CPSC 540 Assignment 1 (due January 11th at midnight)

IMPORTANT!!!!! Before proceeding, please carefully read the homework instructions: `www.cs.ubc.ca/~schmidtm/Courses/540-W18/assignments.pdf`

We will deduct 50% on assignments that do not follow the instructions.

Most of the questions below are related to topics covered in CPSC 340, or other courses listed on the prerequisite form. There are several "notes" available on the webpage which can help with some relevant background.

If you find this assignment to be difficult overall, that is an early warning sign that you may not be prepared to take CPSC 540 at this time. Future assignments will be longer and more difficult than this one.

We use blue to highlight the deliverables that you must answer/do/submit with the assignment.

Basic Information

1. Name:

   Answer: Jonathan Wilder Lavington

2. Student ID:

   Answer: 21090923

3. Graduate students in CPSC/EECE/STAT must submit the prerequisite form as part of a1sol.zip: `https://www.cs.ubc.ca/~schmidtm/Courses/540_prereqs.pdf`

   Answer: Done.

1 Very-Short Answer Questions

Give a short and concise 1-2 sentence answer to the below questions.

1. Why was I unimpressed when a student who thought they did not need to take CPSC 340 said they were able to obtain 97% training accuracy (on their high-dimensional supervised learning problem) as the main result of their MSc thesis?

   Answer: If you are able to train a model on the data that you are applying it to, under sufficient model complexity you can achieve perfect accuracy, even if the model being used is a poor representation of the data.

2. What is the difference between a test set error and the test error?

   Answer: We approximate test error with the error on a test set, therefore the test set error is the monte-carlo approximation of the true test error (i.e. the true expectation).

3. Suppose that a famous person in the machine learning community is advertising their "extremely-deep convolutional fuzzy-genetic Hilbert-long-short recurrent neural network" classifier, which has 500 hyper-parameters. This person claims that if you take 10 different famous (and very-difficult) datasets,

and tune the 500 hyper-parameters based on each dataset's validation set, that you can beat the current best-known validation set error on all 10 datasets. Explain whether or not this amazing claim is likely to be meaningful.

Answer: This claim is irrelevant: tuning hyper-parameters to achieve higher accuracy on a validation set corresponds to just training on your validation set to achieve higher accuracy. Similar to the first question, if you allow yourself to train a model on a given data set, under sufficient complexity it will achieve high accuracy.

4. In a parametric model, what is the effect of the number of training examples n that our model uses on the training error and on the approximation error (the difference between the training error and test error)?

   Answer: As the number of training examples increases, the training error error will monotonically increase (not necessarily strictly monotonically). As the number of training examples increases the approximation error will go down (i.e. the " the training errors approximation of the test error will converge to the true test error).

5. Give a way to set the random tree depth in a random forest model that makes the model parametric, and a choice that makes the model non-parametric.

   Answer: If I set the max depth of the tree to be some number, say 1, then the random forest model will be parametric. If I set the tree depth to be equal to N/3, where N is the number of instances of feature instances within the design matrix, then the model would be non-paramentric.

6. In the regression setting, the popular software XGBoost uses the squared error at the leaves of its regression tree, which is different than the "number of training errors" ($\sum_{i=1}^{n}(\hat{y}^i \neq y^i)$) we used for decision trees in 340. Why does it use the squared error instead of the number of training errors?

   Answer: Under a regression setting its unlikely to get exact accuracy on any given example, but in order to accurately measure the goodness of fit we still require a distance metric; squared error provides a metric for how close our prediction is to the true observation without requiring them to be exact.

7. Describe a situation where it could be better to use gradient descent than Newton's method (known as IRLS in statistics) to fit the parameters of a logistic regression model.

   Answer: Using ISLR (or newtons method) requires d squared storage space, where d is the dimension of your data. Therefore, it would be a bad idea to use it when the data being used to train would induce a matrix that is too large to be stored in memory.

8. How does λ in an L2-regularizer affect the sparsity pattern of the solution (number of w_j set to exactly 0), the training error, and the approximation error?

   Answer: The L2 regularizer will not increase the sparsity of the solution, just the magnitude of some of the coefficients of the solution. As it is a regularization method, it will increase the "simplicity" of the solution, thereby increasing the training error, but hopefully decreasing approximation error.

9. Minimizing the squared error used by in k-means clustering is NP-hard. Given this, does it make sense that the standard k-means algorithm is easily able to find a local optimum?

   Answer: The standard k-means algorithm (also called loyd's algorithm), uses a a sort of greedy approach dependent on the local state of the approximated mean of the k clusters. Because most data have at least some measure of local structure (even as the result of noise), the worst case scenario for k-means is very unlikely to take place.

10. Suppose that a matrix X is non-singular. What is the relationship between the condition number of the matrix, $\kappa(X)$, and the matrix L2-norm of the matrix, $\|X\|_2$.

Answer: The condition number of a matrix is given by: $\kappa(A) = ||A||_2||A^{-1}||_2$, therefore the condition number under the matrix norm induced by the L2 norm gives the product of the largest of the singular value multiplied by one over the smallest singular value. Which as the name implies gives a measure of how close to singular the matrix is.

11. How many regression weights do we have in a multi-class logistic regression problem with k classes?

    Answer: k*(d+1), where d is the number of features and a bias is assumed to be included.

12. Give a supervised learning scenario where you would use the sigmoid likelihood and one where you would use a Poisson likelihood.

    Answer: One could use a sigmoid likelihood when modeling an approximately binary process such as a phase transition during a chemical reaction; where as a Poisson likelihood would better model first arrival times of some process, such as when a bus will arrive.

13. Suppose we need to multiply a huge number of matrices to compute a product like $A_1A_2A_3 \cdots A_k$. The matrices have wildly-different sizes so the order of multiplication will affect the runtime (e.g., $A_1(A_2A_3)$ may be faster to compute than $(A_1A_2)A_3$). Describe (at a high level) an $O(k^3)$-time algorithm that finds the lowest-cost order to multiply the matrices.

    Answer: If we assume the algorithm is recursive, the best option would be to split the set of matrices into two subset, and compute the ordering of the minimal cost multiplication then store the cost and the order each should be multiplied. Doing this for each possible position where the matrices can be split, and then taking the minimum of them will give the desired ordering; however to get O(k^3) operations, we must save the cost of each sub-sequence multiplication that we calculate (memoization). Doing this ensures that the maximum cost at each layer of the split problem will cost at most $O(k^2)$, and becuase the problem will be split k-1 times for a graph of depth $O(k)$ we get the desired $O(k^3)$ run time.

14. You have a supervised learning dataset $\{X, y\}$. You fit a 1-hidden-layer neural network using stochastic gradient descent to minimize the squared error, that makes predictions of the form $\hat{y}^i = v^\top W x^i$ where W and v are the parameters. You find that this gives the same training error as using the linear model ($\hat{y}^i = w^\top x^i$) that minimizes the squared error. You thought the accuracy might be higher for the neural network. Explain why or why not this result is reasonable.

    Answer: This is totally reasonable provided that the data that the model is attempting to predict is simple enough. Additionally because we are using an additional hidden layer, and applying stochastic gradient decent we might take longer to converge to an optimum leading to a perceived convergence at a non optimal value. ....

15. Is it possible that the neural network and training procedure from the previous question results in a higher training error than the linear least squares model? Is it possible that it results in a lower training error?

    Answer: Absolutely, if we are using a stochastic optimization method it is possible to get better convergence as a result of initial conditions or stochasticity, even when using a more complex functional representation. In fact, its likly that under a more complex functional representation of the data, the algorithm will have to run for a longer amount of time to achieve convergence. This is could also happen for a non stochastic optimization problem however, as the one layer neural network (unlike the linear model) is not convex, meaning that we will not be guaranteed a unique optima.

16. What are two reasons that convolutional neural networks over fit less than classic neural networks?

    Answer: The averaging that is done by a convolutional layer helps the network become more robust to small alterations to individual weights, this in turn forces sets of weights that are averaged together with

others to act more harmoniously. Additionally, the number of parameters required for a convolutional neural network of depth d will be less then that of a dense neural network of depth d

2 Calculation Questions

2.1 Minimizing Strictly-Convex Quadratic Functions

Solve for the minimizer w of the below strictly-convex quadratic functions:

1. $f(w) = \frac{1}{2}\|w-u\|_\Sigma$ (projection of u onto the real space under the quadratic norm defined by Σ).

   Answer: The first thing we can do is re-write this in matrix-vector notation:

   $$\frac{1}{2}\|w-u\|_\Sigma = \frac{1}{2}(w-v)^\top\Sigma(w-v) = \frac{1}{2}(w^\top\Sigma w) - (u^\top\Sigma w) + \frac{1}{2}(u^\top\Sigma u)$$

   Next we can take the gradient of the expression:

   $$\begin{aligned}\nabla f(w) &= \nabla(\frac{1}{2}(w^\top\Sigma w) + (w^\top\Sigma u) - \frac{1}{2}(u^\top\Sigma u)) \\ &= \Sigma w + \Sigma u\end{aligned}$$

   Finally we set this expression equal to zero,

   $$\begin{aligned}0 &= \Sigma w - \Sigma u \\ 0 &= \Sigma(w-u) \\ \Sigma^{-1}0 &= \Sigma^{-1}\Sigma(w-u) \\ 0 &= (w-u) \\ w &= u\end{aligned}$$

   Where 0 is the zero vector, and we know that Σ has an inverse because its symmetric positive definite.

2. $f(w) = \frac{1}{2\sigma^2}\|Xw-y\|^2 + w^\top\Lambda w$ (ridge regression with known variance and weighted L2-regularization).

   Answer: Again we are going to split this norm expression into the corrosponding matrix-vector notation.

   $$\begin{aligned}f(w) &= \frac{1}{2\sigma^2}\|Xw-y\|^2 + w^\top\Lambda w \\ &= \frac{1}{2\sigma^2}(Xw-y)^\top(Xw-y) + w^\top\Lambda w \\ &= \frac{1}{2\sigma^2}(w^\top X^\top Xw + y^\top y) - w^\top X^\top y + w^\top\Lambda w\end{aligned}$$

   Next we are going to take the gradient of this expression,

   $$\begin{aligned}\nabla f(w) &= \nabla(\frac{1}{2\sigma^2}(w^\top X^\top Xw + y^\top y) - \frac{1}{\sigma^2}w^\top X^\top y + w^\top\Lambda w) \\ &= \frac{1}{\sigma^2}(X^\top Xw) - \frac{1}{\sigma^2}X^\top y + 2 * \Lambda w \\ &= \frac{1}{\sigma^2}(X^\top X + 2\sigma^2 * \Lambda)w - \frac{1}{\sigma^2}X^\top y\end{aligned}$$

Lastly we set the the gradient equal to zero and solve for w:

$$\begin{aligned}
\nabla f(w) &= 0 \\
0 &= \frac{1}{\sigma^2}(X^\top X + 2\sigma^2 * \Lambda)w - \frac{1}{\sigma^2}Xy \\
\frac{1}{\sigma^2}X^\top y &= \frac{1}{2\sigma^2}(X^\top X + 2\sigma^2 * \Lambda)w \\
X^\top y &= \frac{1}{2}(X^\top X + 2\sigma^2 * \Lambda)w \\
w &= (X^\top X + 2\sigma^2 * \Lambda)^{-1}X^\top y
\end{aligned}$$

3. $f(w) = \frac{1}{2}\sum_{i=1}^n v_i(w^\top x^i - y^i)^2 + \frac{1}{2}(w-u)^\top\Lambda(w-u)$ (weighted least squares shrunk towards u).

   Answer: For this problem, it is already, conveniently, written out as a simplified expression. So we can just start by taking the gradient:

$$\begin{aligned}
f(w) &= \frac{1}{2}\sum_{i=1}^n v_i(w^\top x^i - y^i)^2 + \frac{1}{2}(w-u)^\top\Lambda(w-u) \\
f(w) &= \frac{1}{2}\sum_{i=1}^n v_i(w^\top x^i - y^i)^2 + \frac{1}{2}(w^\top\Lambda w - 2u^\top\Lambda w + u^\top u) \\
\nabla f(w) &= \nabla\frac{1}{2}\sum_{i=1}^n v_i(w^\top x^i - y^i)^2 + \nabla\frac{1}{2}(w^\top\Lambda w - 2u^\top\Lambda w + u^\top u) \\
\nabla f(w) &= \sum_{i=1}^n (x^i)^\top v_i(w^\top x^i - y^i) + \Lambda w - u^\top\Lambda
\end{aligned}$$

   Next we set this quantity equal to zero, and solve for w,

$$\begin{aligned}
\nabla f(w) &= 0 \\
0 &= \sum_{i=1}^n (x^i)^\top v_i(w^\top x^i - y^i) + \Lambda w - u^\top\Lambda \\
0 &= \sum_{i=1}^n (x^i)^\top v_i(w^\top x^i) - \sum_{i=1}^n (x^i)^\top v_i(y^i) + \Lambda w - u^\top\Lambda \\
\sum_{i=1}^n (x^i)^\top v_i(y^i) + u^\top\Lambda &= \sum_{i=1}^n (w^\top x^i)(x^i)^\top v_i + w^\top\Lambda \\
\sum_{i=1}^n (x^i)^\top v_i(y^i) + u^\top\Lambda &= w^\top(\sum_{i=1}^n x^i(x^i)^\top v_i + \Lambda) \\
(\sum_{i=1}^n (x^i)^\top v_i(y^i) + u^\top\Lambda)(\sum_{i=1}^n x^i(x^i)^\top v_i + \Lambda)^{-1} &= w^\top \\
w &= (X^\top VX + \Lambda)^{-1}(X^\top Vy + \Lambda u)
\end{aligned}$$

Above we use our usual supervised learning notation. In addition, we assume that u is $d \times 1$ and v is $n \times 1$, while Σ and Λ are symmetric positive-definite $d \times d$ matrices. You can use V as a diagonal matrix with v along the diagonal (with the v_i non-negative). Hint: positive-definite matrices are invertible.

2.2 Norm Inequalities

Show that the following inequalities hold for vectors $w \in \mathbb{R}^d$, $u \in \mathbb{R}^d$, and $X \in \mathbb{R}^{n \times d}$:

1. $\|w\|_\infty \leq \|w\|_2 \leq \|w\|_1$ (relationship between decreasing p-norms)

   Answer: Starting with the first inequality: $\|w\|_2 \leq \|w\|_1$,

$$
\begin{aligned}
0 &\leq \sum_{i=1}^{n} \sum_{j \neq i} |w_i||w_j| \\
\sum_{i=1}^{n} |w_i|^2 &\leq \sum_{j=1}^{n} |w_j|^2 + \sum_{i=1}^{n} \sum_{j \neq i} |w_i||w_j| \\
\sum_{i=1}^{n} |w_i|^2 &\leq \sum_{i=1}^{n} \sum_{j=1}^{n} |w_i||w_j| \\
\sum_{i=1}^{n} |w_i|^2 &\leq (\sum_{i=1}^{n} |w_i|)(\sum_{j=1}^{n} |w_j|) \\
\|w\|_2^2 &\leq \|w\|_1^2 \\
\|w\|_2 &\leq \|w\|_1
\end{aligned}
$$

   Where the last step follows from the fact that norms are non-negative. Next we can prove the second inequality: $\|w\|_\infty \leq \|w\|_2$

$$
\begin{aligned}
0 &\leq \sum_{i \neq \max} |w|^2 \\
\max|w|^2 &\leq \max|w|^2 + \sum_{i \neq \max} |w_i|^2 \\
\max|w|^2 &\leq \sum_{i=1}^{n} |w_i|^2 \\
\|w\|_\infty^2 &\leq \|w\|_2^2 \\
\|w\|_\infty &\leq \|w\|_2
\end{aligned}
$$

   Where the last step again follows from the no-negative property of norms. Lastly, because $\|w\|_\infty \leq \|w\|_2$ and $\|w\|_2 \leq \|w\|_1$ necessarily: $\|w\|_\infty \leq \|w\|_2 \leq \|w\|_1$.

2. $\frac{1}{2}\|w + u\|_2^2 \leq \|w\|_2^2 + \|u\|_2^2$ ("not the triangle inequality" inequality)

Answer:

$$
\begin{aligned}
0 &\leq \|w-u\|_2^2 \\
0 &\leq (w-u)^\top (w-u) \\
0 &\leq w^\top w - 2w^\top u + u^\top u \\
2w^\top u &\leq w^\top w + u^\top u \\
w^\top u &\leq \frac{1}{2}(w^\top w + u^\top u) \\
\frac{1}{2}(w^\top u + u^\top w) &\leq \frac{1}{2}(w^\top w + u^\top u) \\
\frac{1}{2}(w^\top w + w^\top u + u^\top w + u^\top u) &\leq w^\top w + u^\top u \\
\frac{1}{2}(w+u)^\top (w+u) &\leq w^\top w + u^\top u \\
\frac{1}{2}\|w+u\|_2^2 &\leq \|u\|_2^2 + \|w\|_2^2
\end{aligned}
$$

3. $\|X\|_2 \leq \|X\|_F$ (matrix norm induced by L2-norm is smaller than Frobenius norm)

   Answer: This proof relies on various identities of the associated norms in question.

   - $\|X\|_2 = \max(\sigma_i(x))$ where $\max(\sigma_i(X))$ indicates the largest singular value of the matrix X.
   - $\|X\|_F = \sqrt{Tr(X^\top X)}$
   - $Trace(X^\top X) = \sum \lambda_{i,X^\top X} = \sum \sigma_{i,X^\top X}$, where the last equality follows from the fact that $X^\top X$ is a hermitian matrix.
   - $\sum_{i=1}^{n}\sum_{j=1}^{d} x_{ij}^2 = \sum_{c=1}^{\min\{n,d\}} \sigma_c(X)^2$ (where $\sigma_c(X)$ is singular value c of X)
   - Lastly that the singular values of $X^\top X$ are equal to the square of the singular values of X.

   With these identities in mind, we can proceed with the proof:

$$
\begin{aligned}
0 &\leq \sum_{i \neq \max} \sigma_{i,X}^2 \\
\sigma_{\max,X}^2 &\leq \sum_{i=1}^{n} \sigma_{i,X}^2 \\
\sigma_{\max,X}^2 &\leq \sum_{i=1}^{n} \sigma_{i,X^\top X} \\
\sigma_{\max,X}^2 &\leq \mathrm{Tr}(X^\top X) \\
\sigma_{\max,X} &\leq \sqrt{\mathrm{Tr}(X^\top X)} \\
\|X\|_2 &\leq \|X\|_F
\end{aligned}
$$

You should use the definitions of the norms, but should not use the known equivalences between these norms (since these are the things you are trying to prove). Hint: for many of these it's easier if you

work with squared values (and you may need to "complete the square"). Beyond non-negativity of norms, it may also help to use the Cauchy-Schwartz inequality, to use that $\|x\|_1 = x^\top \text{sign}(x)$, and to use that $\sum_{i=1}^n \sum_{j=1}^d x_{ij}^2 = \sum_{c=1}^{\min\{n,d\}} \sigma_c(X)^2$ (where $\sigma_c(X)$ is singular value c of X).

2.3 MAP Estimation

In 340, we showed that under the assumptions of a Gaussian likelihood and Gaussian prior,

$$y^i \sim \mathcal{N}(w^\top x^i, 1), \quad w_j \sim \mathcal{N}\left(0, \frac{1}{\lambda}\right),$$

that the MAP estimate is equivalent to solving the L2-regularized least squares problem

$$f(w) = \frac{1}{2}\sum_{i=1}^n (w^\top x^i - y^i)^2 + \frac{\lambda}{2}\sum_{j=1}^d w_j^2,$$

in the "loss plus regularizer" framework. For each of the alternate assumptions below, write it in the "loss plus regularizer" framework (simplifying as much as possible):

1. Laplace likelihood (with a scale of 1) for each training example and Gaussian prior with separate variance σ_j^2 for each variable

$$y^i \sim \mathcal{L}(w^\top x^i, 1), \quad w_j \sim \mathcal{N}\left(0, \sigma_j^2\right).$$

   Answer:

$$f(w) = \sum_{i=1}^n |y^i - w^\top x^i| + \frac{1}{2}\sum_{i=1}^d \frac{w_i^2}{\sigma_i^2}$$

2. Robust student-t likelihood and Gaussian prior centered at u.

$$p(y^i|x^i, w) = \frac{1}{\sqrt{\nu}B\left(\frac{1}{2}, \frac{\nu}{2}\right)}\left(1 + \frac{(w^\top x^i - y^i)^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \quad w_j \sim \mathcal{N}\left(u_j, \frac{1}{\lambda}\right),$$

   where u is $d \times 1$, B is the "Beta" function, and the parameter ν is called the "degrees of freedom".[1]

   Answer:

$$f(w) = \frac{\nu+1}{2}\sum_{i=1}^n \log\left[1 + \frac{(w^\top x^i - y^i)^2}{\nu}\right] + \frac{\lambda}{2}\sum_{i=1}^d (w_i - \mu_i)^2$$

3. We use a Poisson-distributed likelihood (for the case where y_i represents counts), and we use a uniform prior for some constant κ,

$$p(y^i|w^\top x^i) = \frac{\exp(y^i w^\top x^i)\exp(-\exp(w^\top x^i))}{y^i!}, \quad p(w_j) \propto \kappa.$$

   (This prior is "improper" since $w \in \mathbb{R}^d$ but it doesn't integrate to 1 over this domain, but nevertheless the posterior will be a proper distribution.)

[1] This likelihood is more robust than the Laplace likelihood, but leads to a non-convex objective.

Answer:

$$f(w) = \sum_{i=1}^{n} \left[\exp w^\top x^i - y^i w^\top x^i\right]$$

For this question, you do not need to convert to matrix notation.

2.4 Gradients and Hessian in Matrix Notation

Express the gradient $\nabla f(w)$ and Hessian $\nabla^2 f(w)$ of the following functions in matrix notation, simplifying as much as possible:

1. Regularized and tilted Least Squares

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2 + w^\top u.$$

where u is $d \times 1$.

Answer:

$$\begin{aligned} f(w) &= (Xw - y)^\top (Xw - y) + \frac{\lambda}{2} w^\top w + w^\top u \\ \nabla f(w) &= (X^\top X + \lambda I)w + u + \lambda w \\ \nabla^2 f(w) &= X^\top X + \lambda I \end{aligned}$$

2. L2-regularized weighted least squares with non-Euclidean quadratic regularizaiton,

$$f(w) = \frac{1}{2}\sum_{i=1}^{n} v_i (w^\top x^i - y^i)^2 + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d} w_i w_j \lambda_{ij}$$

Answer:

$$\begin{aligned} f(w) &= \frac{1}{2}(Xw - y)^\top V (Xw - y) + \frac{1}{2} w^\top \Sigma w \\ \nabla f(w) &= X^\top V X w - X^\top V y + \Sigma w \\ \nabla^2 f(w) &= X^\top V X + \Sigma \end{aligned}$$

where you can use V as a matrix with the v_i along the diagonal and Λ as a positive-definite $d \times d$ (symmetric) matrix with λ_{ij} in position (i, j).

3. Squared hinge loss,

$$f(w) = \frac{1}{2}\sum_{i=1}^{n} \left(\max\{0, 1 - y^i w^\top x^i\}\right)^2.$$

Answer: If we define $diag(y)$ as a diagnol matrix with each of the values along the diagnol indicating a row in the vector y, and I_{max} as an indicator matrix, which will zero out all columns of x and rows

of y where the $0 \leq 1 - y^i w^\top x^i$, then we can write the gradient, and hessian concisely in the following way:

$$f(w) = \frac{1}{2}(1 - \text{diag}(y)Xw)^\top I_{max}(1 - \text{diag}(y)Xw)$$
$$\nabla f(w) = X^\top diag(y) I_{max} diag(y) Xw - X^\top diag(y) I_{max} \mathbb{1}$$
$$\nabla^2 f(w) = X^\top diag(y) I_{max} diag(y) X$$

Also note that above, $\mathbb{1}$ indicates a vector of all ones.

Hint: You can use the results from the linear and quadratic gradients and Hessians notes to simplify the derivations. You can use 0 to represent the zero vector or a matrix of zeroes and I to denote the identity matrix. It will help to convert the second question to matrix notation first. For the last question you'll need to define new vectors to express the gradient and Hessian in matrix notation and you can use $\circ$ as element-wise multiplication of vectors. As a sanity check, make sure that your results have the right dimension.

3 Coding Questions

If you have not previously used Julia, there is a list of useful Julia commands (and syntax) among the list of notes on the course webpage.

3.1 Regularization and Hyper-Parameter Tuning

Download *a1.zip* from the course webpage, and start Julia (latest version) in a directory containing the extracted files. If you run the script *example_nonLinear*, it will:

1. Load a one-dimensional regression dataset.
2. Fit a least-squares linear regression model.
3. Report the test set error.
4. Draw a figure showing the training/testing data and what the model looks like.

This script uses the *JLD* package to load the data and the *PyPlot* package to make the plot. If you have not previously used these packages, they can be installed using:[2]

```
using Pkg
Pkg.add("JLD")
Pkg.add("PyPlot")
```

Unfortunately, this is not a great model of the data, and the figure shows that a linear model is probably not suitable.

1. Write a function called *leastSquaresRBFL2* that implements *least squares using Gaussian radial basis functions (RBFs) and L2-regularization.*
   You should start from the *leastSquares* function and use the same conventions: n refers to the number of training examples, d refers to the number of features, X refers to the data matrix, y refers to the targets, Z refers to the data matrix after the change of basis, and so on. Note that you'll have to add two additional input arguments (λ for the regularization parameter and σ for the Gaussian RBF variance) compared to the *leastSquares* function. To make your code easier to understand/debug, you

[2]Last term, several people (eventually including myself) had a runtime problem on some system. This seems to be fixed using the answer of K. Gkinis at this url: `https://stackoverflow.com/questions/46399480/julia-runtime-error-when-using-pyplot`

may want to define a new function *rbfBasis* which computes the Gaussian RBFs for a given training set, testing set, and σ value. Hand in your function and the plot generated with $\lambda = 1$ and $\sigma = 1$.

Answer: See Figure 1 and Figure 2.

```
# Gaussiam Radial Basis Function
using LinearAlgebra
function RBF_kernal(X, X_hat, sigma)
        return exp.(-distancesSquared(X_hat, X)./(2*sigma*sigma))
end  > RBF_kernal

function rbfBasis(Xtrain, y_train, Xtest, sigma, lambda)
        # dimensions
        (n_train,d_train) = size(Xtrain)
        (n_test, d_test) = size(Xtest)
        # basis
        Z_train = RBF_kernal(Xtrain, Xtrain, sigma)
        Z_test = RBF_kernal(Xtrain, Xtest, sigma)
        # solve for weights v
        v = (Z_train'*Z_train+lambda*Matrix{Float64}(I, n_train, n_train)) \ Z_train'*y_train
        # return pred
        return Z_test*v
end  > rbfBasis
```

Figure 1: Julia program that takes a training data set and a test data set, trains an RBF kernal then uses the test data set to predict.

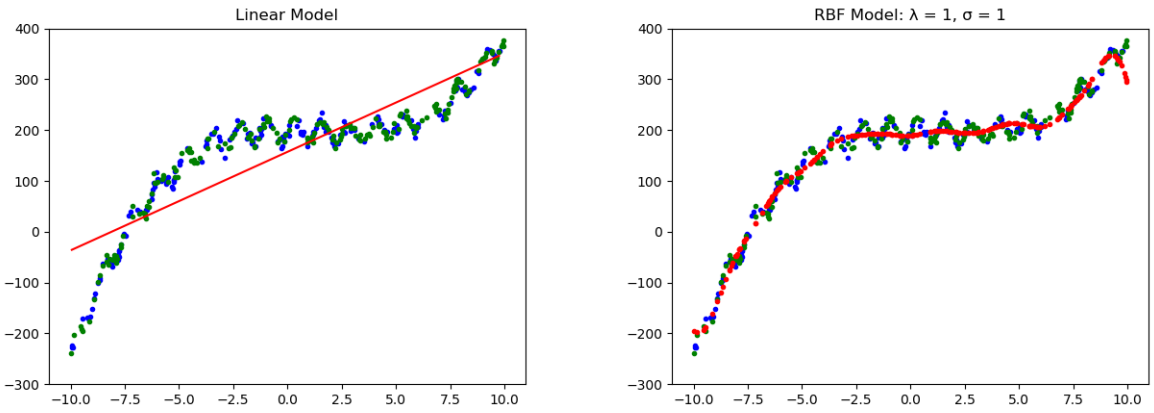

Figure 2: Plots of the original linear model provided, as well as the RBF model with both hyper parameters set to 1.

2. When dealing with larger datasets, an important issue is the dependence of the computational cost on the number of training examples n and the number of features d. What is the cost in big-O notation of training the model on n training examples with d features under (a) the linear basis, and (b) Gaussian RBFs (for a fixed σ)? What is the cost of classifying t new examples under these two bases? Assume that multiplication by an n by d matrix costs $O(nd)$ and that solving a d by d linear system costs $O(d^3)$.

   Answer:

   - Training Cost - Linear Model = $O(d^3 + d^2 n)$

- Training Cost - Gaussian RBF = $O(dn^2 + n^3)$
- Classifying Cost - Linear Model = $O(td)$
- Classifying Cost - Gaussian RBF = $O(ndt)$

3. Modify the script to split the training data into a "train" and "validation" set (you can use half the examples for training and half for validation), and use these to select λ and σ. Hand in your modified script and the plot you obtain with the best values of λ and σ.
Solution: See Figure 3, Figure 4, and Figure 5.

```
# split data function
using Random
function split_data(X, y, split)
        total_data = size(y)[1]
        indices = randperm(total_data)
        training_indices = indices[1:Int(floor(split*total_data))]
        validation_indices = indices[Int(floor(split*total_data))+1:end]
        X_train, y_train = X[training_indices,:], y[training_indices]
        X_val, y_val = X[validation_indices,:], y[validation_indices]
        return X_train, y_train, X_val, y_val
end
```

Figure 3: Code to split up the training data into training and validation

```
function minimize_hyper_parameters(X, y, iterations)
        # split up the data
        X_train, y_train, X_val, y_val = split_data(X, y, 0.5)
        t = size(X_val,1)

        # iterate through combinations of lambda and sigma
        best_error = Inf
        sigmas, lambdas  = range(0.0001, stop = 50, length = iterations) |> collect, range(0.0001, stop = 50, length = iterations) |> collect
        sigma_best, lambda_best = 0.0001, 0.0001

        # iterate
        for iter_1 = 1:iterations
                for iter_2 = 1:iterations
                        sigma_current = sigmas[iter_1]
                        lambda_current = lambdas[iter_2]
                        yhat = rbfBasis(X_train, y_train, X_val, sigma_current, lambda_current)
                        current_error = sum((yhat - y_val).^2)/t
                        if current_error < best_error
                                sigma_best = sigma_current
                                lambda_best = lambda_current
                                best_error = current_error
                        end
                end
        end
        return best_error, lambda_best, sigma_best
end

# get the best hyper parameters
best_error, lambda_best, sigma_best = minimize_hyper_parameters(X, y, 100)
```

Figure 4: Code to determine optimal hyper-parameters λ and σ using a grid search approach.

4. There are reasons why this dataset is particularly well-suited to Gaussian RBFs are that (i) the period of the oscillations stays constant and (ii) we have evenly sampled the training data across its domain. If either of these assumptions are violated, the performance with our Gaussian RBFs might be much worse. Consider a scenario where either (i) or (ii) is violated, and describe a way that you could address this problem.

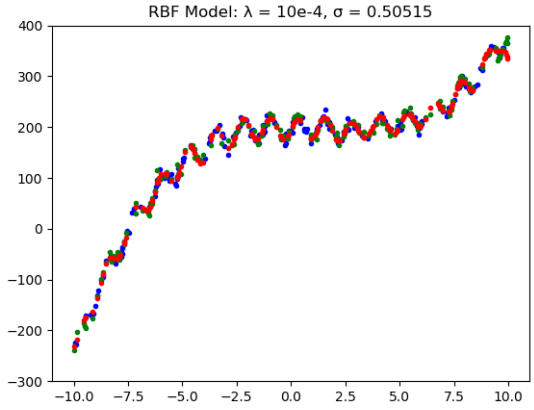

Figure 5: Model predictions using optimal hyper parameters, overlayed with the true data.

Answer:

- (i) If the period of the osculations is does not remain constant one could apply an invertable transformation to the data prior to applying a regression model.
- (ii) If the training data is not evenly sampled across the domain, one could apply weighted regression in order up-weight values in areas that have a low density of points, similar to sparse data problems (often come up in predicting driver failure).

Note: the *distancesSquared* function in *misc.jl* is a vectorized way to quickly compute the squared Euclidean distance between all pairs of rows in two matrices.

3.2 Multi-Class Logistic Regression

The script *example_multiClass.jl* loads a multi-class classification dataset and fits a "one-vs-all" logistic regression classifier, then reports the validation error and shows a plot of the data/classifier. The performance on the validation set is ok, but could be much better. For example, this classifier never even predicts some of the classes.

Using a one-vs-all classifier hurts performance because the classifiers are fit independently, so there is no attempt to calibrate the columns of the matrix W. An alternative to this independent model is to use the softmax probability,

$$p(y^i|W, x^i) = \frac{\exp(w_{y^i}^\top x^i)}{\sum_{c=1}^k \exp(w_c^\top x^i)}.$$

Here c is a possible label and w_c is column c of W. Similarly, y^i is the training label, w_{y^i} is column y^i of W. The loss function corresponding to the negative logarithm of the softmax probability is given by

$$f(W) = \sum_{i=1}^n \left[-w_{y^i}^\top x^i + \log\left(\sum_{c'=1}^k \exp(w_{c'}^\top x^i) \right) \right].$$

Make a new function, *softmaxClassifier*, which fits W using the softmax loss from the previous section instead of fitting k independent classifiers. Hand in the code and report the validation error.

Hint: you can use the *derivativeCheck* option when calling *findMin* to help you debug the gradient of the softmax loss. Also, note that the *findMin* function treats the parameters as a vector (you may want to use *reshape* when writing the softmax objective).

Answer: See Figures 6, 7, 8, and 9 for the desired code. The result of training using the generic parameters for findMin gave a training error of .002, and a validation error of 0.012. It should be noted that with increased training iterations and a decreased tolerance the error can be decreased even further (roughly 0.0 training error and 0.005 test error).

```
# softmax objective
function softmax_objective(W_vec, X, y, f, n, k)
    W = reshape(W_vec, (f, k))
    objective = 0
    for i = 1:n
        objective += -W[:,y[i]]'*X[i,:]
        log_sum = 0
        for j = 1:k
            log_sum += exp.(W[:,j]'*X[i,:])
        end
        objective += log.(log_sum)
    end
    return objective
end
```

Figure 6: Code for the softmax objective function.

```
# softmax gradient
function softmax_gradient(W_vec, X, y, f, n, k)
    W = reshape(W_vec, (f, k))
    dims = size(W)
    objective = zeros(dims)
    for index_1 = 1:dims[1]
        for index_2 = 1:dims[2]
            objective_current = 0
            for i = 1:n
                # first chunk
                indicator = 0
                if y[i] == index_2
                    indicator = 1
                end
                objective_current -= X[i,index_1]*indicator
                # sum of exps
                sum_of_exp = 0
                for c = 1:k
                    sum_of_exp += exp(W[:,c]'*X[i,:])
                end
                objective_current += exp(W[:,index_2]'*X[i,:])*X[i,index_1] / sum_of_exp
            end
            # add to gradient
            objective[index_1, index_2] = objective_current
        end
    end
    return vec(objective)
end
```

Figure 7: Code for the softmax gradient function.

3.3 Robust and Brittle Regression

The script *example_outliers.jl* loads a one-dimensional regression dataset that has a non-trivial number of "outlier" data points. These points do not fit the general trend of the rest of the data, and pull the least

```
function softmaxClassifier(X, y, f, n, k)
    # set objective function
    funObj(W_vec) = (softmax_objective(W_vec, X, y, f, n, k), softmax_gradient(W_vec, X, y, f, n, k))
    # initialize weights
    w_init =0.005*rand(Int(f*k))
    # train weights
    w = findMin(funObj, w_init)
    # return function
    return w
end  > softmaxClassifier
```

Figure 8: Code for the softmax classifier function, which trained the softmax classifier on some data set X and then returned the trained weights w.

```
# predict function
function softmax_prediction(W_vec, X, f, n, k)
    # initialize predictions
    pred = zeros(n)
    W = reshape(W_vec, (f, k))
    # iteratate trough data set
    for i = 1:n
        # calculate sum
        current_sum = 0
        for j = 1:k
            current_sum += exp.(W[:,j]'*X[i,:])
        end
        p = zeros(k)
        # iterate through classes
        for j = 1:k
            p[j] = exp.(W[:,j]'*X[i,:]) / current_sum
        end
        # find index with highest probability
        pred[i] = Int(findmax(p)[2])
    end
    return pred
end  > softmax_prediction
```

Figure 9: Code for the softmax prediction function.

squares model away from the main cluster of points. One way to improve the performance in this setting is simply to remove or downweight the outliers. However, in high-dimensions it may be difficult to determine whether points are indeed outliers (or the errors might simply be heavy-tailed). In such cases, it is preferable to replace the squared error with an error that is more robust to outliers.

1. Write a new function, *leastAbsolutes(X,y)*, that adds a bias variable and fits a linear regression model by minimizing the absolute error instead of the square error,

$$f(w) = \|Xw - y\|_1.$$

   You should turn this into a *linear program* as shown in class, and you can solve this linear program using the *linprog* function the *MathProgBase* package. Hand in the new function and the updated plot.

   Answer: To see the Code used to solve the robust regression problem using linear programming, see Figure 10. To check out the plot of Robust Regression compared to normal least squares, check out Figures 11 and 12.

2. The previous question assumes that the "outliers" are points that we don't want to model. But what if we want good performance in the worst case across *all* examples (including the outliers)? In this setting we may want to consider a "brittle" regression method that chases outliers in order to improve

```
function leastAbsolutes(X,y)
    # add bias to X
    bias = ones(size(y))
    phi = [bias X]
    # initialize matrices
    M = size(phi)[2]
    b = zeros(2*size(y)[1])
    A = zeros(2*size(y)[1], M)
    # generate constraint matrices
    for i = 1:2:2*size(y)[1]
        A[i,1:M] = phi[Int((i+1)/2),:]
        A[i+1,1:M] = -phi[Int((i+1)/2),:]
        b[i] = y[Int((i+1)/2)]
        b[i+1] = -y[Int((i+1)/2)]
        zero_vec = zeros(2*size(y)[1])
        zero_vec[i] = -1
        zero_vec[i+1] = -1
        A = [A zero_vec]
    end
    # generate minimization matrix
    C = zeros(size(y)[1]+M)
    for i = M:size(y)[1]+M
        C[i] = 1
    end
    # apply linsolve
    w = linprog(C, A, -Inf, b, -Inf, Inf, GLPKSolverLP())
    return w.sol[1:M]
end
```

Figure 10: Code for to generate linear program required to solve robust regression.

the worst-case error. For example, consider minimizing the absolute error in the worst-case,

$$f(w) = \|Xw - y\|_\infty.$$

This objective function is non-smooth because of the absolute value function as well as the max function. Show how to formulate this non-smooth optimization problem as a linear program.

Answer: Starting with the original optimization problem given by,

$$\underset{w}{\text{minimize}} \quad \|Xw - y\|_\infty$$

We can introduce a decision variable z and rewrite this using an equivalent optimization problem,

$$\begin{aligned} \text{minimize} \quad & z \\ \text{subject to} \quad & \|Xw - y\|_\infty \leq z. \end{aligned}$$

If we then rewrite this over the columns of A, defined a_i

$$\begin{aligned} \text{minimize} \quad & z \\ \text{subject to} \quad & \max_{1 \leq m} |a_i \top x - b| \leq z. \end{aligned}$$

Which is equivalent to, just requiring that all columns satisfy this requirement:

$$\begin{aligned} \text{minimize} \quad & z \\ \text{subject to} \quad & |a_1 \top x - b| \leq z. \\ & \ldots \\ & |a_n \top x - b| \leq z. \end{aligned}$$

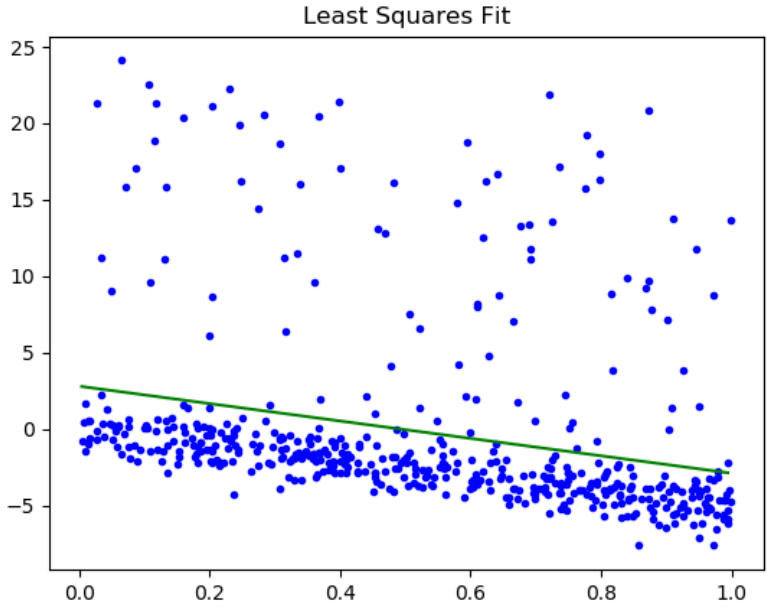

Figure 11: Vanilla Regression Applied to Outliers Problem

Using the definition of absolute values, we can then rewrite the constraints such that,

$$\begin{aligned} \text{minimize} \quad & z \\ \text{subject to} \quad & -z \leq a_1 \top x - b \leq z \\ & \ldots \\ & -z \leq a_n \top x - b \leq z \end{aligned}$$

Finally using matrix vector notation this give us something that we can put into linprog,

$$\begin{aligned} \text{minimize} \quad & z \\ \text{subject to} \quad & -z\mathbb{1} \leq Ax - b \leq z\mathbb{1} \end{aligned}$$

Where $\mathbb{1}$ indicates a column vector composed of ones.

3. Write and hand in a function, *leastMax*, that fits this model using *linprog* (after adding a bias variable). Hand in the new function and the updated plot.

   Answer: To see the Code used to solve the brittle regression problem using linear programming, see Figure 13. To check out the plot of brittle Regression, see out Figure 14 .

To use the *linprog* function, you can use:

```
using MathProgBase, GLPKMathProgInterface
solution = linprog(c,A,d,b,lb,ub,GLPKSolverLP())
x = solution.sol
```

This requires installing the appropriate packages, and finds a vector x minimizing the function $c^\top x$ subject to $d \leq Ax \leq b$ and $\text{lb} \leq x \leq \text{ub}$. You can set values of c to 0 for variables that don't affect the cost function,

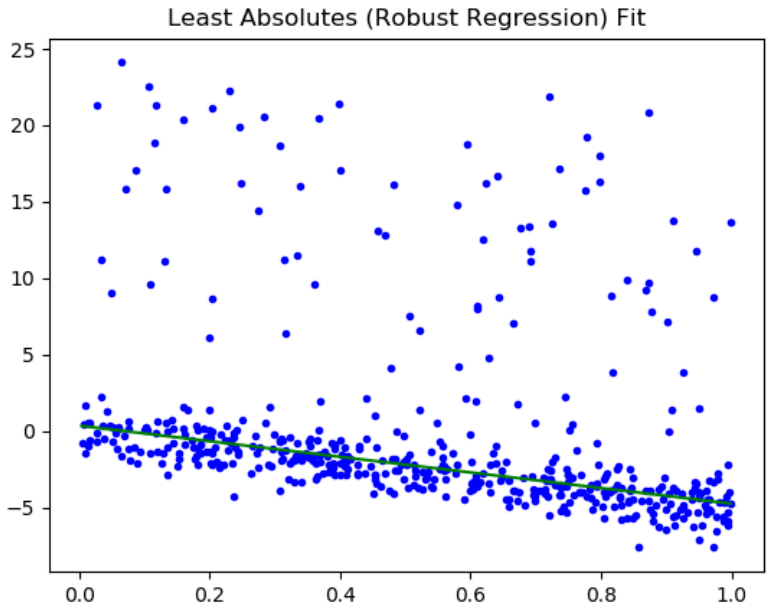

Figure 12: Robust Regression Applied to Outliers Problem

and you can set values in b/d/lb/ub (or the other variables) to appropriate infinite values if there are no lower/upper bounds. The vectors $c/d/b/lb/ub$ should all be lists.

```
function leastMax(X,y)
    # add bias to X
    bias = ones(size(y))
    phi = [bias X]
    # initialize matrices
    M = size(phi)[2]
    b = zeros(2*size(y)[1])
    A = zeros(2*size(y)[1], M)
    # generate constraint matrices
    for i = 1:2:2*size(y)[1]
        A[i,1:M] = phi[Int((i+1)/2),:]
        A[i+1,1:M] = -phi[Int((i+1)/2),:]
        b[i] = y[Int((i+1)/2)]
        b[i+1] = -y[Int((i+1)/2)]
    end
    zero_vec = -1*ones(2*size(y)[1])
    A = [A zero_vec]
    # generate minimization matrix
    C = zeros(M+1)
    C[M+1] = 1
    # apply linsolve
    w = linprog(C, A, -Inf, b, -Inf, Inf, GLPKSolverLP())
    return w.sol[1:M]
end  > leastMax
```

Figure 13: Code for the linear program associated with Brittle Regression

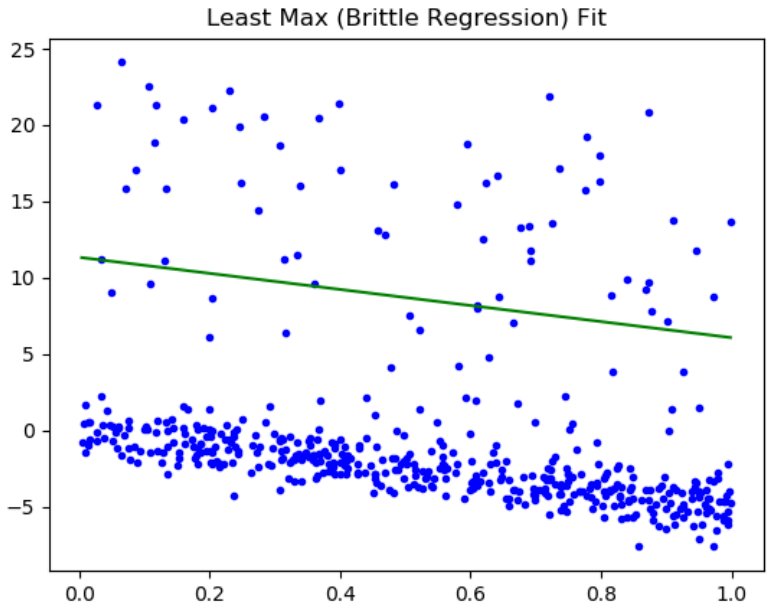

Figure 14: Brittle Regression Applied to Outliers Problem