Homework 2
CSE 446: Machine Learning

University of Washington

1 Policies [0 points]

Please read these policies. **Please answer the three questions below and include your answers marked in a "problem 0" in your solution set.** Homeworks which do not include these answers will not be graded.

Gradescope submission: When submitting your HW, please tag your pages correctly as is requested in gradescope. Untagged homeworks will not be graded, until the tagging is fixed.

Readings: Read the required material.

Submission format: Submit your report as a *single* pdf file. Also, please include all your code in the PDF file in a section at the end of your document, marked "Code"; also specify which problem(s) the code corresponds to. The report (in a single pdf file) must include all the plots and explanations for programming questions (if required). Homework solutions must be organized in order, with all plots arranged in the correct location in your submitted solutions. We highly recommend typesetting your scientific writing using LaTeX(see the website for references for free tools). 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.

Including your Python source code: Updated policy for Jupyter. For the programming assignments, submit your code in the pdf file along with a neatly written README file that instructs us how you ran your code with different settings (if applicable). Please note that we will not accept screenshots of Jupyter notebooks. If you do use Jupyter, you must export your code to a text file and put the text of your code in the submitted pdf file (in the last section) in a manner that can be executed in that order (without any extraneous or missing code).

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, TensorFlow, or PyTorch, unless explicitly specified for that question. If in doubt, post to the message boards.

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.

1.1 List of Collaborators

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

1.2 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".

1.3 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.

2 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.

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

Throughout this problem you will use the **entire** training set, consisting of 50K images.

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).

2.1 Covariance matrix construction and the advantages of matrix operations *[14 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 entire problem, you must write your answers where vectors refer to *column* vectors. So a data point x_i refers to a d-dimensional (column) vector.

Hint: 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 point]*** Choose 10 digits (make sure they are from different classes). Visualize these digits. In particular, include figures of these 10 digits.
2. ***[1 point]*** 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. Let x_i be a vector representing the i-th data point. Let us write out two methods for computing the covariance matrix:
   - ***[2 points]*** 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.

- ***[2 points]*** 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. (Appreciating linear algebra) Compute the covariance matrix using two different methods. Now you must measure the runtime of both procedures using a timing procedure in Python (not your own clock). 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). 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.
   - ***[4 points]*** List what time function you used. How long did each method take, and what is the *percentage* increase in runtime? Check that these two matrices give you the same answer. In particular, report the average absolute difference $\frac{1}{d^2}\sum_{i=1}^{d}\sum_{j=1}^{d}|\Sigma_{i,j}^{\text{vec. method}} - \Sigma_{i,j}^{\text{mat. method}}|$ (which should be numerically near to 0).

6. (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].

   - ***[2 points]*** This leads to the puzzling question: why was your matrix based method (using matrix multiplications) 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

[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 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.

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.

2.2 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 $\sum_{i=1}^{d} \lambda_i$?
2. ***[4 points]*** It is not difficult to see that the fractional reconstruction error of using the top k out of d directions is $1 - \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$. (Note that this error is 0 when $k = d$.) 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, for what k is the fractional reconstruction error less than 0.5? And for what k is this error less 0.2? (An answer to this is not required: Do you see why you can interpret this k as the projected dimension which captures 50% and, respectively, 80% of the total variation in your data?)
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 point]*** Provide a brief interpretation of what you think they capture.

2.3 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 point]*** 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 be stated in terms of X, μ, and $\widetilde{U}$, and 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.

2.4 Visualization and Reconstruction *[8 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 did your code 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). Include figures of all the reconstructed images in all of the following cases.

   (a) Do this for $k = 5$.
   (b) Do this for $k = 15$.
   (c) Do this for $k = 40$.
   (d) Do this for $k = 100$.

(No answer needed: Feel free to think about the subjective reconstruction quality vs. the dimensionality you used.)

3 Binary Classification with Linear Regression on MNIST *[30 points]*

This question requires the "mnist_2_vs_9.gz" dataset and can be downloaded from the website. Load the MNIST 2 vs 9 dataset in Python as follows.

```
import gzip, pickle
with gzip.open("mnist_2_vs_9.gz") as f:
    data = pickle.load(f, encoding="bytes")
    Xtrain, Ytrain, Xtest, Ytest, Xdev, Ydev = \
        data[b"Xtrain"], data[b"Ytrain"], data[b"Xtest"], \
        data[b"Ytest"], data[b"Xdev"], data[b"Ydev"]
```

In binary classification, we restrict Y to take on only two values. Suppose $Y \in \{0, 1\}$ rather than $Y \in \{-1, 1\}$. Let us consider the classification problem of recognizing if a digit is a "2" or a "9"; $y = 1$ is the label corresponding to class "2" and $y = 0$ corresponds to the class "9".

Throughout this problem, all vectors (in written form) should refer to column vectors, by default. $\mathbf{x}_n$ represents a vector of the pixel intensities of the image. If a row vector is needed, then use a transpose. Whenever we write X, Y or $\widehat{Y}$, let us assume that they have the following sizes: $X \in \mathbb{R}^{N\times d}$, $Y \in \mathbb{R}^{N\times 1}$, $\widehat{Y} \in \mathbb{R}^{N\times 1}$.

The Bias Term: Be sure to include a "bias" term in this problem, else you will have significantly worse results. You can do this by adding an additional column (of all ones) to the train, test, and dev matrices. By adding an additional column, you will not need to explicitly write your code with an extra parameter.

3.1 Linear Regression, using the Closed Form Estimator *[12 points]*

Now let us use least squares linear regression for classification. Define the objective function (the cost function):

$$\mathcal{L}_\lambda(\mathbf{w}) := \frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}(y_n - \mathbf{w}^\top \mathbf{x}_n)^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

This can be equivalently written as:

$$\mathcal{L}_\lambda(\mathbf{w}) = \frac{1}{N}\frac{1}{2}\|Y - X\mathbf{w}\|^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

(this form helps us compute the loss quickly). The optimization problem we seek to solve here is:

$$\underset{\mathbf{w}}{\operatorname{argmin}}\,\mathcal{L}_\lambda(\mathbf{w})$$

and the solution is:

$$\mathbf{w}_\lambda^* = \left(\frac{1}{N}X^\top X + \lambda \mathbb{I}_d\right)^{-1}\left(\frac{1}{N}X^\top Y\right).$$

where $\mathbb{I}_d$ denotes the $d \times d$ identity matrix. (See the class notes and CIML. One can also directly derive this through differentiation.)

For the purposes of classification using the square loss, we can use the following rule: label a digit as a "2", i.e. $y = 1$, if $\mathbf{w} \cdot \mathbf{x} \geq 1/2$ (You should be able to see why $\frac{1}{2}$ is a reasonable threshold to use.).

1. ***[3 points]*** Let's try out the vanilla least squares estimator. This corresponds to using $\lambda = 0$. What happened? If something went wrong, give a brief description of what happened and give a compelling reason as to what is causing it on this particular dataset (Hint: as you have visualized these digits, you can actually provide a very specific reason.)
2. Now choose an appropriate value of λ, based on trying out a few different values. Report:
   (a) ***[1 point]*** your choice of λ.
   (b) ***[3 points]*** your average squared error, i.e. $\frac{1}{N}\frac{1}{2}\|Y - X\mathbf{w}\|^2$, on the training set, the dev set, and test set (you should have X and Y correspond to the training set, the dev set, and test set, respectively).
   (c) ***[4 points]*** Report your misclassification error (as a percentage) on the training set, the dev set, and test set.

Remark: While our decision rule said to use a $1/2$, you are free to tune this threshold on the dev set (which is often done in practice); you might note that this leads to a drop in error. While not necessary, you are welcome to use a different threshold if you prefer to report a lower error. Please make a note in your report if you used a different threshold.

3. ***[1 point]*** Why might linear regression be a poor idea for classification?

3.2 Linear regression using gradient descent *[18 points]*

It is convenient to define:

$$\widehat{y}_n = \mathbf{w} \cdot \mathbf{x}_n \, .$$

We can interpret $\widehat{y}_n$ as our prediction based on the n-th input due to using $\mathbf{w}$; let us keep in mind that $\widehat{y}_n$ implicitly depends on $\mathbf{w}$.

1. ***[4 points]*** Show that:

$$\frac{d\mathcal{L}_\lambda(\mathbf{w})}{d\mathbf{w}} = \frac{-1}{N}\sum_{n=1}^{N}(y_n - \widehat{y}_n)\mathbf{x}_n + \lambda\mathbf{w} \, .$$

If you are not comfortable taking the derivative directly respect to the vector $\mathbf{w}$, you are free to instead take the derivative with respect to each of the d components, $\mathbf{w} = (w_1, w_2, \ldots w_d)^\top$, of the vector $\mathbf{w}$. You should be able to see that the gradient $\frac{d\mathcal{L}_\lambda(\mathbf{w})}{d\mathbf{w}}$ is simply a vector of these d component wise derivatives.

Remark: You may gain intuition for gradient descent by interpreting it as a certain "error driven" update based on the form of this gradient.

2. ***[4 points]*** In order to make our code faster, simplify this gradient expression, by removing the "sum over n", with matrix algebra by expressing it in terms of $X \in \mathbb{R}^{N\times d}$, $Y \in \mathbb{R}^{N\times 1}$, $\widehat{Y} \in \mathbb{R}^{N\times 1}$ (and other relevant quantities).
3. Now run gradient descent:

   (a) ***[1 point]*** What stepsize do you find works well, and what value of λ did you use? You might have to search around a little (it helps to search by trying things out like 1 and appropriately searching up or down in multiples of 10).

   (b) ***[4 points]*** Make a plot showing your training averaged squared error and your development averaged squared error on the y-axis and the iteration on the x-axis. Both curves should be on one same plot.

   Remark: When we run gradient descent (or stochastic gradient descent) descent, there is a sense in which the algorithm itself performs regularization. In this particular setting, you might note that things to do not seem do diverge even if $\lambda = 0$ and may work just fine. This is a form of "early stopping"; often, when prevent our algorithm from running "too long" we implicitly control the model complexity.

   (c) ***[3 points]*** Make this plot again (with both curves), except use the misclassification error, as a percentage, instead of average squared error. Here, make sure to start your x-axis at a slightly later iteration (past the first iteration), so that your error starts below 5%, which makes the behavior more easy to view (it is difficult to view the long run behavior if the y-axis is over too large a range).

   (d) ***[2 points]*** What is the lowest train and dev misclassification % error that you achieved? What is your test error for the corresponding model that obtained the smallest dev error? It is expected that you obtain a good dev error (meaning you train long enough and you regularize appropriately, if needed). You should be able to get a sense of how long you should run your code.

4 Maximum Likelihood Estimation *[12 points]*

This question uses a discrete probability distribution known as the Poisson distribution. A discrete random variable X follows a Poisson distribution with parameter λ if

$$p(X = k) = \frac{\lambda^k}{k!} e^{-\lambda} \qquad \forall k \in \{0, 1, 2, \dots\}$$

You work for the city of Seattle picking up ballots from ballot dropoff boxes around town. You visit the box near UW every hour on the hour and pick up the ballots that have been left there. Here are the number of ballots you picked up this morning, starting at 1am.

time	1am	2am	3am	4am	5am	6am	7am	8am
new ballots picked up	6	4	2	7	5	1	2	5

Let $\boldsymbol{G} = \langle G_1, \dots, G_N \rangle$ be a random vector where G_n is the number of ballots picked up on iteration n.

1. ***[4 points]*** Give the log-likelihood function of $\boldsymbol{G}$ given λ.
2. ***[4 points]*** Compute the MLE for λ in the general case.
3. ***[4 points]*** Compute the MLE for λ using the observed $\boldsymbol{G}$.

5 Bayes Optimal Prediction: Classification *[4 points]*

In class and in CIML, it is argued that, for binary prediction, the classifier defined by $f^{(\text{BO})}(x) = \operatorname{argmax}_y \mathcal{D}(x, y)$ 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. ***[4 points]*** Show that an equivalent way to write this classifier is $f^{(\text{BO})}(x) = \operatorname{argmax}_y \mathcal{D}(y|x)$. (Answer not needed: Feel free to think of an intuitive reason (in words rather than with mathematics) as to why this classifier, written in terms of the conditional probability, is unbeatable.)

6 Code

Please include all your code in the PDF file in this section. Specify which problem(s) the code corresponds to. Re Jupyter: refer to the policies section of the HW.