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: Erik Arne Huso
2. Student ID: 91227694
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: A model is fitted using a training set, meaning that a model with 97 % training accuracy is tested using already seen data. What matters is the model performance when tested on unseen data, test error.

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

   Answer: The test set error is computed by applying the model on a test set, while the test error is the expected test error given all possible test sets.

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 is most likely not meaningful. With higher model complexity of a model, the risk of overfitting increases. This is probably the case here, with the model being useless on any other data set.

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 n of training examples increases, the amount of information also increases and the test error should decrease as overfitting becomes less likely . The training error may increase since it might be harder to fit a model to an increasing amount of data, but in general the approximation error will decrease.

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: Fixing a tree depth will make the model parametric, since the complexity won't depend on n. Setting the tree depth to infinity would make the model non-parametric, as the tree depth would increase with the amount of data.

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: XGBoost would rather use squared error because this loss term is easier to optimize since it is convex and smooth while number of training errors is not.

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: A situation where it would be better to use gradient descent rather than Newton's method would be when the second derivative is not known, complicated or computationally infeasible, as Newton's method uses knowledge of the second derivative.

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: Sparsity: when λ goes to infinity, the sparsity increases. Training error: when λ increases, the model complexity decreases and the training error increases. Approximation error: when λ increases, the test error goes down as the chance of overfitting decreases with simpler model complexity. The approximation error will thus decrease until optimal λ is found, then increase again until reaching a constant level.

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 optimum the algorithm finds is most likely a local optimum, which is easier to find than a global optimum.

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 measures the sensitivity of a linear system to variations in data. This is done by measuring the product of the euclidean norm of the matrix and its inverse: $\kappa(X) = ||X||_2 \cdot ||X^{-1}||_2$.

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

Answer: In a multi-class logistic regression the weight matrix will consist of kd where k is the number of classes and d is the number of features plus bias variable.

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

    Answer: Sigmoid likelihood could be used in a case where the response variable is binary. Poisson likelihood could be used when modelling count data.

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 can be solved by splitting the problem up into subproblems recursively. One would find the minimum cost of each subproblem and save it, add these together and do this for every possible split and take the minimum of them all. This is dynamic programming and results in a run time cost of $O(k^3)$.

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 $\hat{y}^i = v^\top Wx^i$ 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 particular neural network would make linear predictions, it would thus be reasonable that the training error is the same as for the linear model.

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: The network uses stochastic gradient descent to find weights, while the linear model could use the normal equations to find an unique and optimal solution. The weights of the neural network might not be optimal since they're found with SGD and thus the training error might be the same or higher, but not lower.

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

    Answer: A convolutional neural network is less complex than an ordinary neural network, which reduces the chance of overfitting. In addition, the pooling one would usually find in convolutional neural network also helps reducing overfitting.

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: By using the given information, we write

$$f(w) = \frac{1}{2}||w - u||_\Sigma = \frac{1}{2}||\Sigma^{1/2}(w - u)||_2. \tag{1}$$

Then,

$$\nabla f(w) = \frac{\Sigma(w-u)}{||\Sigma^{1/2}(w-u)||_2}. \tag{2}$$

Since any euclidean norm is bigger than zero, we can simplify the expression to be

$$f(w) \tag{3}$$

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: We find the gradient:

$$\nabla f(w) = \frac{1}{\sigma^2} X^T(Xw-y) + 2\Lambda w \tag{4}$$

   , and furthermore

$$w = \big(\frac{1}{\sigma^2}X^TX + 2\Lambda\big)^{-1}\frac{1}{\sigma^2}X^Ty. \tag{5}$$

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: We rewrite the equation as

$$f(w) = \frac{1}{2}V||Xw-y||^2 + \frac{1}{2}\Lambda||w-u||^2. \tag{6}$$

   The gradient is then

$$\nabla f(w) = X^TVXw - X^TVy + \Lambda(w-u). \tag{7}$$

   Finally, solve for w to obtain

$$w = \big(X^TVX + \Lambda\big)^{-1}\big(X^TVy + \Lambda u\big). \tag{8}$$

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

   Answer: We start by showing the first inequation:

$$||x||_\infty = \max_n |x_i| = \max_n \sqrt{|x_i|^2} \le \sqrt{\sum_{i=1}^n |x_i|^2} = ||x||_2. \tag{9}$$

   Now, ahowing the second equation:

$$||x||_2 = \sqrt{\sum_i |x_i|^2} \le \sum_i \sqrt{|x_i|^2} = \sum_i |x_i| = ||x||_1. \tag{10}$$

   Since $||x||_\infty \le ||x||_2$ and $||x||_2 \le ||x||_1$ then also $||x||_\infty \le ||x||_1$ and the relationship is proven.

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

   Answer: We start with
   $$\left|||w|| - ||u||\right|^2 = \left(||w|| - ||u||\right)^2 \geq 0. \tag{11}$$
   Then, by expanding the terms:
   $$||w||^2 - 2||w||||u|| + ||u||^2 \geq 0. \tag{12}$$
   Moving the cross-term to the left hand side and adding the squared terms on both sides yields
   $$2||w||^2 + 2||u||^2 \geq ||w||^2 + 2||w||||u|| + ||u||^2 \geq ||w||^2 + 2wu + ||u||^2, \tag{13}$$
   where the last inequality is a result of using Cauchy-Schwartz. Then,
   $$2||w||^2 + 2||u||^2 \geq ||w+u||^2, \tag{14}$$
   and the inequality is proven by dividing both sides by 2.

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

   Answer: Using the fact that $||X||_2 = \sigma_{max}(X)$ and that $||X||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{d}|x_{ij}|^2} = \sqrt{\sum_{c=1}^{min(n,d)}\sigma_c(X)^2}$, we get
   $$||X||_2 = \sigma_{max}(X) \leq \sqrt{\sum_{c=1}^{min(n,d)}\sigma_c(X)^2} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{d}|x_{ij}|^2} = ||X||_F. \tag{15}$$

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 $\|x\|_1 = x^\top \text{sign}(x)$, 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), and to use that $\|X\|_2 = \sigma_1(X)$ (the top singular value).

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: We have
   $$p(y_i|w, x_i) = \frac{1}{2}exp(-|w^t x_i - y_i|) \tag{16}$$

and

$$p(w_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} exp(-\frac{w^2}{2\sigma_j^2}). \tag{17}$$

Using the procedure of negative log-likelihood, we obtain

$$f(w) = ||Xw - y||_1 + \frac{\Sigma^{-2}}{2}||w||^2. \tag{18}$$

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: Using NLL We have

$$p(y_i|w, x_i) = \frac{\nu+1}{2}\sum_{i=1}^{n} log(1 + \frac{(w^T x_i - y_i)^2}{\nu}), \tag{19}$$

and

$$p(w_j) = \frac{\lambda}{2}\sum_{j=1}^{n}(w_j - u)^2. \tag{20}$$

Thus, we get

$$f(w) = \frac{\nu+1}{2}\sum_{i=1}^{n} log(1 + \frac{(w^T x_i - y_i)^2}{\nu}) + \frac{\lambda}{2}||w - u||^2. \tag{21}$$

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,

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

(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: Using NLL we have

$$l(w) = (y_i w^T x_i) - exp(w^T x_i) - log(y_i!) + \kappa, \tag{22}$$

so our loss and regularizer can be written as

$$f(w) = \sum_{i=1}^{n}(-y_i w^T x_i + exp(w^T x_i)). \tag{23}$$

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

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

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: Differentiating with respect to w yields

$$\nabla f(w) = X^T X w - X^T y + \lambda w + u \tag{24}$$

and Hessian

$$\nabla^2 f(w) = X^T X + \lambda I. \tag{25}$$

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: We get

$$\nabla f(w) = X^T V X w - X^T V y + \Lambda w \tag{26}$$

and Hessian

$$\nabla^2 f(w) = X^T V X + \Lambda. \tag{27}$$

3. Squared hinge loss,

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

(Technically, the second derivative isn't everywhere, so just give an expression that works for locations where it is defined.)

Answer: We split the derivatives into two. For the first derivative we get

$$\nabla f = \sum_i -(1 - y^i w^T x^i) y^i x^i \tag{28}$$

when $y^i w^T x^i < 1$ and 0 otherwise. Differentiating once more we get

$$\nabla^2 f = \sum_i (y^i)^2 x^i \cdot x^i \tag{29}$$

when $y^i w^T x^i < 1$ and the zero matrix otherwise.

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 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: The function can be found in the `code` folder. The plot generated is shown in figure 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) For the linear basis, training costs $O(nd + d^3)$ by solving the linear equations. Classifying t examples costs $O(dt)$.

[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`

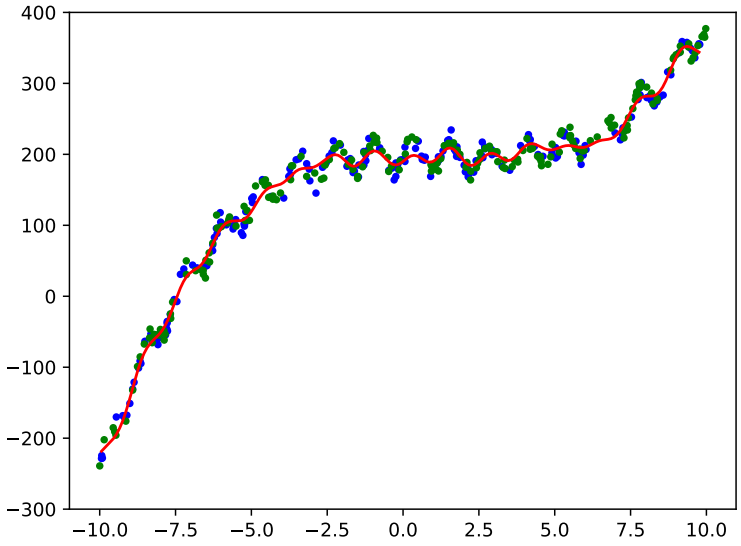

Figure 1: Least squares with gaussian RBF and L2-regularization, with $\lambda = 1$ and $\sigma = 1$.

(b) For the Gaussian RBF, forming Z takes $O(n^2d)$, finding n^2 distances of length d. Solving the linear system costs $O(n^3 + n^2 + nd^2)$. Forming $\tilde{Z}$ costs $O(ntd)$ and solving the system costs $O(nt)$, thus classifying t examples costs $O(ntd)$.

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.

   Answer: The code can be found in the *example_ nonLinear.jl* program. The resulting error was 61.88. The plot is shown in figure 2.

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: One way to adress a scenario when (i) is violated is to increase σ to make it more sensible to changes in the data structure. If (ii) is violated we can adjust the weights by adjusting λ and thus adjusting the weighting of the points.

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.

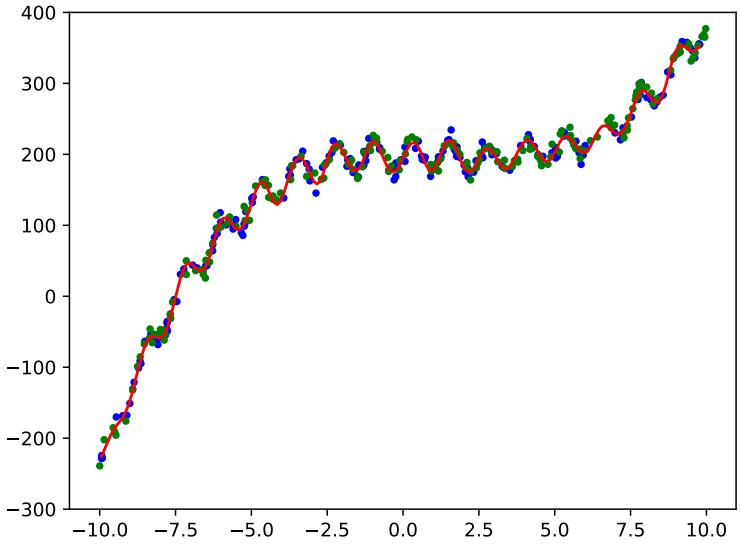

Figure 2: The plot with the lowest error, with $\lambda = 0.10$ and $\sigma = 0.61$.

Code for question 3.1:

```
function leastSquaresRBFL2(X, y, lambda, sigma)

        # Add bias column + RBF
        (n, d) = size(X)


        Xdist = distancesSquared(X, X)
        G = rbfBasis(lambda, sigma, Xdist)


        Z = [ones(n,1) X G]


        I = UniformScaling(1)

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

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

        return LinearModel(predict, v)
end


function rbfBasis(lambda, sigma, X)
```

```
        (n, d) = size(X)
        G = zeros(n, d)


        for i in 1:n
                for j in 1:d
                        G[i, j] = exp(- X[i,j]^2/(2 * sigma^2))
                end
        end

        return G
end

function crossValidate(X, y)
    n = size(X,1)

    # Random permutation for shuffle
    rng = MersenneTwister(1234);
    rndInd = randperm(n)
    mid = floor(Int, n/2)

    # Random shuffle, using the same permutation to conserve correct labels
    Xshuffle, yshuffle = X[rndInd], y[rndInd]

    # Dividing into train and validation set
    Xtrain, ytrain = Xshuffle[1:mid,:], yshuffle[1:mid]
    Xval, yval = Xshuffle[mid+1:end,:], yshuffle[mid+1:end]

    sigma = range(0,stop=10,length=1000)
    lambda = range(0,stop=100,length=1000)
    maxTestError = Inf #best:83.21

    bestSigma = sigma[1] #best: 0.61
    bestLambda = lambda[1] #best: 0.10 500 simulations
    for s in 2:length(sigma)-1
        for l in 2:length(lambda)-1

            model = leastSquaresRBFL2(Xtrain, ytrain, lambda[l], sigma[s])

            Xdist = distancesSquared(Xval,Xtrain)
            Gtest = rbfBasis(lambda[l], sigma[s], Xdist)
            Ztest =  [ Xval Gtest]

            t = size(Ztest,1)
            yhat = model.predict(Ztest)
            testError = sum((yhat - yval).^2)/t

            if (testError < maxTestError)
                maxTestError = testError
                bestSigma = sigma[s]
                bestLambda = lambda[l]
```

```
            end

        end
    end

    @printf("Best sigma = %.2f\n",bestSigma)
    @printf("Best lambda = %.2f\n",bestLambda)
    @printf("Best error = %.2f\n",maxTestError)

end
```

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.

Answer: The validation error with the softmax classifier is 0.026 and the plot can be seen in figure 3. 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).

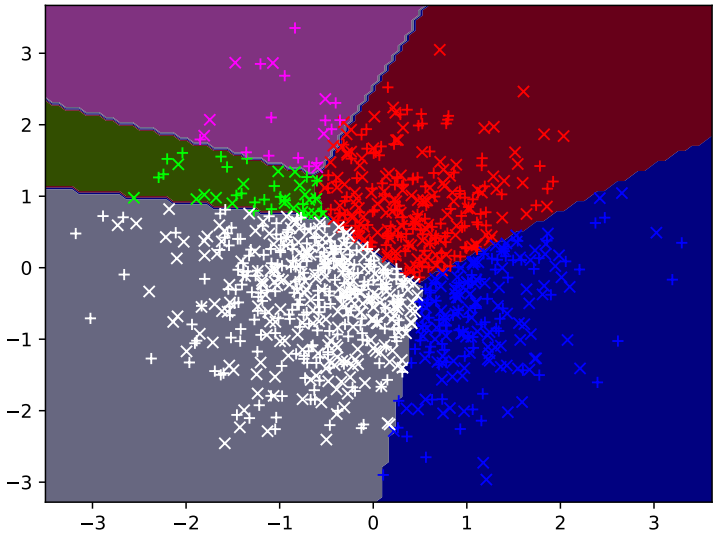

Figure 3: Softmax classifier for a data set with $k = 5$ classes.

Code for question 3.2:

```
    function softmaxClassifier(X, y)
        (n, d) = size(X)
        k = maximum(y)
        w = zeros(d*k)


        funObj(w) = softmaxObj(w, X, y)
        W = findMin(funObj, w, derivativeCheck=true,verbose=true)


        W = reshape(W, d, k)


        # Make linear prediction function
        predict(Xhat) = mapslices(argmax,Xhat*W,dims=2)
        return LinearModel(predict, W)
end

function softmaxObj(w, X, y)
        (n, d) = size(X)
        k = maximum(y)

        w = reshape(w, (d, k))
        f = 0
        g = zeros(d, k)
```

```
        #classSum = zeros(n)
        classSum = sum(exp.(X*w),dims=2)
        for i in 1:n

                f += -(w[:,y[i]]'*X[i,:]) + log.(classSum[i])
        end

        for c in 1:k
                for j in 1:d
                        for i in 1:n
                                g[j, c] += X[i,j] *
                                (exp.(w[:,c]'*X[i,:])/classSum[i] - (y[i]==c))
                        end

                end
        end

        g = reshape(g, (d*k))
        return (f,g)
end
```

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.

   Answer: The code can be found in the `leastSquares.jl` file and the plot is shown in figure 4.

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: Since the infinity norm returns a scalar, we can write the problem as

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

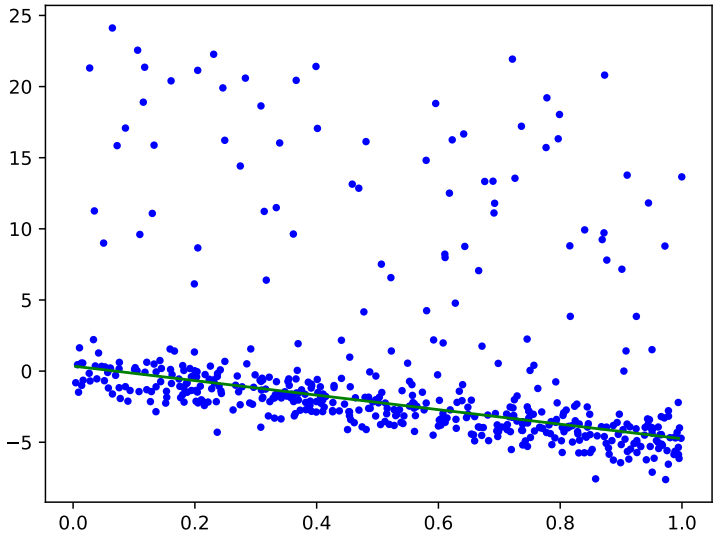

Figure 4: Least absolute squares.

or

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \max_i |w^T x_i - y_i| \leq t. \end{array}$$

This again can be written as

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & Xw - y \leq t\mathbb{1}_n \\ & y - Xw \leq t\mathbb{1}_n, \end{array}$$

where $\mathbb{1}_n$ is a vector of ones of length n. This would be the linear program to use for minimizing brittle regression.

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: The function can be found in `leastSquares.jl` and the plot is shown in figure 5.

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, and you can set values in $b/d/\text{lb}/\text{ub}$ (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.

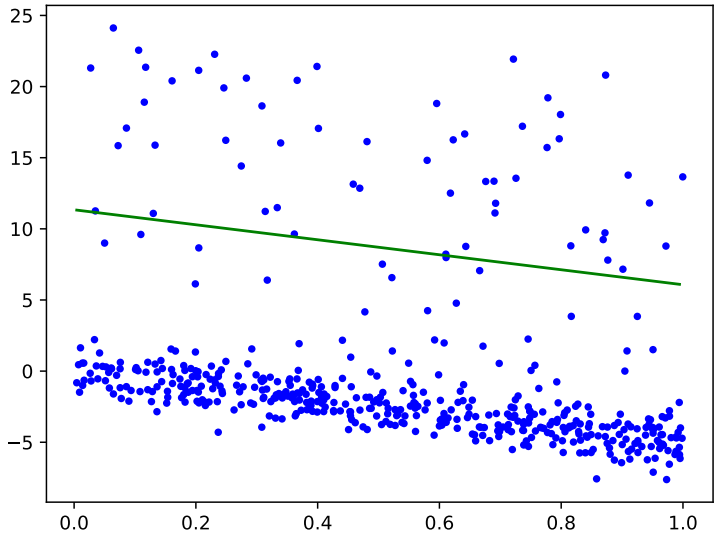

Figure 5: Brittle regression.

Code for question 3.3:

```
function leastAbsolutes(X, y)

        (n, d) = size(X)


        b = zeros(2*n)
        a = zeros(2*n)
        B = zeros(2*n)
        R = zeros(2*n,n)

        counter = 1
        for i in 1:2*n
                if (i % 2 == 0)
                        b[i] = y[counter]
                        a[i] = X[counter]
                        R[i, counter] = -1
                        B[i] = 1
                        counter = counter + 1
                else
                        b[i] = -y[counter]
                        a[i] = -X[counter]
                        R[i, counter] = -1
                        B[i] = -1
                end
        end
```

```
        A = [a B R ]

        zeroVec = zeros(1,2 )
        oneVec = ones(1,n)

        f = [zeroVec oneVec]

        f = reshape(f, length(f),1)
        f = vec(f)
        ub = Inf*ones(502)
        lb = zeros(502)
        lb[1] = -Inf
        d = -Inf*ones(1000)


        solution = linprog(f,A,d,b,lb,ub,GLPKSolverLP())
        w = solution.sol

        predict(Xtilde) = [Xtilde ones(size(Xtilde,1),1) ]*w[1:2]

        return LinearModel(predict,w)
end

function leastMax(X,y)
        (n, d) = size(X)
        b = zeros(2*n)
        a = zeros(2*n)
        B = zeros(2*n)
        R = ones(2*n,n)
        counter = 1
        for i in 1:2*n
                if (i % 2 == 0)
                        b[i] = y[counter]
                        a[i] = X[counter]
                        #R[i, counter] = -1
                        B[i] = 1
                        counter = counter + 1
                else
                        b[i] = -y[counter]
                        a[i] = -X[counter]
                        #R[i, counter] = -1
                        B[i] = -1
                end
        end
        r = -1*ones(2*n)
        A = [a B r ]
        zeroVec = zeros(1,2 )
        oneVec = ones(1,1)
        f = [zeroVec oneVec]
        f = reshape(f, length(f),1)
```

```
        f = vec(f)
        ub = Inf*ones(length(f))
        lb = zeros(length(f))
        lb[1] = -Inf
        d = -Inf*ones(2*n)
        solution = linprog(f,A,d,b,lb,ub,GLPKSolverLP())
        w = solution.sol
        predict(Xtilde) = [Xtilde ones(size(Xtilde,1),1) ]*w[1:2]
        return LinearModel(predict,w)
end
```