $x \in \mathbb{R}^d$, the norm of the random projection $\phi(x)$ approximately maintains the norm of the original x with high probability:

$$P\left((1-\epsilon)\|x\|^2 \leq \|\phi(x)\|^2 \leq (1+\epsilon)\|x\|^2\right) \geq 1 - 2e^{-(\epsilon^2-\epsilon^3)m/4}, \tag{1}$$

where $\epsilon \in (0, 1/2)$.

Using the norm preservation theorem, prove that for any $u, v \in \mathbb{R}^d$ s.t. $\|u\| \leq 1$ and $\|v\| \leq 1$,

$$P(|u \cdot v - \phi(u) \cdot \phi(v)| \geq \epsilon) \leq 4e^{-(\epsilon^2-\epsilon^3)m/4}. \tag{2}$$

Note that $u \cdot v$ is the original dot product, and $\phi(u) \cdot \phi(v)$ is the dot product for the random projections. This statement puts a probabilistic bound on the distance between the two dot products.

2 LSH for Angle Similarity [20 Points]

Now suppose we want to use LSH for angular similarity. Suppose our set of n points $D = \{p_1, \ldots p_n\}$ are vectors in d dimensions. Our problem is: given a query point q find a point $p \in D$ which has a small angle with q. Recall the angle between two vectors a and b is $\cos^{-1} a \cdot b/(\|a\|\|b\|)$

As doing this exactly may be computationally expensive, let us try to do this approximately with a fast algorithm. The approximate objective is as follows: suppose there exists a point $p \in D$ which has cosine similarity larger than θ, then our goal is return a point with cosine similarity greater than $c\theta$, where $c < 1$.

Let us try to do this with LSH. Let us consider the a family of hash functions, where $h(p) = \text{sign}(u \cdot p)$, where we will sample u uniformly at random from a Gaussian (or from a unit sphere).

1. (12 points) Provide an exact expression for $\Pr(h(p) = h(p'))$ based on the some geometric relation between p and p'.

2. (4 points) Provide an expression for P_1 and P_2 in terms of θ and $c\theta$. Note that since we want a large angle, you should use:

   (a) If angle$(p, p') > \theta$, then $\Pr(h(p) = h(p')) \geq P_1$.

   (b) If angle$(p, p') < c\theta$, then $\Pr(h(p) = h(p')) \leq P_2$.

3. (4 points) Now provide expressions for query time for point q, the space to store the hash tables, and the construction time of our datastructure.

3 (Dual) Coordinate Ascent [25 Points]

In this problem, we will derive the dual coordinate ascent algorithm for the special case of ridge regression. In particular, we seek to solve the problem:

$$\min_w L(w) \text{ where } L(w) = \sum_{i=1}^{n} (w \cdot x_i - y_i)^2 + \lambda \|w\|^2$$

Recall that the solution is

$$w^* = (X^\top X + \lambda I)^{-1} X^\top Y,$$

where X be the $n \times d$ matrix whose rows are x_i, and Y is an n-dim vector.

Recall the argument:

$$\begin{aligned} w^* &= (X^\top X + \lambda I)^{-1} X^\top Y \\ &\overset{(a)}{=} X^\top (X X^\top + \lambda I)^{-1} Y \\ &:= \frac{1}{\lambda} X^\top \alpha^* \end{aligned}$$

where $\alpha^* = (I + X X^\top / \lambda)^{-1} Y$.

1. *(5 points)* (Linear algebra brush up) Show that the equality labeled (a) in the above is valid (without making any assumptions on n or d). Would this equality be valid if $\lambda = 0$? Why or why not?

As in class, define:

$$G(\alpha_1, \alpha_2, \ldots, \alpha_n) = \frac{1}{2} \alpha^\top (I + X X^\top / \lambda) \alpha - Y^\top \alpha$$

Now let us consider the consider the coordinate ascent algorithm:

- start with $\alpha = 0$.
- choose coordinate i randomly, and update:

$$\alpha_i = \arg\min_z G(\alpha_1, \ldots, \alpha_{i-1}, z, \alpha_{i+1}, \ldots, \alpha_n)$$

- repeat the previous step until some termination criterion is reached.
- return $w = \frac{1}{\lambda} X^\top \alpha$.

2. *(7 points)* Show that the solution to the inner optimization problem for α_i is:

$$\alpha_i = \frac{(y_i - \frac{1}{\lambda}(\sum_{j \neq i} \alpha_j x_j) \cdot x_i)}{1 + \|x_i\|^2 / \lambda} \tag{3}$$

3. *(2 points)* What is the computational complexity of this update, as it is stated? (Assume that scalar multiplication and scalar addition is order one.)

4. *(1 points)* What is the computational complexity of one stochastic gradient descent update?

Now let us consider the update rule from class:

- Start with $\alpha = 0$, $w = \frac{1}{\lambda} X^\top \alpha = 0$.
- Choose coordinate i randomly and perform the following update:
  - Compute the differences:

$$\Delta\alpha_i = \frac{(y_i - w \cdot x_i) - \alpha_i}{1 + \|x_i\|^2/\lambda} \tag{4}$$

  - Update the parameters as follows:

$$\alpha_i \leftarrow \alpha_i + \Delta\alpha_i, \quad w \leftarrow w + \frac{1}{\lambda} x_i \cdot \Delta\alpha_i \tag{5}$$

- Repeat the previous step until some termination criterion is reached.
- Return $w = \frac{1}{\lambda} X^\top \alpha$.

Now let us examine why this algorithm is more efficient.

5. *(7 points)* Prove that the update of equation (5) is valid.

6. *(3 points)* What is the computational complexity of the update defined by equations (4) and (5)? Compare to the complexity of equation (3).

4 Project milestone [80 points]

Introduction This task requires you to build a simple object detection model. Since this is a part of your project, we only give you high level instructions. You are free to be creative and fill in the details yourself. First we shall review some preprocessing primitives, the code for which has been provided:

- Region proposals: candidate image patches that are likely to contain objects.
- Extracting features of an image patch.
- Extracting ground truth labels.
- IoU: Discerning whether a patch contains an image or not.

Region proposals The first step is to identify patches in an image that are likely to contain objects. One simple but costly approach is to iterate through all possible rectangles in an image. The cost of this operation however, is $O(m^2n^2)$ for an image of size $m \times n$. In other words, it is prohibitively expensive. Instead, we shall use a region proposal algorithm known as selective search. Selective search essentially returns a large number (say, 1000 or 2000) of bounding boxes corresponding to all patches in an image that are most likely to be objects. These region proposals can be noisy, overlapping and may not contain the object perfectly but amongst all these region proposals, we usually have one proposal which will be very close to the actual object in the image. We have provided code hat shows you how to use the selective search functionality of OpenCV. We have also provided the outputs of one run of selective search since it can be quite time consuming (several seconds per image).

Extracting features of an image patch You have been provided features corresponding to an entire image. Suppose we have a bounding box, we now need to extract features of the image patch corresponding to a bounding box. We have provided code that extracts $\sim 10K$ dimensional feature vector per image patch. We use adaptive max pooling to achieve this. There are better ways of obtaining features, but this is computationally inexpensive and easy.

Extracting ground truth labels The next step is to extract ground truth labels for the locations of these objects. Fortunately, the COCO API comes to our rescue. In the homeworks, you have extracted categories and supercategories. You can likewise access the ground truth bounding boxes as well.

IoU Now that we have candidate bounding boxes, how do we decide whether the corresponding patch contains an object? We shall use a metric known as the intersection-over-union (IoU). The IoU between two rectangles r_1 and r_2 is

$$\mathrm{IoU}(r_1, r_2) = \frac{\mathrm{Area}(r_1 \cap r_2)}{\mathrm{Area}(r_1 \cup r_2)}\,.$$

Clearly, the IoU is bounded between 0 and 1, and will be higher for rectangles that overlap more. Here is the rule we will use: *a rectangle r_1 is said to contain an object if there exists a ground truth bounding box r containing said object such that* $\text{IoU}(r_1, r) \geq 0.5$. We have provided code that calculates the IoU of two rectangles.

Object detection

Our object detector will comprise of a binary classifier per category: given features of an image patch corresponding to a bounding box, does this patch contain an object of the category of interest? You are welcome to try out your favorite binary classifier. Common examples include linear SVM, logistic regression, MLPs. Note that you will have to regularize appropriately to prevent overfitting. Also, keep in mind the amount of data you have before you choose high capacity models such as MLPs.

Subsampling negatives The classification problem is pretty unbalanced. We have a small number of positives (corresponding to the rectangles with IoU > 0.5 with some ground truth bounding box) and a large number of negatives (all others). You must sub-samples negatives to ensure that your models learns something meaningful. You are free to decide how many negatives to subsample. A good starting point is to have two negatives per positive.

Hard negative mining You may first start with randomly subsampled negatives. The next step is to iteratively improve your classifier with what is known as "hard negative mining". Essentially, you re-train your model with hard negatives: negative exampels that your model actually thought were positives. It is crucial to have randomly subsampled negatives as well. You now iterate this process. You may try other tricks such as maximizing diversity in your negative examples. This part is open ended and you are welcome to be creative.

An **example pipeline** is given below: Suppose you have n positive examples for a given class.

- Randomly sample $2n$ negative examples. Train a binary logisitic regression model C_0.
- For $t = 1, \cdots, 10$ (or until convergence)
  - Randomly sample n negatives. Pick n negative examples that were assigned the highest probability of being positive by C_{t-1}.
  - Train C_t on the n positive examples and this newly constructed set of $2n$ negative examples.

Performance Metrics

Performance is measured by the average precision (AP) per class and the mean average precision (mAP) a single number summary across all classes.

Tips and tricks

Extracting features for image patches is time consuming. On the other hand you could extract features corresponding to all patches once as a preprocessing step. However, caching all features takes up too much memory. You need to be clever about how you trade off memory consumption vs. computation. For instance, it would make sense to cache features when training your binary classifier. There are simply too many patches to extract features for all of them a priori. Working with big data often entails trade off such as this. We hope that you gain some practical experience on how to deal with these issues.

Familar useful pointers:

- It is important to note that AWS credits must be rationed and used through out the course. Running out of AWS credits is a sign that the rationing and management of AWS credits has not been good enough. The assignments/projects have been designed in a way that a budgeted utilization will make sure that a student does not run out of credits.
- Always use your own laptops for the purpose of debugging and unit testing your code. AWS should be used only when you are sure that the code is stable and does not carry any bugs, to the best of your knowledge.

Report

Your report must contain the following

- Details about your experimental setup. Which binary classifier did you use and why? What other options have you tried?
- Details about how you chose your negative examples and how many negative examples you chose. Which other methods did you try? What was the rationale behind your choice?
- Optimization plots: train and validation loss and AP as you train your model, measured twice every pass through the data. Follow good plotting practices as delineated on Canvas.
- Performance metrics: AP per class and mAP on the test set.
- Other details: Any other tricks you may have used to improve your model.