

Assignment 3

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.

EXTRA CREDIT POLICY: Before you do the extra credit problems, do all the regular questions. Extra credit points will only be awarded if there are answers provided on all the regular questions. Also, credit will be given for an extra credit question only if you honestly attempt the question (e.g. don’t only answer a few ‘interpretation’ questions in an extra credit question.)

Readings: Read the required material.

Submission format: Submit your report as a *single* pdf file and submit all your code in separate files, corresponding to separate problems, in a gzipped tarball named `code_problem#.tgz`. For example, `code_problem2.tgz` should contain all the code you used to solve problem 2. 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 L^AT_EX. Some free tools that might help: ShareLaTeX (www.sharelatex.com), TexStudio (Windows), MacTex (Mac), TexMaker (cross-platform), and Detexify² (online). If you want to type, but don’t know (and don’t want to learn) L^AT_EX, 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.

Comments

Please check back for updates with clarifications/typo fixes (announcements will be made on canvas).

Instructions for how to open data files, along with coding tips, will be posted to the Discussion board on canvas.

1 Binary Classification with Linear Regression on MNIST [50 points]

Grab the "mnist.2_vs_9.gz" dataset on Canvas.

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 \hat{Y} , let us assume that they have the following sizes: $X \in \mathbb{R}^{n \times d}$, $Y \in \mathbb{R}^{n \times 1}$, $\hat{Y} \in \mathbb{R}^{n \times 1}$.

Important: 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.

Remark: If, like the instructor, you prefer an easy hack if you think it will work, you could simply just overwrite the last column to be all ones, e.g. “Xtrain[:,783]=1”, “Xdev[:,783]=1”, “Xtest[:,783]=1”. This just overwrites some pixel in the corner of the image with a one; we not expect that this single pixel would be that helpful anyways. It should be clear that this is not the best practice.

1.1 Linear Regression, using the Closed Form Estimator [15 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 $\mathbf{w} \cdot \mathbf{x} \geq 1/2$ (do you see why $\frac{1}{2}$ is a reasonable threshold?).

- (4 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.)
- (10 points) Now choose an appropriate value of λ , based on trying out a few different values. Report:
 - (1 point) your choice of λ .

- (b) (4 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) (5 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 notable drop in error. While not necessary, you are welcome to use a different threshold if you prefer to report a lower error.

- 3. (1 point) Why might linear regression be a poor idea for classification?

1.2 Linear regression using gradient descent [20 points]

It is convenient to define:

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

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

- 1. (4 points) Show that:

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

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, $\mathbf{w} = (w_1, w_2, \dots, w_d)^\top$, of the vector \mathbf{w} . You should be able to see that the derivative $\frac{d\mathcal{L}_\lambda(\mathbf{w})}{d\mathbf{w}}$ 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 $X \in \mathbb{R}^{n \times d}$, $Y \in \mathbb{R}^{n \times 1}$, $\hat{Y} \in \mathbb{R}^{n \times 1}$ (and other relevant quantities).
- 3. (12 points) Now run gradient descent:
 - (a) (2 points) What stepsize do you find works well? 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) (5 points) Make a plot showing your training squared error, your development squared error, and your test squared error on the y -axis and the iteration on the x -axis. All three curves should be on one same plot. What value of λ did you use?

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 do not seem to diverge even if $\lambda = 0$ and may actually work just fine. This is a form of “early stopping”; often, when we prevent our algorithm from running “too long” we implicitly control the model complexity.

- (c) (5 points) Make this plot again, except use the misclassification error, as a percentage, instead of 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). What is the lowest test misclassification % error that you can achieve? It is expected that you obtain a good test error (meaning you train long enough and you regularize appropriately, if needed).

1.3 Linear regression using stochastic gradient descent [15 points]

1. (15 points) Now run stochastic gradient descent, using one point at a time:
 - (a) (3 points) Roughly, what is the stepsize at which SGD starts to diverge? Why would you expect it to diverge at too large a learning rate?
 - (b) (6 points) After every 500 updates (starting before your first update at 0), make a plot showing your training squared error, your development squared error, and your test squared error on the y -axis and the iteration on the x -axis. All three curves should be on one same plot. What value of λ did you use? Specify your learning rate scheme if you chose to decay your learning rate.
 - (c) (6 points) Make this plot again, except use the misclassification error, as a percentage, instead of squared error. Here, make sure to start your x -axis at a slightly later 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). Again, it is expected that you obtain a good test error (meaning you train long enough and you regularize appropriately, if needed).

1.4 EXTRA CREDIT: Mini-batch, stochastic gradient descent [10 points]

Again, due to the manner in which matrix multiplication methods (as opposed to using “For Loops”) allow for faster runtimes (often through GPU processors), it is often much faster to use “mini-batch” methods by sampling m points at a time. By increasing the batch size, we reduce the variance in stochastic gradient descent. In practice (and in theory), this tends to be very helpful as increase m , and then there tends to be (relatively sharp) diminishing returns.

1. Now run stochastic gradient descent, using a mini-batch size of $m = 100$ points at a time. Here, each parameter updates means you use $m = 100$ randomly sampled training points.
 - (a) (1 point) Roughly, what is the stepsize at which SGD starts to diverge? You might find it interesting that this stepsize is different than the $m = 1$ case.
 - (b) (4 points) After every 500 updates (starting before your first update 0), make a plot showing your training squared error, your development squared error, and your test squared error on the y -axis and the iteration on the x -axis. All three curves should be on one same plot. What value of λ did you use (if you used it)? Specify your learning rate scheme if you chose to

decay your learning rate.

Remark Note that every update now touches 100 points. However, an update should not be 100 times slower (even though, technically, your computer is doing 100 as much computation). This is, again, due to how matrix multiplication is implemented.

- (c) (4 points) Make this plot again, except use the misclassification error, as a percentage, instead of squared error. Here, make sure to start your x -axis at a slightly later 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). Again, it is expected that you obtain a good test error (meaning you train long enough and you regularize appropriately, if needed).
- 2. (1 points) Comment on how your plots differ from using SGD with $m = 1$.

1.5 EXTRA CREDIT: Using polynomial features! [10 points]

Now let us “blow up” the dimensionality by including all the quadratic interactions along with the linear terms. Let us understand this by example. Suppose x is three dimensional, i.e. $x = (x[1], x[2], x[3])$. Define a new feature vector as follows:

$$\phi(x) = (1, x[1], x[2], x[3], x[1]^2, x[2]^2, x[3]^2, x[1]x[2], x[1]x[3], x[2]x[3]).$$

The first four coordinates above are considered the “linear” terms (and a bias term).

- 1. (optional) What is the dimensionality of $\phi(x)$ for a d -dimensional input x ?
- 2. (2 points) Crudely (just get the right order of magnitude), if we did this on our images (of dimensionality $d = 784$), what would be the dimensionality of the original terms plus the quadratic interactions terms? Why might this be a poor idea?
- 3. (1 point) Instead, let us consider using building our feature vector using the $d = 40$ reduced dimension. Why is this a better idea?

Construction: Obtain the dataset from Canvas with 40 dimensional PCA features. In this dataset, the instructor overwrote the last PCA coordinate with a one. You will see why this is a nice hack in a moment. Now make a new dataset to see what our classification error is with these quadratic plus linear interaction terms. You can easily build a dataset of $40^2 = 1600$ features, which includes a bias term, a linear term, and all the quadratic terms. By simply using the original 40 PCA features (with the instructor’s hack baked in), you can do two four loops suffices to build all the features. Do you see how to do this and why this get’s you the bias, the linear term, and the quadratic terms? I don’t mind a discussion of this in the discussion board, and I don’t mind if someone posts this feature construction code to the discussion board.

- 4. (7 points) Now run the algorithm of your choice (along with any regularization you deem appropriate). Report your squared training error, your dev error, and your test error. Also report the corresponding misclassification errors.

Remark: You can see where this is going.... We could certainly try cubic features, etc. If you start on this path, regularization becomes important as does having fast algorithms.

2 Binary Classification with Logistic Regression [10 Points]

Recall the probabilistic model:

$$\begin{aligned}p_{\mathbf{w}}(y = 1 \mid \mathbf{x}) &= \frac{1}{1 + \exp(-\mathbf{w} \cdot \mathbf{x})} \\p_{\mathbf{w}}(y = 0 \mid \mathbf{x}) &= 1 - p_{\mathbf{w}}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w} \cdot \mathbf{x})}\end{aligned}$$

Our objective function here is:

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

Now let us minimize our cost.

It is helpful to define:

$$\hat{y}_n = p_{\mathbf{w}}(y_n \mid \mathbf{x}_n)$$

You can view this as a probabilistic prediction. This definition will help in allowing you to easily modify your previous code.

1. (3 points) Show that:

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

Remark: You might find this expression to be rather curious! It looks identical to the expression for our gradient in the squared error case. You are free to think about why this was a fortunate coincidence. The choice of $y \in \{0, 1\}$ was indeed intentional.

2. (1 point) Again, in order to make our code fast, simplify this gradient expression with matrix algebra by expressing it in terms of $X \in \mathbb{R}^{n \times d}$, $Y \in \mathbb{R}^{n \times 1}$, $\hat{Y} \in \mathbb{R}^{n \times 1}$ (and other relevant quantities).
3. (6 points) Let us understand a few properties of logistic regression.
 - (a) Suppose our training data are linear separable and $\lambda = 0$. What does our weight vector converge to? Why?
 - (b) Suppose now that $d \geq n$, $\lambda = 0$, and our n data points are all linearly independent. What does our weight vector converge to? Why?
 - (c) In both of these cases, why does regularization or “early stopping” make sense? Consider the implications for your true error in your answer.

2.1 EXTRA CREDIT: Try it out! [15 points]

Implement logistic regression on our “2” vs “9” dataset. Note: here you need to modify the decision rule: we should label a digit as a “2”, i.e. $y = 1$, if $\mathbf{w} \cdot \mathbf{x} \geq 0$ (do you see why?).

1. Now run GD or SGD (with your choice or batch size).
 - (a) (1 point) Specify *all* your parameter choices (this must include m , λ , the step size scheme if you decay it).
 - (b) (6 points) Show your log loss on the training set, the development set, and the test set (where the y -axis is the log loss and x -axis is the iteration). You should be able to convince yourself that the log loss (sometimes referred to as the cross entropy) can be computed as:

$$\frac{-1}{N} \sum_n (y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)) ,$$

which can be directly computed in python with operations on the vectors Y and \hat{Y} . All three curves should be on the same plot. What value of λ did you use? Specify your learning rate scheme if you chose to decay your learning rate.

- (c) (3 points) Make this plot again, except use the misclassification error, as a percentage, instead of the squared error. Here, make sure to start your x -axis at a slightly later 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). Again, it is expected that you obtain a good test error (meaning you train long enough and you regularize appropriately, if needed).
2. (5 points) Now try out using the quadratic interaction features of the PCA projections from before. Specify the algorithm you used along with your parameter choices (you do not need to provide plots). What misclassification % error did you achieve on the training set, the dev set, and test set?

3 Multi-Class Classification using Least Squares [20 Points]

The MNist dataset, <http://yann.lecun.com/exdb/mnist/>, has been historically interesting. Also, much of the earlier focus on certain methods and algorithms in machine learning was partly due to obtaining good results on this dataset. If you work on the extra credit problem later on, you will gain a little more perspective on how many of these issues are not so relevant, once we start having a much more “flexible” representation.

Let us now build a classifier for digit recognition on all 10 digits.

3.1 “One vs all classification” with Linear Regression [15 Points]

In the previous two class problem, we used linear regression with $y \in \{0, 1\}$. Now we have 10 classes. Here we will use a “one-hot” encoding of the label. The label y_n will actually be a 10 dimensional vector, where the k -th entry is 1 if the label is for the k -th class and all other entries will be 0.

1. (0 points) Create a label matrix of size $Y \in \mathbb{R}^{n \times 10}$ for both your training set and your test set.

Here, we can consider a vector valued prediction:

$$\hat{y}_n = W^\top \cdot \mathbf{x}_n.$$

where sized $W \in \mathbb{R}^{d \times 10}$ matrix.

As discussed in class, we can define the objective function here as:

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

where you can view the penalty as the sum of the squares of the entries of W .

Note that this formulation is literally the same as doing k -binary classification problems on each of the classes separately, you will do a linear regression where you label a digit as $Y = 1$ if and only if the label for this digit is k (for $k = 0, 1, 2, \dots, 9$).

It is straightforward to verify that the solution is:

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

Note that here are stacking our the vectors y_n and \hat{y}_n into the matrices $Y \in \mathbb{R}^{n \times 10}$ and $\hat{Y} \in \mathbb{R}^{n \times 10}$.

For classification, you will then take the largest predicted score among your 10 predictors.

Remark: This is sometimes referred to as “one against all” prediction. Note that we are actually just doing 10 separate linear regressions and are stacking our answers together.

Also, the derivative of this loss function can be expressed as:

$$\frac{d\mathcal{L}_\lambda(\mathbf{w})}{dW} = \frac{-1}{N} \sum_{n=1}^N \mathbf{x}_n (y_n - \hat{y}_n)^\top + \lambda W. \quad (1)$$

Note that this expression is of size $d \times k$.

Dataset You will use the MNIST dataset you used from the last assignment. It contains all 10 digits with the labels. The instructor will post a function on Canvas for computing the misclassification % error that you are free to use.

1. (3 points) Based on the above gradient expression, write out the matrix algebra expression for this derivative in terms of $X \in \mathbb{R}^{n \times d}$, $Y \in \mathbb{R}^{n \times 10}$, $\hat{Y} \in \mathbb{R}^{n \times 10}$ (and other relevant quantities), where there is no “sum over n ” in your expression.
2. (12 points) Decide which method you would like to use: the closed form expression, GD, or SGD. Specify your method along with all your parameters. On the training set, dev set, and test set, what are your square losses and what is your misclassification % error?

3.2 EXTRA CREDIT: Take a matrix derivative on your own [5 points]

Prove equation 1. To do this, lookup some facts about matrix derivatives on the internet (there are all sorts of “matrix cookbooks”, “cheat sheets”, etc. out there). Provide the one or two rules for how one takes a matrix derivative to obtain the proof. The proof should be just a few steps.

Also, you should be able to convince yourself as to how this follows from the vector proof you did earlier.

3.3 EXTRA CREDIT: Let's not forget about the softmax! [10 Points]

We now turn to the softmax classifier. Here, y takes values in the set $\{1, \dots, k\}$. The model is as follows: we have k weight vectors, $w^{(1)}, w^{(2)}, \dots, w^{(k)}$. For $\ell \in \{1, \dots, k\}$,

$$p_W(y = \ell | x) = \frac{\exp(w^{(\ell)} \cdot x)}{1 + \sum_{i=1}^k \exp(w^{(i)} \cdot x)}$$

Again, note that this is a valid probability distribution (the probabilities are positive and they sum to 1). Also, note that we have “over-parameterized” the model, since:

$$p_W(y = k | x) = 1 - \sum_{i=1}^{k-1} p_W(y = i | x)$$

We could actually could define the model without using $w^{(k)}$. However, the instructor likes this choice as the derivative expressions become a little simpler (and it makes it easier to re-use code).

As before, it is helpful to define the “prediction vector”:

$$\hat{y}_n = p_W(y | \mathbf{x}_n)$$

where we view $p_W(y | \mathbf{x}_n)$ as k -dimensional (column) vector where the i -th component is $p_W(y = i | x_n)$.

As before, the (negative) likelihood function on an N size training set is:

$$\mathcal{L}_\lambda(W) = \frac{-1}{N} \sum_{n=1}^N \log p_W(y = y_n | \mathbf{x}_n) + \frac{\lambda}{2} \|W\|^2.$$

where the sum is over the N points in our training set (where $y_n \in \{1, \dots, k\}$, so are we not using the one hot encoding in this expression).

For classification, one can choose the class with the largest predicted probability.

1. (8 points) Write out the derivative of the log-likelihood of the soft-max function as a sum over the N datapoints in our training set and in terms of the vectors x_n , the one hot encoding y_n , and the (vector) prediction \hat{y}_n . (Hint: at this point, you likely will be able to guess the answer. You should still be able to write out a concise derivation). Make sure dimensions give you a gradient of size $d \times k$.
2. (2 points) Now write out the matrix algebra expression for this derivative in terms of $X \in \mathbb{R}^{n \times d}$, $Y \in \mathbb{R}^{n \times 10}$, $\hat{Y} \in \mathbb{R}^{n \times 10}$ (and other relevant quantities).

4 Probability and Maximum Likelihood Estimation [30 points]

4.1 Probability Review (12 points)

1. You are just told by your doctor that you tested positive for a serious disease. The test has 99% accuracy, which means that the probability of testing positive given that you have the disease is

0.99, and also that the probability of testing negative given that you do not have the disease is 0.99. The good news is that this is a rare disease, striking only 1 in 10,000 people.

- (a) Why is it good news that the disease is rare? [1 point]
 - (b) What is the probability that you actually have the disease? Show your work. [4 points]
2. A group of students were classified based on whether they are senior or junior (random variable S) and whether they are taking CSE446 or not (random variable C). The following data was obtained.

| | Junior ($S = 0$) | Senior ($S = 1$) |
|-------------------------------|--------------------|--------------------|
| taking CSE446 ($C = 1$) | 23 | 34 |
| not taking CSE446 ($C = 0$) | 41 | 53 |

Suppose a student was randomly chosen from the group. Calculate the following probabilities. Show your work.

- (a) $\hat{p}(C = 1, S = 1)$ [2 points]
 - (b) $\hat{p}(C = 1|S = 1)$ [2 points]
 - (c) $\hat{p}(C = 0|S = 0)$ [2 points]
3. Why are there “hats” on the p (for probability) symbols above? [1 point]

4.2 Maximum Likelihood Estimation [18 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

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

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

1. Give the log-likelihood function of \mathbf{G} given λ . [6 points]
2. Compute the MLE for λ in the general case. [6 points]
3. Compute the MLE for λ using the observed \mathbf{G} . [6 points]

5 EXTRA CREDIT: Let's get to state of the art on MNIST! [50 points]

If you seek extra credit for this problem, you must also do the extra credit problems in problems one and two, which include the mini-batch SGD method, the quadratic features problem, and the logistic regression problem. If you do this, you actually have the all the code you need to get going! Also, you must answer all of the regular questions as well.

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. In fact, we will shoot to get “state of the art” performance with vanilla least squares.

Remark: At the recent “NIPS” conference (one the premier machine learning conferences), the following talk is quite related to how we will tackle this problem: [youtube video](#), and there are differing views on the state of machine learning. The instructors hope is that, with more hands on experience, you will be better informed about the issues in play. The talk is relevant, since we are basically implementing the method discussed in [this paper](#). The paper itself does not provide the clearest justification, though really is just a “quick and dirty way” to make features. And, in practice, there are often better feature generation methods; this one is remarkably simple.

A few comments are in order. The views of the instructor are that about 1.5% or less is “state of the art”, without “distortions” or “pre-processing” or “convolutional” methods (as discussed on the MNIST website). There are a small number of results below 1.5% (without the aforementioned problem specific tricks), though the instructor is a little skeptical how well these methods transfer to other datasets. Finally, the approach below might seem a little non-sensical. However, there is an important method in that *large* feature representations, appropriately blown up, often perform remarkably well once you have a lot of data.

Remark: In this problem, we will engage in the bad practice where do not have a dev set. To a large extent, looking “a little” at the test set is done in practice (and this shouldn't hurt us too much if we understand how confidence intervals work). However, this has been done for quite sometime on this dataset, which is why the instructor is suspect of the numbers below 1.5%, among those methods that do no use “distortions” or “pre-processing” or “convolutional” methods (we should expect the latter methods to give performance bumps).

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

5.1 Making the features

Now let us try to make “better” features; we are not going to be particularly clever in the way we make these features, though they do provide remarkable improvements. Let x be an image (as a vector in \mathbb{R}^d). Now we will map each x to a \tilde{d} -dimensional feature vector as follows: we will first construct k random vectors, $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ (these will be sampled from a Gaussian distribution). Then our feature vector will be the following vector:

$$\phi(\mathbf{x}) = (\sin \mathbf{v}_1^\top \mathbf{x}, \sin \mathbf{v}_2^\top \mathbf{x}, \dots \sin \mathbf{v}_k^\top \mathbf{x})$$

Note that $\phi(x)$ is a k dimensional vector.

We will use (drumroll please....) $k = 60000$ features! This seems like an unwieldy number. However, it will not actually be so bad since we you never actually explicitly construct and store this dataset. You will construct it “on the fly”. And, the hope, is that

5.2 Let’s implement it.

Details to follow... You will be re-using much of your code here.