CSE546 Machine Learning, Autumn 2016: Homework 3

Due: Friday, November 18th, 5pm

Policies

Coding: You must write your own code. You may use any numerical linear algebra package, but you may *not* use machine learning libraries (e.g. sklearn, tensorflow) unless otherwise specified. In addition, each student must write and submit their own code in the programming part of the assignment (we may run your code).

Acknowledgments: We expect you to make an honest effort to solve the problems individually. As we sometimes reuse problem set questions from previous years, covered by papers and webpages, we expect the students not to copy, refer to, or look at the solutions in preparing their answers (referring to unauthorized material is considered a violation of the honor code). Similarly, we expect to not to google directly for answers (though you are free to google for knowledge about the topic). If you do happen to use other material, it must be acknowledged here, with a citation on the submitted solution.

Readings

Read the required material.

Required HW submission format:

The following is the required HW submission format:

- Submit all the answers to the HW as one single *typed* pdf document (not handwritten). This document must contain all plots. It is encouraged you latex all your work, though you may use another comparable typesetting method.
- Additionally, submit your code as a separate archive file (a .tar or .zip file). All code *must* be submitted. The code should be in a runnable file format like .py files or .txt files. Jupyter notebooks are not acceptable.
- List the names of all people you collaborated with and for which question(s). Please mark this as Question 0. Each student must understand, write, and hand in their own answers. Also, each student must understand, write, and hand in their own code.

0 Collaboration and Acknowledgements

List the names of all people you collaborated with and for which question(s). Each student must understand, write, and hand in their own answers. Also, each student must understand, write, and hand in their own code.

1 PCA and reconstruction [25 points]

Let's do PCA and reconstruct the digits in the PCA basis.

As you will be making many visualizations of images, plot these images in reasonable way (e.g. make the images smaller).

1.1 Matrix Algebra Review [5 points]

The trace of a square matrix M, denoted, by $\mathrm{Tr}(M)$ is defined as the sum of the diagonal entries of M.

1. *(2 points)* Show that $\mathrm{Tr}(AB^\top) = \mathrm{Tr}(B^\top A)$ for two matrices A and B of size $n \times d$.
2. *(3 points)* Now we prove a few claims that helpful in the next problem. Define $\mathbf{\Sigma} := \frac{1}{n}\mathbf{X}^\top\mathbf{X}$, where X is the $n \times d$ data matrix (and $d \leq n$). Let X_i be the i-th row (so X_i is a d-vector). Let $\lambda_1, \lambda_2, \ldots \lambda_d$ be the eigenvalues of Σ. Show that:

$$\mathrm{Tr}(\Sigma) = \sum_{i=1}^{d} \lambda_i = \frac{1}{n}\sum_{i=1}^{n} \|X_i\|^2$$

1.2 PCA [8 points]

Define Σ, a $784x784$ matrix, as follows:

$$\Sigma = \frac{1}{N}\sum_i x_i x_i^\top$$

where the x's are points in our training set.

Now compute the top 50 PCA dimensions; these are the 50 dimensions which best reconstruct the data.

1. *(2 points)* What are the eigenvalues λ_1, λ_2, λ_{10}, λ_{30}, and λ_{50}? Also, what is the sum of eigenvalues $\sum_{i=1}^{d}\lambda_i$? (Hint: use the answer from the previous question)
2. *(3 points)* It is straight forward to see that the fractional reconstruction error of using the top k out of d directions is $1 - \frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$. Plot this fractional reconstruction error from 1 to 50? (so the X-axis is k and the Y-axis is the fractional reconstruction error).
3. *(1 points)* What does the first eigenvalue capture, and why do you think λ_1 is much larger than the other eigenvalues?
4. *(2 points)* To view things more easily, plot the fractional reconstruction error from dimensions 2 to 50? (so the X-axis is from 2 to 50)

1.3 Visualization of the Eigen-Directions [4 points]

Now let us get a sense of the what the top PCA directions are capturing (recall these are the directions which capture the most variance).

1. *(3 points)* Display the first 10 eigenvectors as images.
2. *(1 points)* Provide a brief interpretation of what you think they capture.

1.4 Visualization and Reconstruction [8 points]

1. *(1 points)* Choose 5 digits (make sure they are from different classes). Visualize these digits. In particular, include pictures of these 5 digits.
2. *(6 points)* Now let us visualize these digits, after we project them onto the top k eigenvectors of Σ. In particular, if U is the $d \times k$ matrix of eigenvectors, the reconstruction matrix will be $UU^\top$.
   (a) Do this for $k = 2$.
   (b) Do this for $k = 5$.
   (c) Do this for $k = 10$.
   (d) Do this for $k = 20$.
   (e) Do this for $k = 50$.
3. *(1 points)* Provide a brief interpretation, in terms of your perceptions of the quality of these reconstructions

2 Let's get to state of the art on MNIST! [45 points]

We will now shoot to get to "state of the art" on MNIST, without "distortions" or "pre-processing". The table in `http://yann.lecun.com/exdb/mnist/`, shows which strategies use this pre-processing.

If we wanted even higer accuracy, we should really move to convolutional methods, which we will discuss later in the class.

SK: I am altering this problem to make it more stable and easier to get to state of the art. If you used the previous Kernel method, then I found it harder to get to near to state of the art (in a robust manner). I'll still give credit if you go with the previous version (and with a lower test error). However, here I'll lay out a simpler and more robust scheme, which uses Fourier features (these actually work extremely well in practice).

Dimension Reduction with PCA

Project each image onto the top 50 PCA directions. This reduces the dimension of each image from 784 to 50. The reason for doing this is to speed up the computation [1].

[1] In this particular case, PCA actually seems to improve classification performance non-trivially.

Random Fourier Features to approximate the RBF Kernel

Now we will map each x to a k-dimensional feature vector as follows:

$$x \to (h_1(x), h_2(x), \dots h_k(x))$$

We will use $k = N = 60,000$. Each h_j will be of the form:

$$h_j(x) = sin\left(\frac{v_i \cdot x}{\sigma}\right)$$

To construct all these vectors $v_1, v_2, \dots v_k$, you will independently sample every coordinate for every vector from a standard normal distribution (with unit variance). Here, you can think of σ as a bandwidth parameter.

In practice, setting σ is often done with the 'median trick', which is the median of the pairwise distances (between the x's) in your dataset. As this a scalar parameter, I often set σ by: First, I just randomly grab a few pairs of points and estimate the mean distance between a random pair of points (I often use the mean rather than the median). Second, I usually multiplicatively cut it down by some factor (maybe $2, 4, 8, ...$ depending on the problem).

SK: You can read more about this at: `https://people.eecs.berkeley.edu/~brecht/kitchensinks.html`*. Technically, you should really use sin's and cosine's (or a random phase). For this problem, just sin's works fine.*

Tips (and some of my own practices)

SK: For this problem, I found cutting the mean down by a factor of 2 *works well (for the bandwidth). I used a batch size of* 10*, and I get under* 2% *in one pass through the dataset.*

Let's optimize with SGD (there are not many other options out there for problems of this size).

Stochastic optimization is still a bit of an art on large problems. Practitioners tend to have their own best practices, and the exercises are designed to give you experience developing your own. Here are some tips, broadly speaking (not necessarily in the context of this problem):

- (memory) As the dimension is large in this problem, we seek to avoid explicitly computing and storing the full data matrix, which is of size $k \times N = N \times N$. Instead, you can compute the feature vector $h(x)$ 'on the fly', i.e. you recompute the vector $h(x)$ whenever you access image x.
- (regularization) As the dimension is N, note that without regularization and with exact optimization, you would have 0 square loss (and horribly overfit). However (at least for the square loss case), as you are using the SGD, I would consider starting without regularization, as SGD implicitly regularizes (though this is your decision, and if you keep optimizing for a long time, then you should eventually overfit). You can decide if you need to add any regularization (this is could be more of an issue for the logistic case due to the possibility of large weights. Be aware of this).
- (mini-batching) Mini-batching helps. I'd consider doing it. Typically, there are diminishing returns in terms of how much it helps.
- (learning rates, square loss case) With the square loss, I like to set my learning rates pretty large. I actually start by finding where things diverge (for the square loss, it really does diverge. for logistic, weird things happen). Then I usually just cut this down roughly by a factor of 2 and just use that for the remainder of the time. In fact, for the convex case, I then average my parameters (in a way I discuss later in the problem). With averaging, I rarely ever turn down the learning rates.
- (learning rates, logistic loss case) I would consider something similar to the previous tips for the square loss scheme. Some concerns: 1) the weights for logistic tend to become larger than for the square

loss case; it is worth deciding if you need to add regularization. 2) If you are over-parameterizing by using 10 weight vectors (rather than 9) — this was a discussion board question, where I commented that this is usually 'ok' — you might want to check this is not causing you numerical problems (think about why numerical problems might arise due to the over-parameterization). 3) mini-batching helps to make logistic more stable.

Part of the difficulty is that the schemes specified by the theory guarantee convergence, though they are somewhat pessimistic in practice. I'll make a discuss board post more on this point.

2.1 Least Squares [45 points]

Let us optimize the 0/1 loss with the goal of obtaining good classification accuracy (on the test set).

In practice, instead of complete randomization, we often run in *epochs*, where we randomly permute our dataset and then pass through out dataset in this permuted order. Each time we start another epoch, we permute our dataset again. Note that if you mini-batch with with m samples, than one epoch will consist of N/m updates.

Let us plot the losses of two quantities. Let us keep around your parameter vector w_t (your weight vector at iteration t). Let us also track the average weight vector $\overline{w}_\tau$ over the last epoch, i.e. $\overline{w}_\tau$ is the average parameter vector over the τ-th epoch.

Define the total square loss as the *sum* square loss over the 10 classes (so you do not divide by 10 here).

1. *(5 points)* Specify your parameter choices: your learning rate (or learning rate scheme), mini-batch size, and the kernel bandwidth.
2. *(10 points)* You should have one plot showing the both the squared loss after every epoch (starting with your initial squared error). Please label your axes in a readable way. Plot the loss of both w_t and the average weight vector $\overline{w}_\tau$. You shave both the training and test losses on the same plot (so there should be four curves).
3. *(10 points)* For the 0/1 loss, do the same, except start your plots when the 0/1 loss is below 5% so they are more readable. Again, there should be four curves.
4. *(5 points)* What is your final training squared loss and 0/1 loss? What is the total number of mistakes that are made on the training set of your final point?
5. *(15 points)* What is your final training squared loss and 0/1 loss? What is the total number of mistakes that are made on the test set of your final point?

2.2 EXTRA CREDIT: Softmax Classification [20 points]

Now let us address the same problem with the softmax classifier, with the log loss. Partial credit will be given only if you attempt all parts of this problem. Also, use the random Fourier features for this problem.

Use your same choice of the kernel bandwidth as before.

1. *(1 points)* Specify your parameter choices: your learning rate (or learning rate scheme) and mini-batch size.
2. *(4 points)* You should have one plot showing the both the log loss after every epoch (starting with your initial log loss). Please label your axes in a readable way. Plot the loss of both w_t and the average weight vector $\overline{w}_\tau$. You shave both the training and test losses on the same plot (so there should be four curves).

3. *(4 points)* For the 0/1 loss, do the same, except start your plots when the 0/1 loss is below 5% so they are more readable. Again, there should be four curves.
4. *(3 points)* What is your final training log loss and 0/1 loss? What is the total number of mistakes that are made on the training set of your final point?
5. *(8 points)* What is your final training log loss and 0/1 loss? What is the total number of mistakes that are made on the test set of your final point?

If you find it helpful in terms of speed (for plotting purposes), you are free to evaluate the training loss on a random 10K sized subset of the training set (keep this random set fixed). You must train on the full training set though. And, for the final reported losses, please report this on the full training set.

2.3 EXTRA CREDIT: Back to those random Neural Net Features [10 points]

Now let us use the same random features as in HW2, except now we will use $k = 60000$. Let us compare how well we do with these features compared to the Fourier features (we will just use the square loss in this case).

Partial credit will be given only if you attempt all parts of this problem.

1. *(0 points)* Specify your parameter choices: your learning rate (or learning rate scheme), mini-batch size.
2. *(3 points)* You should have one plot showing the both the squared loss after every epoch (starting with your initial squared error). Please label your axes in a readable way. Plot the loss of both w_t and the average weight vector $\overline{w}_\tau$. You shave both the training and test losses on the same plot (so there should be four curves).
3. *(3 points)* For the 0/1 loss, do the same, except start your plots when the 0/1 loss is below 5% so they are more readable. Again, there should be four curves.
4. *(1 points)* What is your final training squared loss and 0/1 loss? What is the total number of mistakes that are made on the training set of your final point?
5. *(3 points)* What is your final training squared loss and 0/1 loss? What is the total number of mistakes that are made on the test set of your final point?

3 SVMs: Hinge loss and mistake bounds [6 Points]

This question is postponed to HW3.

Suppose we classify with the hypothesis $h_w(x) = \mathrm{sgn}(w^\top x)$ where $\mathrm{sgn}(z) = 1$ if z is positive and -1 otherwise. Here we are dealing with binary prediction where each label is in $\{-1, 1\}$. Let us say we make a mistake on x, y with w if $y \neq \mathrm{sgn}(w^\top x)$. The number of mistakes with w on a dataset is the classification loss.

The SVM loss function can be viewed as a relaxation to the classification loss. The *hinge* loss on a pair (x, y) is defined as:

$$\ell((x, y), w) = \max\{0, 1 - yw^\top x\}$$

where $x \in \mathrm{R}^d$ and $y \in \{-1, 1\}$. The SVM attempts to minimize:

$$\frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i w^\top x_i\} + \lambda\|w\|^2$$

where λ is a regularizer (there are different conventions as to how we write the regularizer, e.g. sometimes we write $\frac{\lambda}{n}$ instead of just λ).

The hinge loss provides a way of dealing with datasets that are not separable. Let us go through the argument as to why we view this as a relaxation of finding the max margin classifier.

1. *(2 Points)* Argue that the function $\ell((x,y),w) = \max\{0, 1 - yw^\top x\}$ is convex (as a function of w).
2. *(2 Points)* (Margin mistakes) Suppose that for some w we have a correct prediction of y_i with x_i, i.e. $y_i = \text{sgn}(w^\top x_i)$. What range of values can the hinge loss, $\ell((x_i,y_i),w)$, take on this correctly classified example? Points which are classified correctly and which have non-zero hinge loss are referred to as margin mistakes.
3. *(2 Points)* (Mistake bound) Let $M(w)$ be the number of mistakes made by w on our dataset (in terms of the classification loss). Show that:

$$\frac{1}{n}M(w) \leq \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i w^\top x_i\}$$

   In other words, the average hinge loss on our dataset is an upper bound on the average number of mistakes we make on our dataset.

4 Fitting an SVM classifier by hand [9 Points]

(Source: Murphy text, Exercise 14.1) Consider a dataset with 2 points in 1d:

$$(x_1 = 0, y_1 = -1) \text{ and } (x_2 = \sqrt{2}, y_2 = 1).$$

Consider mapping each point to 3d using the feature vector $\phi(x) = [1, \sqrt{2}x, x^2]^T$ (This is equivalent to using a second order polynomial kernel).

When we examined the perceptron algorithm in class, we related the number of mistakes to the margin of the data. If our data are linearly separable, then a natural method is to try to find a classifier which maximizes the margin. Here, the max margin classifier has the form

$$\hat{\mathbf{w}}, \hat{w_0} = \arg\min ||w||^2 \quad s.t. \tag{1}$$

$$y_1(w^T\phi(x_1) + w_0) \geq 1 \tag{2}$$

$$y_2(w^T\phi(x_2) + w_0) \geq 1 \tag{3}$$

1. *(1 Points)* Write down a vector that is parallel to the optimal vector $\hat{\mathbf{w}}$. Hint: Recall from Figure 14.12 (page 500 in the Murphy text) that $\hat{\mathbf{w}}$ is perpendicular to the decision boundary between the two points in the 3d feature space.
2. *(2 Points)* What is the value of the margin that is achieved by this $\hat{\mathbf{w}}$? Hint: Recall that the margin is the distance from each support vector to the decision boundary. Hint 2: Think about the geometry of the points in feature space, and the vector between them.
3. *(2 Points)* Solve for $\hat{\mathbf{w}}$, using the fact the margin is equal to $1/||\hat{\mathbf{w}}||$.

4. *(2 Points)* Solve for $\hat{w_0}$ using your value for $\hat{\mathbf{w}}$ and Equations 1 to 3. Hint: The points will be on the decision boundary, so the inequalities will be tight. A "tight inequality" is an inequality that is as strict as possible. For this problem, this means that plugging in these points will push the left-hand side of Equations 2 and 3 as close to 1 as possible.

5. *(2 Points)* Write down the form of the discriminant function $f(x) = \hat{w_0} + \hat{\mathbf{w}}^T \phi(x)$ as an explicit function of x. Plot the 2 points in the dataset, along with $f(x)$ in a 2d plot. You may generate this plot by hand, or using a computational tool like Python.

5 K-Means [20 points]

Let us now try out clustering on MNIST. You many cluster with either the projected 50 dimensional dataset or with the full dataset. (Please specify which you used and make sure to project back to "image" space for visualization if you worked in the lower dimensional space).

5.1 Run the algorithm [12 points]

1. *(5 points)* Run the K-means algorithms with $K = 16$. Describe your initialize procedure. Plot the following:
   (a) The squared reconstruction error vs iteration number.
   (b) Let us say that the number of assignments for a mean is the number of points assigned to that mean. Plot the number of assignments for each center in descending order.
   (c) Visualize the 16 centers that you learned, and display them in an order in that corresponds to the frequency in which they were assigned (if you use a grid, just describe the ordering).
2. *(1 points)* Provide a brief interpretation for the $k = 16$ case.
3. *(6 points)* Now let's run the K-means algorithms with $K = 250$ (also keep track of how often each center is assigned). Again, plot the following:
   (a) The squared reconstruction error vs iteration number.
   (b) Let us say that the number of assignments for a mean is the number of points assigned to that mean. Plot the number of assignments for each center in descending order.
   (c) Visualize 16 of these centers, chosen randomly. Display them in the order in an order in that corresponds to the frequency in which they were assigned.

5.2 Classification with K-means [8 points]

We often desire that unsupervised learning helps to discover representations of our data that will be helpful for downstream tasks of interest, such as classification. The extent to which this works in practice varies widely. Let's see how well it works in the context of this problem.

The training algorithm is simple. You will simply find a corresponding label for each center based on the most frequent digit assigned to it. For classification, you will just find the closest center and then use the corresponding label of this center.

1. *(4 points)* For $K = 16$, what are your training and test 0/1 losses?
2. *(4 points)* For $K = 250$, what are your training and test 0/1 losses?

5.3 Your thoughts? [0 points]

(no answer needed) Do you have thoughts on how “unsupervised methods” may bring us to state of the art? Broadly speaking, this is one of the major challenges in ML, how we can use unlabeled data to reduce the burden of labeled data collection (which is often time consuming and costly).