Assignment 2
CSE 446: Machine Learning

University of Washington

0 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 report as a *single* pdf file and submit all your code in a gzipped tarball named `code.tgz`. The report (in a single pdf file) must include all the plots and explanations for programming questions (if required). We highly recommend typesetting your scientific writing using LaTeX. 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). Writing solutions by hand will be accepted provided they are neat; written solutions need to be scanned and included into a single pdf.

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. You may **not**, however, use any machine learning libraries such as Scikit-Learn or TensorFlow. 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.

1 Perceptron Exercise [20 points]

Recall that a perceptron learns a linear classifier with weight vector $\mathbf{w}$. It predicts

$$\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$$

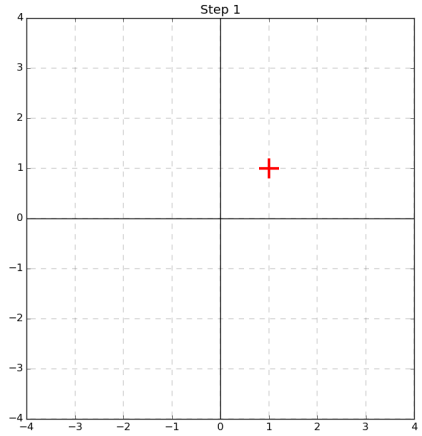
(We assume here that $\hat{y} \in \{+1, -1\}$. Also, note that we are *not* using a bias weight b.) When the perceptron makes a mistake on the nth training example, it updates the weights using the formula

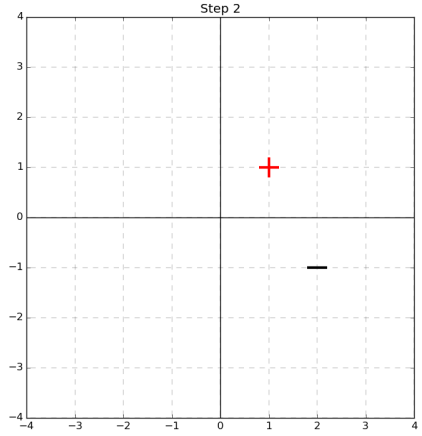
$$\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$$

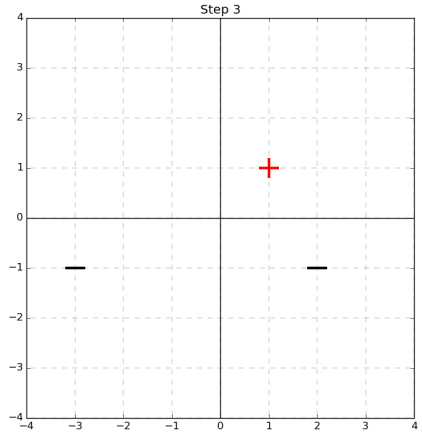
Imagine that we have each $\mathbf{x}_n \in \mathbb{R}^2$, and we encounter the following data points

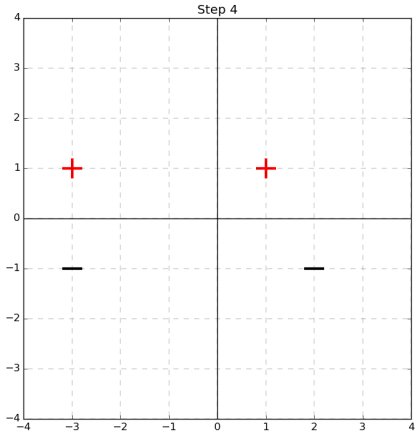
$\mathbf{x}[1]$	$\mathbf{x}[2]$	y
1	1	1
2	-1	-1
-3	-1	-1
-3	1	1

1. Starting with $\mathbf{w} = [0 \quad 0]^\top$, use the perceptron algorithm to learn on the data points in the order from top to bottom. Show the perceptron's linear decision boundary after observing each data point in the graphs below. Be sure to show which side is classified as positive. [10 points]

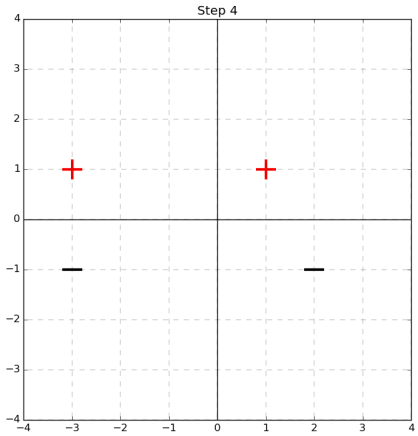

Step 1


Step 2


Step 3



2. Does our learned perceptron maximize the margin between the training data and the decision boundary? If not, draw the maximum-margin decision boundary on the graph below. [5 points]



3. Assume that we continue to collect data and train the perceptron. If all data we see (including the points we just trained on) are linearly separable separable with margin $\gamma = 0.5$ and have maximum norm $\|\mathbf{x}\|_2 \leq 5$, what is the maximum number of mistakes we can make on future data? [5 points]

2 Implementation: Perceptron [70 points]

Implement the perceptron learning algorithm for binary classification, as described in class, from scratch, in Python. All of the files mentioned here are available in the tarball `A2.tgz` available on Canvas.

Implementation details:

- First, read in training data from a single file (we suggest that your program take the filename as a command-line argument) and store it in memory. You can assume for this assignment that you'll never have more than 1000 training examples.
- At the end of each epoch, report the number of mistakes made on that epoch to standard out.
- Test data will always be in a separate file, optionally provided on the command-line. At the end of training, report the number of mistakes on the test data (if there is a test dataset).
- The format for training and testing data will always be one-instance-per-line, formatted as:

  y `tab` $\mathbf{x}[1]$ `tab` $\mathbf{x}[2]$ `tab` ... `tab` $\mathbf{x}[d]$ `newline`

  where `tab` is a single tab character and `newline` is a single newline character.
- If there's other diagnostic output you'd like to generate, send it to standard error. This will make it easier in case we need to run your code.

As a sanity check, train the perceptron on the dataset in `A2.1.tsv`, which has 50 positive and 50 negative examples and is separable by the hyperplane corresponding to weights:

$$\mathbf{w}^* = [-1.2 \quad 2.7 \quad -7.6 \quad -2.8 \quad -2.8]^\top$$

You should be able to converge to a weight vector that achieves zero training-set error in about 20 epochs; your weight vector might not be identical to $\mathbf{w}^*$, of course.

The tarball for this assignment contains eight additional training datasets (`A2.2.train.tsv`, `A2.3.train.tsv`, ..., `A2.9.train.tsv`). Each one has a corresponding test dataset (files `A2.*.test.tsv`). For each dataset [8 points per dataset]:

1. Generate a plot in which the x-axis is the epoch, and the y-axis shows the training error rate in blue and the test error rate in red, at the end of that epoch, where you train on *all of the training data*. We suggest that you train for 1000 or more epochs. You might want to also include a plot that doesn't show all epochs (only earlier ones) so that the "interesting" trends are visible. You might also want to generate a more fine-grained plot that reports training and test error rates after every iteration of the *inner* loop (i.e., after each prediction the perceptron makes on a training instance); don't include such a plot in your writeup unless it shows something interesting we couldn't see in the first plots.
2. Do you believe that the training dataset is linearly separable? If so, give a lower bound on the margin.
3. Do you believe that your learning algorithm overfit the training data? If so, how do you know?
4. Do you suspect that any features are noise? Which ones, and why? (Please index features from 1 to 20 in the order they appear in the data files.)
5. Carve out a development dataset from the training data and use it to tune the perceptron's hyperparameter(s). Report training, development, and test data accuracy, as well as the hyperparameter(s) you tuned and the value you selected for the hyperparameter(s).

Present your answers to the non-plotting questions in a table formatted like Table 1.

Surprise! [6 points] You may not have noticed that test sets 6–8 are identical (but they are). The three corresponding training datasets increase in size from 6 to 7 and 7 to 8, but: (i) they don't

dataset	question number 2	3	4	5
2				
3				
4				
5				
6				
7				
8				
9				

Table 1: Write your answers on part 2 into a table like this one.

overlap and (ii) they do come from the same source. What do you notice when you compare your findings for these three datasets? Can you use this information to get better performance on test set 6 (equivalently, test set 7, equivalently test set 8), still using your perceptron implementation? If so, go for it and give answers to the same questions as above (1–5).

3 Beat the Perceptron [10 points]

Choose one of the datasets (2–9) where the perceptron's test-set performance was not strong, and try to improve using any of the following:

- Decision stump
- K-nearest neighbors
- Voted perceptron

If there are hyperparameters, you should tune them on your development data, **not the test data**.

Tell us which dataset (2–9) you focused on and why. Fully (and concisely) explain the exploration and choices you made as you tuned hyperparameters. Report on how well you were able to perform on the test set, making a fair comparison to the perceptron. It's fine to test more than one of the methods listed above, but you must implement everything yourself and you must report on all of the algorithms you tried (no tuning on test data, even to select the algorithm!). You are only required to test one of them.

4 PCA on digits [40 points]

Let's do PCA and reconstruct the digits in the PCA basis. Download the file mnist.pkl.gz. Load the MNIST dataset in Python as follows.

If you use Python 2:

```
import gzip, pickle
with gzip.open('mnist.pkl.gz') as f:
    train_set, valid_set, test_set = pickle.load(f)
```

If you use Python 3:

```
import gzip, pickle
with gzip.open('mnist.pkl.gz') as f:
    train_set, valid_set, test_set = pickle.load(f, encoding='bytes')
```

The array train_set[0] contains the images, represented as vectors of 784 dimensions. Each example can be reshaped to a matrix of size 28×28. Due to that PCA acts on vectors, you will have to understand the conceptual (and algorithmic) conversion to vectors (and back so you can plot your reconstructions as images).

As you will be making many visualizations of images, plot these images in a reasonable arrangement (e.g. make the images smaller and arrange them in some easily viewable subplot. Points may be deducted for disorganized plots). Also, if you are using the python imshow() command, make sure you understand how to visualize grayscale images. Also, this function, if provided with a matrix of floats, needs the values to be between 0 and 1, so you may need to rescale an image when plotting it so that it lies between 0 and 1 (else the plots could look odd, as imshow() drops values out of this range).

4.1 Covariance matrix construction and the advantages of matrix operations [13 points]

Let d be the number of dimensions and n be the number of samples. Suppose X is your data matrix of size $n \times d$. Let μ be the mean (column) vector of your data. Let us now compute the covariance matrix of the dataset. Throughout this problem you will use the (entire) training set.

1. *(1 points)* Choose 10 digits (make sure they are from different classes). Visualize these digits. In particular, include figures of these 10 digits.
2. *(1 points)* What are dimensions of the covariance matrix? What is the dimension of the mean vector μ?
3. *(1 point)* Visualize the mean image (and provide a figure).
4. *(3 points)* Let x_i be a vector representing the i-th data point. Let us write out two methods for computing the covariance matrix:
   - vector based method: Write out the covariance matrix, Σ, in terms of the x_i's and μ (and not the matrix X). The method should have a sum over i in it.
   - matrix based method: Now, in matrix algebra notation, write out the covariance matrix as a function of the matrix X and the (column) vector μ (and there should be no sum over i).
5. *(5 points)* (Appreciating linear algebra) Compute the covariance matrix using two different methods. You must measure the run-time of both of the following methods:

- the vector based method: Compute the covariance matrix using your vector based algorithm, from the previous question. Note that this will involve a for loop over all 50K data points (and you may want to use the python linear algebra function 'outer' to compute the outer product, as the transpose operation does not work on arrays). This method may take a little time if you are running it on your laptop.
- the matrix based method: Compute the covariance matrix using matrix multiplications, as you wrote out earlier.

Measure the runtime using a timing procedure in Python (not your own clock). List what time function you used. The time must reflect the time of just the compute operations and it should *not* reflect the time to load the data matrices (as you should have these already loaded in memory before your code starts the timer). Check that these two matrices give you the same answer. In particular, report the average absolute difference $\frac{1}{d^2}\sum_{i,j}|\Sigma_{i,j}^{\text{vec. method}} - \Sigma_{i,j}^{\text{mat. method}}|$ (which should be numerically near to 0). How long did each method take? What is the *percentage* increase in runtime?

6. *(2 points)* (Looking under the hood) As you used Python and a standard linear algebra library (e.g. Numpy), both of these two methods were implemented on your computer with the same underlying number of primitive mathematical operations: under the hood, they both were implemented with the same number of additions of (scalar) floating point numbers and same number of multiplications of (scalar) floating point numbers[1]. This leads to the puzzling question: why was your matrix based method (using matrix multiplications) was so much faster than your for-loop based, vector code? A concise answer, hitting the important reason, is sufficient. You are free to do a google search to find the answer. (Hint: had the question asked you to code the vector based method in the C or $C++$ languages, you would not have observed much of a difference.)
   Implications: this comparison has important implications for practical machine learning. Matrix-algebra structured algorithms are the bread and butter of modern machine learning for good reason. It is worth considering having linear algebra skills at your fingertips. It may also give you an appreciation of the value of GPU processers, which speed up matrix multiplication and other basic matrix algebra methods (such as FFTs) by factors between 10 and 100 times; the bottleneck of GPUs is often just copying data onto them, in blocks.

4.2 Eigen-Analysis [13 points]

Let $\Sigma = UDU^\top$ be the SVD of Σ.

1. *(3 points)* What are the eigenvalues λ_1, λ_2, λ_{10}, λ_{30}, and λ_{50}? Also, what is the sum of eigenvalues $\sum_{i=1}^{d}\lambda_i$?

[1] In theory, and rather surprisingly, the matrix multiplication procedure can actually be implemented with a *smaller* number of float additions and float multiplications than that which is used by the vector based approach. One such algorithm is the famous Strassen's algorithm, which lead to the (nearly the theoretically fastest) Coppersmith-Winograd algorithm. However, due to constant factors and realistic modern architecture constraints, these theoretically faster methods are rarely used; instead, the naive brute force approach to matrix multiplication is that which is under the hood in linear algebra packages.

2. *(4 points)* It is not difficult 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 100. (so the X-axis is k and the Y-axis is the fractional reconstruction error).
3. *(2 points)* Roughly, what eigenvector captures 50% of the variance? 80% of the variance?
4. *(3 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). Display the first 10 eigenvectors as images.
5. *(1 points)* Provide a brief interpretation of what you think they capture.

4.3 Pseudocode for Reconstruction [5 points]

This problem will help you to implement your dimensionality reduction algorithm and your reconstruction algorithm with matrix algebra (and avoid writing a for-loop over all the data points). Throughout this problem, assume you will project each image down to k dimensions and that you want to obtain the best reconstruction on this dataset (in terms of the squared error, as discussed in class and in CIML).

(Hint: In this problem, it may also be helpful to define an n-dimensional (column) vector of all 1's and to denote this vector by $\vec{1}$. Note that $\vec{1}\mu^\top$ is an $n \times d$ matrix)

1. *(1 points)* The SVD will give you d eigenvectors, each corresponding to some eigenvalue λ_i. Which of these vectors (and how many of them) will you use?
2. *(2 points)* Specify a matrix $\widetilde{U}$ which is of of size $d \times k$ that is constructed out of your chosen eigenvectors. Write a matrix algebra expression that reduces the dimension of all the points in your dataset. This expression should give you a matrix of size $n \times k$, containing all of your dimensionality reduced data. Your expression should use matrix algebra and should be stated in terms of X, μ, and $\widetilde{U}$ (this expression should have no sums in it).
3. *(2 points)* Now you should have a data matrix of size $n \times k$, which consists of all your data after the dimensionality has been reduced. Write out an expression that will give you a reconstruction of all your data. In particular, you should recover a matrix of size $n \times d$. Your method should be stated using matrix algebra (and should have no sums in it). Be sure that your expression also adds the mean back into your data, else the reconstruction will not be correct.

4.4 Visualization and Reconstruction [9 points]

1. *(3 points)* Now code your matrix algebra method that reconstructs X after it has projected it down to k dimensions. Do this for all 50K datapoints. For $k = 30$, time this procedure. How long does it take to project and reconstruct all 50K datapoints?
2. *(5 points)* Now let us visualize these reconstructed digits for different values of k. Visualize these reconstructions for all of the following cases (using the same 10 digits you visualized earlier).

   (a) Do this for $k = 3$.
   (b) Do this for $k = 5$.
   (c) Do this for $k = 10$.

(d) Do this for $k = 25$.
(e) Do this for $k = 50$.
(f) Do this for $k = 200$.

3. *(1 point)* Provide a brief interpretation, in terms of your perceptions of the quality of these reconstructions.

5 Conditional Probability Review: Revisiting the Bayes Optimal Classifier [8 points]

In class and in CIML, it is argued that, for binary prediction, the classifier defined by $f^{(\text{BO})}(x) = \text{argmax}_y \mathcal{D}(x, y)$ achieves the minimal expected classification error among all possible classifiers (CIML provides a proof). This classifier is referred to as the "Bayes Optimal Classifier"; the classifier must know the underlying distribution $\mathcal{D}$.

1. *(6 points)* Show that an equivalent way to write this classifier is $f^{(\text{BO})}(x) = \text{argmax}_y \mathcal{D}(y|x)$.
2. *(2 points)* Give an intuitive reason (in words rather than with mathematics) why this classifier, written in terms of the conditional probability, is unbeatable.

6 Bonus: Dimensionality Reduction in Learning [up to 10 extra points]

For the dataset you considered in part 3, reduce its dimensionality using PCA and use the resulting features instead of the original ones, with both the perceptron and the other algorithm(s) you implemented. For maximal extra points, you must give a compelling argument that PCA offered some benefit (and explain the benefit), or that it could not help. Be sure to explain how you chose k. Plots that help make your argument more clear are encouraged.

7 Bonus: The Bayes Optimal Hypothesis for "Regression" [up to 10 extra points]

Suppose now that y is real valued. The square loss of a hypothesis f, where f maps an input to a real value, is defined as:

$$\epsilon(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}(y - f(x))^2 .$$

Prove that the Bayes optimal hypothesis for the square loss (i.e. the hypothesis that achieves the minimal possible square loss among all possible hypothesis) is:

$$f^{(\text{BO})}(x) = \mathbb{E}[y|x]$$

(where the conditional expectation is with respect to $\mathcal{D}$). You may prove this assuming that the number of possible x and y are finite, if that makes things conceptually easier for you.