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: Kyle Leeners
2. Student ID: 30591119
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`

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:
   Because CPSC 340 lays down a lot of the foundation required to be successful in 540. Furthermore, having a high training accuracy does not say anything about the test accuracy so we have no way to tell if the model is accurate or not.

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

   Answer:
   The test set error is simply the error of a given set of test data whereas the test error is the expected error within our model.

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:
As we increase k E_{valid} goes down however E_{approx} goes up. This famous person is likely overfitting their model.

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:
   Increasing n will result in a higher training error but will decrease the approximation 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:
   Parametric: depth = 2. Non-parametric: depth = n / 2

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:
   The 340 decision trees were used for classification problems, regression trees are needed for regression problems and therefore must use some sort of continuous loss function to make predictions

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:
   The hessian is often expensive to compute (second derivative) whereas gradient descent only requires knowledge of the gradient.

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:
   As λ increases we favor the regularizer over the loss function. Since we are using a squared regularizer, we will place more weight on large coefficients over smaller ones thus promoting sparsity.

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:
   K-means is guaranteed to converge to a local optimum but there is no guarantee that this is also the global optimum. That is why in practice, random restarts of often performed and the best optimum out of a set is picked.

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:
    $\kappa(X) = \|X\|\|X^{-1}\|$

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

    Answer:
    With d features, we will have $d \times k$ matrix of coefficients

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

    Answer:
    We would want to use the sigmoid likelihood when the output variable is binary (ex. yes or no). The Poisson likelihood is better suited when the output variable is a count (ex. number of cars passing through an intersection)

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:
    This is a classic DP program that can be solved using the top-down approach. The gist of what we need to do is recursively split the matrix sequence into two subsequences, use the recursion result to get the cost of creating the subsequences (and the dimensions of the subsequences) and then return the cost of multiplying the subsequences together (as well as the dimensions of the multiplied subsequences). We do this for all possible splits at each level of the recursion and then starting from the result, traverse the cost matrix to find the best way to spit. Since we will be computing the cost of the same subsequences repeatedly we use memoization and store these results in a lookup table.

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 just a round-about way to perform linear regression since $Wx^i = z^i$. To properly utilize a neural net we need to add a non-linear transformation: $v^T h(Wx^i)$

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:
    It could in theory get a worse training error. Since we are using SGD there is a chance we overshoot the minimum or terminate too early.

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

    Answer:

    (a) Using a filter often increases parameter sparsity which reduces overfitting (and increases training speed)

    (b) We often pool the result after applying a filter, reducing the dimension of our parameter matrix and thus number of parameters

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:
   $f(w) = \frac{1}{2}\|w\|^2 - w^T u + \|u\|^2$ so $\nabla f(w) = w - u \Rightarrow w = u$

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:
   $f(w) = \frac{1}{2\sigma^2} w^T X^T X w - w^T X^T y + \frac{1}{2\sigma^2} y^T y + w^T \Lambda w$
   $\nabla f(w) = \frac{1}{\sigma^2} X^T X w - X^T y + 2\Lambda w$
   $\Rightarrow (\frac{X^T X}{\sigma^2} - 2\Lambda) w = X^T y$ and if we assume $X^T X$ is invertible we have:
   $w = (\frac{X^T X}{\sigma^2} - 2\Lambda)^{-1} X^T y$

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:
   $f(w) = \frac{1}{2}(Xw-y)^T V (Xw-y) + \frac{1}{2}(w-u)^T \Lambda (w-u)$
   $= \frac{1}{2}(w^T X^T V X w - 2w^T X^T V y + y^T V y) + \frac{1}{2}(w^T \Lambda w - 2w^T \Lambda u + u^T \Lambda u$
   $\nabla f(w) = X^T V w - X^T V y + \Lambda w - \lambda u$
   $\Rightarrow (X^T V X + \Lambda) w = X^T V y + \Lambda u$ and if we assume $X^T V X$ is invertible we have:
   $w = (X^T V X + \Lambda)^{-1} (X^T V y + \Lambda u)$

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:
   $\|w\|_\infty = \max_i |w^i| = \max_i \sqrt{|w^i|^2} \leq \sqrt{\sum_i |w^i|^2} = \|w\|_2$
   $\|w\|_2 = \sqrt{\sum_i |w^i|^2} \leq \sum_i \sqrt{|w^i|^2} = \sum_i |w^i| = \|w\|_1$

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

   Answer:
   $0 \leq (\|w\| - \|u\|)^2 = \|w\|^2 + \|u\|^2 - 2\|w\|\|v\|$
   $2\|w\|\|v\| \leq \|w\|^2 + \|u\|^2$
   $2\|w\|\|v\| + \|w\|^2 + \|u\|^2 \leq 2(\|w\|^2 + \|u\|^2)$
   $\frac{1}{2}\|w+u\|^2 \leq \|w\|^2 + \|u\|^2$

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

   We will use the following properties:

   (a) $\|X\|_2 = \sqrt{\lambda_{max}(X^T X)}$

   (b) $\|X\|_F = \sqrt{Tr(X^T X)}$

   (c) $Tr(X) = \sum_i \lambda_i$

   Answer:

   $$\|X\|_2 = \sqrt{\lambda_{max}(X^T X)} \ (a)$$

$$\leq \sqrt{Tr(X^T X)} \ (c)$$
$$= \|X\|_F \ (b)$$

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:
$$f(w) = -\sum_{i=1}^n \log\left[\frac{1}{2}\exp(-|w^T x^i - y^i|)\right] - \log\left[\frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}\sum_{j=1}^d \frac{w_j^2}{\sigma_j^2})\right]$$
$$= -\sum_{i=1}^n \left[\log\frac{1}{2}\right] + \sum_{i=1}^n |w^T x^i - y^i| - \log\left[\frac{1}{\sqrt{2\pi}}\right] + \frac{1}{2}\sum_{j=1}^d \frac{w_j^2}{\sigma_j^2}$$
$$= \gamma + \sum_{i=1}^n |w^T x^i - y^i| + \frac{1}{2}\sum_{j=1}^d \frac{w_j^2}{\sigma_j^2}$$
$$\propto \|Xw - y\|_1 + \frac{1}{2}\sum_{j=1}^d \frac{w_j^2}{\sigma_j^2}$$

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:

$$f(w) = -\sum_{i=1}^{n} \log\left[\frac{1}{\sqrt{\nu}B\left(\frac{1}{2},\frac{\nu}{2}\right)}\left(1+\frac{(w^\top x^i - y^i)^2}{\nu}\right)^{-\frac{\nu+1}{2}}\right] - \log\left[\frac{1}{\sqrt{2\pi}}\exp(-\frac{\lambda}{2}\sum_{j=1}^{d}(\mu^j - w^j)^2\right]$$

$$= -\sum_{i=1}^{n} \log\left[\frac{1}{\sqrt{\nu}B\left(\frac{1}{2},\frac{\nu}{2}\right)}\right] + \frac{\nu+1}{2}\sum_{i=1}^{n}\log\left(1+\frac{(w^\top x^i - y^i)^2}{\nu}\right) - \log\left[\frac{1}{\sqrt{2\pi}}\right] + \frac{\lambda}{2}\sum_{j=1}^{d}(\mu^j - w^j)^2$$

$$= \gamma + \frac{\nu+1}{2}\sum_{i=1}^{n}\log\left(1+\frac{(w^\top x^i - y^i)^2}{\nu}\right) + \frac{\lambda}{2}\sum_{j=1}^{d}(\mu^j - w^j)^2$$

$$\propto \frac{\nu+1}{2}\sum_{i=1}^{n}\log\left(1+\frac{(w^\top x^i - y^i)^2}{\nu}\right) + \frac{\lambda}{2}\|\mu - w\|^2$$

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.)

Answer:

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

$$= -\sum_{i=1}^{n} y^i w^\top x^i + \sum_{i=1}^{n}\exp(w^\top x^i) - \sum_{i=1}^{n}\log y^i! - d\log\kappa$$

$$= \gamma - \sum_{i=1}^{n} y^i w^\top x^i + \sum_{i=1}^{n}\exp(w^\top x^i)$$

$$\propto -\sum_{i=1}^{n} y^i w^\top x^i + \sum_{i=1}^{n}\exp(w^\top x^i)$$

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

1. Regularized and tilted Least Squares

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

wher u is $d \times 1$.

Answer:

$$= \frac{1}{2}w^T X^T X w - w^T X^T y + \frac{1}{2}y^T y + \frac{\lambda}{2}w^T w + w^T u$$

$$\nabla f(w) = X^T X w - X^T y + \lambda w + u$$

$$\nabla^2 f(w) = X^T X + \lambda I$$

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

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

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).

Answer:

$$= \frac{1}{2}(Xw - y)^T V (Xw - y) + \frac{1}{2}w^T \Lambda w = \frac{1}{2}w^T X^T V X w - w^T X^T V y + \frac{1}{2}y^T V y + \frac{1}{2}w^T \Lambda w$$

$$\nabla f(w) = X^T V X w - X^T V y + \Lambda w$$

$$\nabla^2 f(w) = X^T V X + \Lambda$$

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: First we will define $f(w) = \sum_{i=1}^{n} g(h(k(w)))$ where $g(h) = h^2$, $h(k) = \max\{0, 1 - y^i k\}$, $k(w) = w^T x$.

$$\Rightarrow \frac{\partial f(w)}{\partial w^j} = \frac{1}{2}\sum_{i=1}^{n}\left[\frac{\partial g}{\partial h}^{(i)} \frac{\partial h}{\partial k}^{(i)} \frac{\partial k}{\partial w^j}^{(i)}\right]$$

$$\frac{\partial g}{\partial h} = 2h$$

$$\frac{\partial h}{\partial k} = \begin{cases} -y^i & y^i k < 1 \\ 0 & y^i z \geq 1 \end{cases}$$

$$\frac{\partial k}{\partial w^j} = x^{i,j}$$

$$\Rightarrow \frac{\partial f}{\partial w^j} = \sum_{i=1}^{n}\left[\begin{cases} -y^i x^{i,j} \max\{0, 1 - y^i w^T x^i\} & y^i w^T x^i < 1 \\ 0 & y^i w^T x^i \geq 1 \end{cases}\right]$$

$$= \sum_{i=1}^{n} \left[\begin{cases} -y^i x^{i,j}(1 - y^i w^T x^i) & y^i w^T x^i < 1 \\ 0 & y^i w^T x^i \geq 1 \end{cases} \right]$$

$$\Rightarrow \nabla f(w) = -(X \circ y)^T r$$

Where $r \approx \vec{1} - y \circ Xw$ but all negative elements are set to 0

$$\frac{\partial^2 f}{\partial w_j^2} = \sum_{i=1}^{n} \left[\begin{cases} (y^i x^{i,j})^2 & y^i w^T x^i < 1 \\ 0 & y^i w^T x^i \geq 1 \end{cases} \right]$$

$$\nabla^2 f(w) = (X \circ y)^T (y \circ X)$$

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

CODE / PLOTS AT END OF DOCUMENT

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.

[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`

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 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 $\sigma = 1$.

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:

   (a) Training cost = matrix multiplication + solve $d \times d$ linear system = $O(nd^2 + d^3)$. Classifying new examples = $O(td)$

   (b) Training cost = compute d distances + solve $n \times n$ linear system $O(n^2 d + n^3)$. Classifying new examples = $O(tnd)$

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.

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.

Answer:
By varying the shape parameter we can increase the "width" of the Gaussian bumps under the curve. This will effectively smooth our our predicted function and help account for non-evenly sampled data.

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).

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 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.

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 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:
   First introduce a dummy scalar variable r where $r = \|Xw - y\|_\infty$ then we can formulate the LP as:

   $$\min_{x,y} r$$

   $$s.t.\ max\{(Xw - y)_k, -(Xw - y)_k\} \leq r,\ k = 1, ...$$

   $$\Rightarrow \min_{w,r} \begin{bmatrix} 0 \\ 1 \end{bmatrix}^T \begin{bmatrix} w \\ r \end{bmatrix}$$

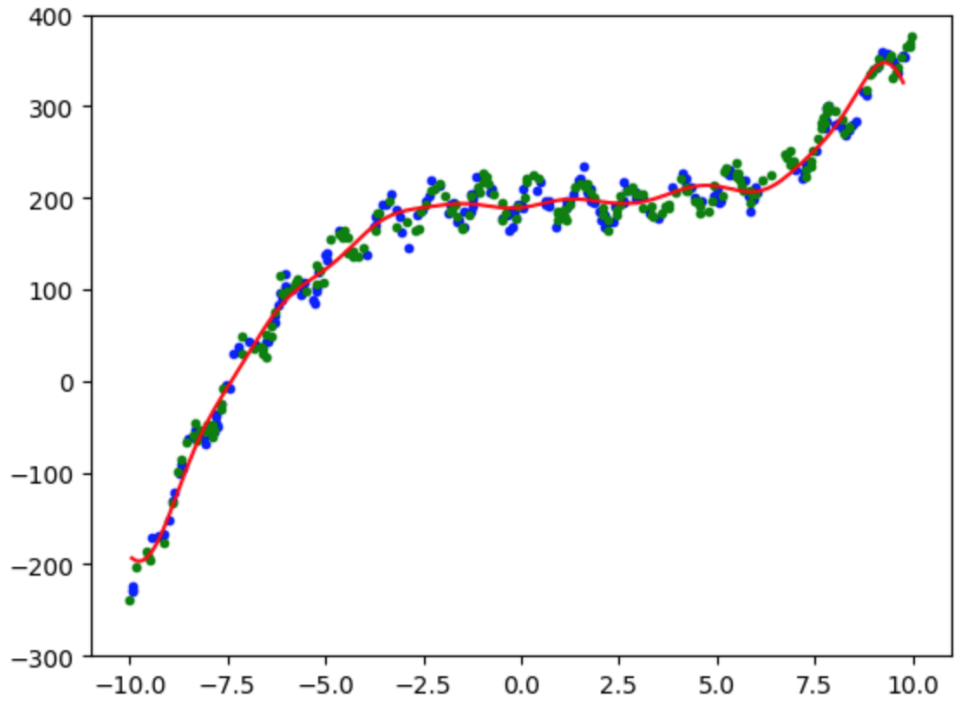
   $$s.t. \begin{bmatrix} X & -1 \\ -X & -1 \end{bmatrix} \begin{bmatrix} w \\ r \end{bmatrix} \leq \begin{bmatrix} y \\ -y \end{bmatrix}$$

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.

4 Code and plots

3.1.1

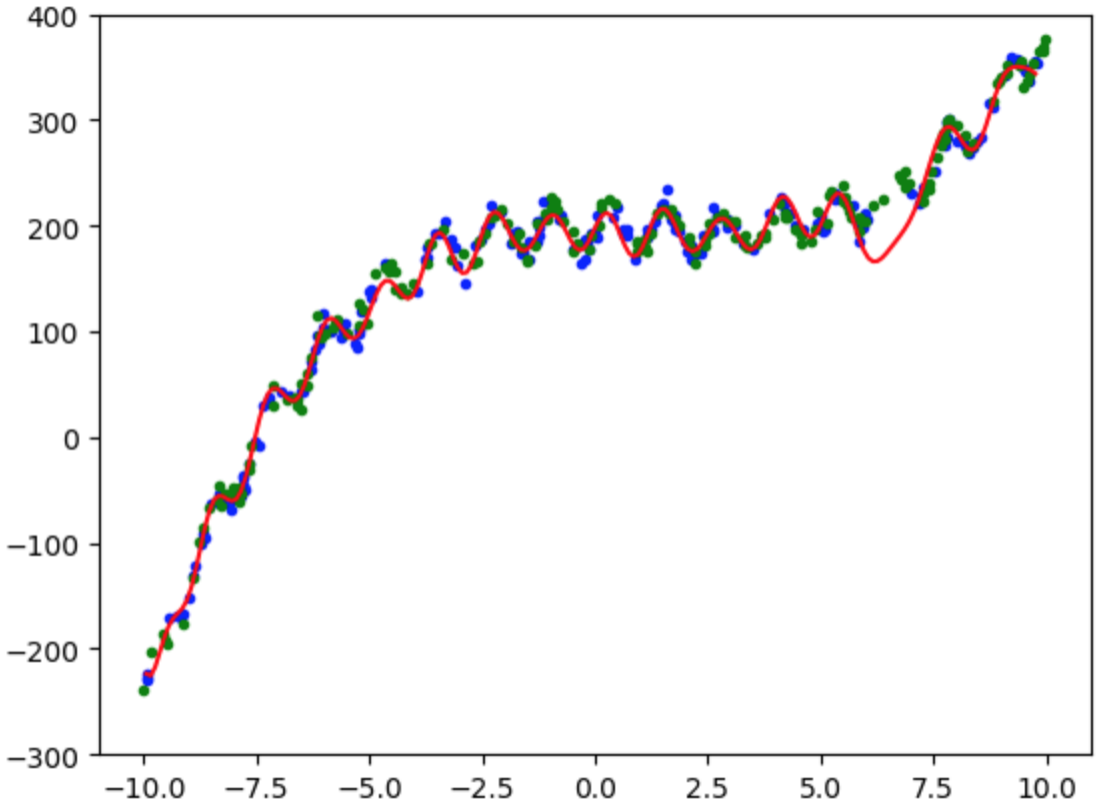
```
function leastSquaresRBFL2(X,y,sigma,lambda)
    # create rbf
    rbf = rbfBasis(X,X,sigma)

    # Add bias column
    n = size(rbf,1)
    Z = [ones(n,1) rbf]

    # Find regression weights minimizing squared error
    w = (Z'*Z + I*lambda)\(Z'*y)

    # Make linear prediction function
    predict(Xtilde) = [ones(size(Xtilde,1),1) rbfBasis(Xtilde,X,sigma)]*w

    # Return model
    return LinearModel(predict,w)
end

## Compute gaussian rbf of given dataset with given variance
function rbfBasis(Xi, Xj, sigma)
    return exp.(-distancesSquared(Xi,Xj) / 2sigma^2)
end
```



TestError = 356.07

3.1.3

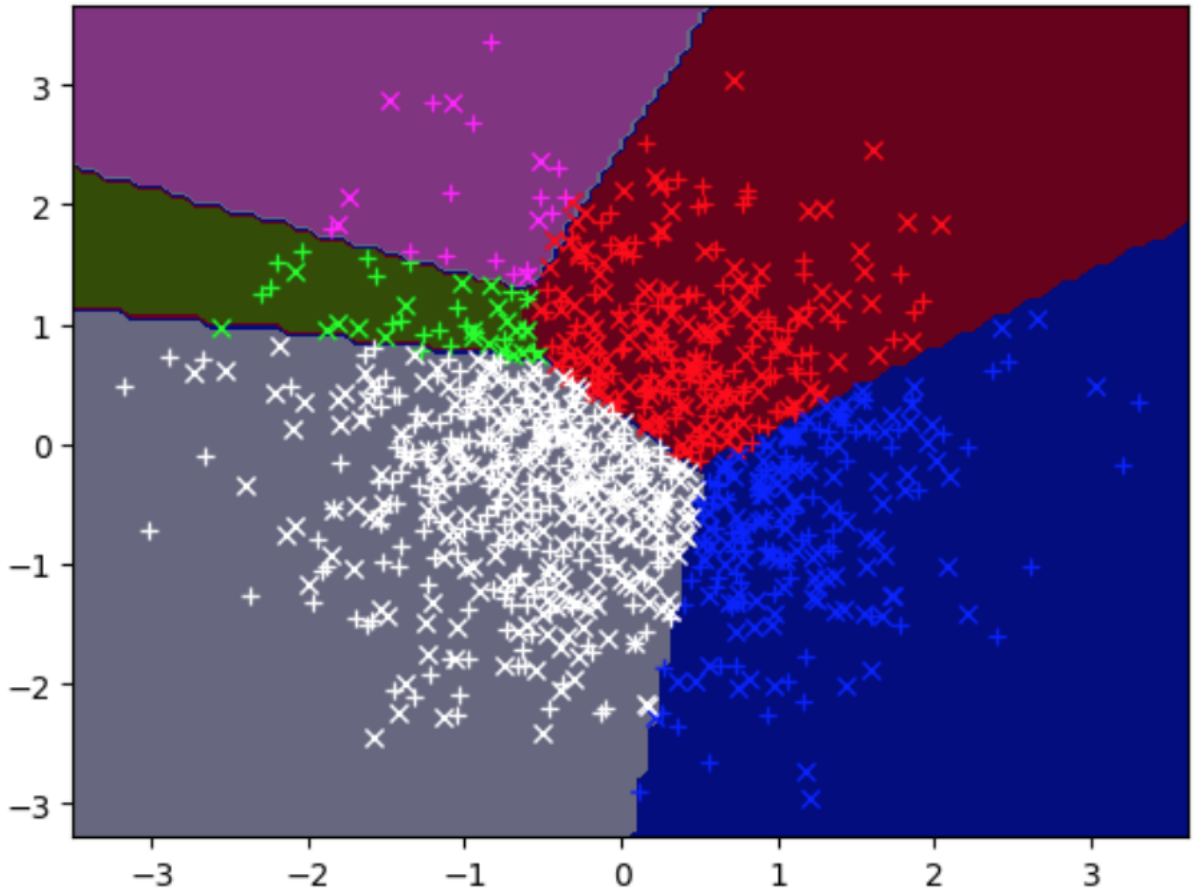
```
function leastSquaresRBFL2CV(X,y)
    # split data into training and test
    Xtrain, ytrain, Xvalid, yvalid = partitionTrainTest(X,y)

    # find best hyperparams
    sigma,lambda = gridSearch(Xtrain, ytrain, Xvalid, yvalid)

    return leastSquaresRBFL2(Xtrain,ytrain,sigma,lambda)
end

## Randomly split data
function partitionTrainTest(data, y, train_perc = 0.7)
    n = size(data,1)
    mid = Int(ceil(n/2))
    idx = collect(1:n)
    rand_idx = idx[randperm(length(idx))]
    trainIdxs = rand_idx[1:mid]
    testIdxs = rand_idx[mid+1:end]
    return data[trainIdxs,:], y[trainIdxs,:], data[testIdxs,:], y[testIdxs,:]
end

function gridSearch(Xtrain,ytrain,Xtest,ytest)
    # set defaults
    bestError = typemax(Int32)
    bestLamb = nothing
    bestSig = nothing

    # iterate over possible values of   ,
    for curSig in exp10.(range(-2, stop=2, length=10))
        for curLamb in exp10.(range(-3, stop=3, length=10))

            # compute weights
            model = leastSquaresRBFL2(Xtrain,ytrain,curSig,curLamb)

            # calculate error
            yhat = model.predict(Xtest)
            t = size(Xtest,1)
            curError = sum((yhat - ytest).^2)/t

            # update defaults
            if curError < bestError
                bestError = curError
                bestLamb = curLamb
                bestSig = curSig
            end
        end
    end
    return bestSig, bestLamb
end
```



TestError = 206.75
sigma = 0.215443, lambda = 0.001000

3.2

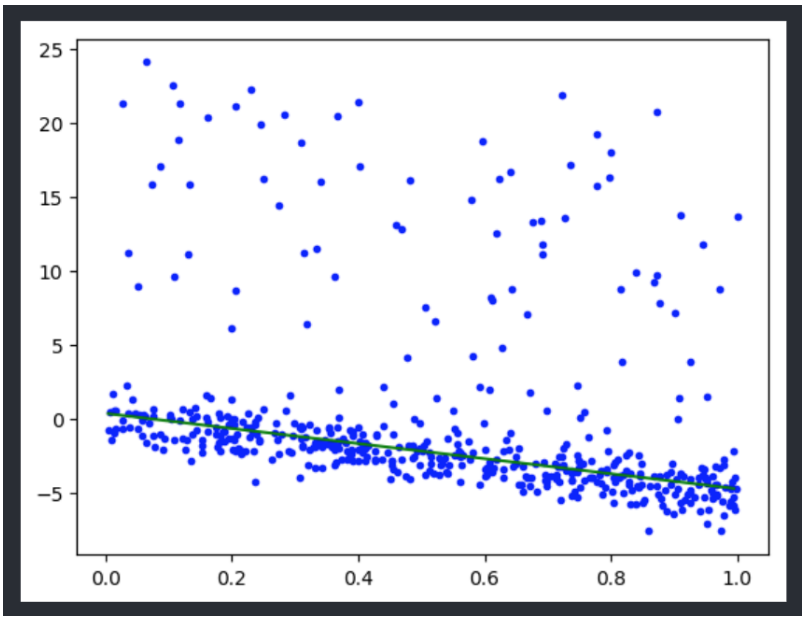
```
function softmaxObj(w,X,y)
    # X dimensions and number of classes
    n,d = size(X)
    k = maximum(y)

    ## create masking matrix
    mask = falses(n, k)
    for i = 1:length(y)
        mask[i,y[i]] = true
    end

    # compute XW
    W = reshape(w, (d,k))
    XW = X*W

    # compute inner function
    sumExp = sum(exp.(XW), dims=2)

    # compute function
    f = sum(-XW[mask] + log.(sumExp))

    # compute gradient
    g = X'*((exp.(XW) ./ sumExp) - mask)

    return (f,g[:])
end

function softmaxClassifier(X,y)
    n,d = size(X)
    k = size(unique(y),1)

    w = zeros(d*k)

    funObj(w) = softmaxObj(w,X,y)

    w = findMin(funObj,w,verbose=false)

    # Make linear prediction function
    W = reshape(w,(d,k))
    predict(Xhat) = mapslices(argmax,Xhat*W,dims=2)

    return LinearModel(predict,W)
end
```



trainError = 0.004
validError = 0.026

3.3.1

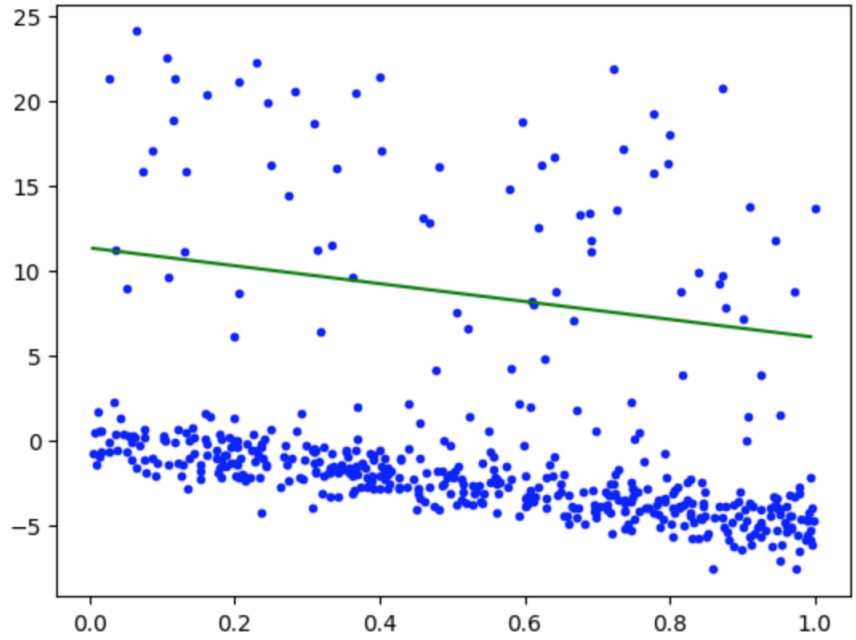
```
function leastAbsolutes(X,y)
    # resphape y
    y = reshape(y, length(y))

    # Add bias column
    n = size(X,1)
    Z = [ones(n,1) X]
    m = size(Z,2)

    # prepare LP
    A = [Z -I ; -Z -I]
    c = [zeros(m); ones(n)]
    b = [y ; -y]
    d = -Inf*ones(size(A,1))
    lb = -Inf*ones(n+m)
    ub = Inf*ones(n+m)

    # run through solver
    solution = linprog(c, A, d, b, lb, ub, GLPKSolverLP())
    w = solution.sol[1:m]

    # Make linear prediction function
    predict(Xtilde) = [ones(size(Xtilde,1),1) Xtilde]*w

    # Return model
    return LinearModel(predict,w)
end
```



Squared train Error with least absolutes: 40.437
Squared test Error with least absolutes: 1.194

3.3.2

```
function leastMax(X,y)
    # resphape y
    y = reshape(y, length(y))

    # Add bias column
    n = size(X,1)
    Z = [ones(n,1) X]
    m = size(Z,2)
    res = -ones(size(y))

    # prepare LP
    A = [Z res; -Z res]
    c = [zeros(m); 1]
    b = [y ; -y]
    d = -Inf*ones(size(b))

    # run through solver
    solution = linprog(c, A, d, b, -Inf, Inf, GLPKSolverLP())
    w = solution.sol[1:m]

    # Make linear prediction function
    predict(Xtilde) = [ones(size(Xtilde,1),1) Xtilde]*w

    # Return model
    return LinearModel(predict,w)
end
```



Squared train Error with least max: 112.612
Squared test Error with least max: 127.454