CPSC 540 Assignment 1 (due January 12th at midnight)

IMPORTANT!!!!! Before proceeding, please carefully read the homework instructions:
`www.cs.ubc.ca/~schmidtm/Courses/540-F17/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:
2. Student ID:
3. Faculty (e.g., applied science):
4. Department (e.g., computer science):
5. 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-sentence answer to the below questions.

1. Why is the IID assumption important in supervised learning?

   Answer: Under this assumption, we expect our test data to behave similarly to the training data.

2. Suppose we have a supervised learning problem where we think the examples x_i form clusters. To deal with this, we combine our training and test data together and fit a k-means model. We then add the cluster number as an extra feature, fit our supervised learning model based on the training data, then evaluate it on the test data. What have we done wrong?

   Answer: We violated the golden rule, as the test data has now influenced the training procedure.

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

   Answer: The test error is the expected error average over the entire data distribution, while a validation set error is the error we get on a set of samples from this distribution (ideally IID and not used to influence training).

4. Describe a setting where using a validation set to choose hyper-parameters can lead to overfitting.

   Answer: You try out a large of number of models on the validation set and choose the one that performs the best (and your validation set isn't sufficiently big).

5. What is the effect of the number of features d that our model uses on the training error and on the approximation error?

   Answer: As d increases the training error goes down but the approximation error goes up.

6. What is wrong with with using $\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i \neq \tilde{y}_i)$ as the validation error of a regression model?

   Answer: You probably won't be able to exactly fit the real-valued y^i values so this is likely to be one, you should something squared error to reflect that predictions are "close".

7. Describe a situation where it could be better to use gradient descent than the normal equations to solve a least squares problem.

   Answer: If d is large. (And gradient descent doesn't take too many iterations.)

8. How does λ in an L0-regularizer (like BIC) affect the sparsity pattern of the solution, the training error, and the approximation error?

   Answer: As λ increase the training error goes up, the sparsity goes up, and the approximation error goes down.

9. Minimizing the squared error with L0-regularization is NP-hard, what does this imply?

   Answer: We do not expect there to exist an algorithm with a polynomial runtime.

10. For a fixed target y, what is the likely effect of increasing the condition number of X (in a least squares problem) on the approximation error?

    Answer: As the condition number of X increases the approximation error should increase.

11. For supervised training of a linear model $w^T x^i$ with $y_i \in \{-1, +1\}$, why do we use the logistic loss intead of the squared error?

    Answer: The squared error penalizes for being "too right" (predicting $\hat{y}^i = 1000$ when $\tilde{y}^i = 1$ incurs a huge error, despite making the right decision).

12. What is the key difference between "one vs. all" logistic regression and training using the softmax loss?

    Answer: We train all of our parameters simultaneously (to encourage the maximum value of $w_c^T x_i$ to be $w_{y_i}^T x_i$).

13. Give a supervised learning scenario where you would use the Laplace likelihood and a scenario where you would use a Laplace prior.

    Answer: You have outliers (Laplace likelihood). You want a sparse solution (Laplace prior).

14. What do we use the backpropagation algorithm for?

    Answer: We need optimal and redundant sub-problems.

15. What are the two key properties of a problem that let us use dynamic programming?

    Answer: You can write the solution as a maximization over sub-problems, and these sub-problems are repeated with different parameters.

16. Consider a deep neural network with 1 million hidden units. Explain whether this is a parametric or a non-parametric model.

    Answer: Parametric because the size of the model is fixed as we increase n.

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

    Answer: Possible answers are sparsity in the weights, repeated values in the weights, or max pooling.

Above, we are referring to the "approximation error" of the training error, $(E_{\text{test}} - E_{\text{train}})$.

2 Calculation Questions

2.1 Minimizing Strictly-Convex Quadratic Functions

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

1. $f(w) = \frac{1}{2}(w-u)^T\Sigma(w-u)$ (projection of u onto the real space under Σ-norm).

   Answer:
$$w = u.$$

2. $f(w) = \frac{1}{2\sigma^2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2$ (ridge regression with known variance).

   Answer:
$$w = \left(\frac{1}{\sigma^2}X^TX + \lambda I\right)^{-1}\left(\frac{1}{\sigma^2}X^Ty\right),$$
   or it can be simplified to
$$w = (X^TX + \sigma^2\lambda I)^{-1}(X^Ty).$$

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

   Answer:
$$w = (X^TVX + \Lambda)^{-1}(X^TVy + \Lambda u).$$

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 $w \in \mathbb{R}^d$ and $u \in \mathbb{R}^d$:

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

   Answer: If x is an arbitrary vector in $\mathbb{R}^d$, then from the definition of the norms and non-negative of the absolute value we have
$$\|x\|_\infty^2 = \max_i |x_i|^2 \leq \sum_i |x_i|^2 = \|x\|_2^2,$$
   and
$$\|x\|_1^2 = \left(\sum_i |x_i|\right)^2 = \left(\sum_i |x_i|^2\right) + (|x_ix_j| \text{ terms})) \geq \sum_i |x_i|^2 = \|x\|^2.$$
   Taking square roots gives the inequalities.

2. $\|w\|_1 \leq \sqrt{d}\|w\|_2 \leq d\|w\|_\infty$ (relationship between increasing p-norms)

   Answer: First we have
$$\|x\|_1 = x^T \text{sign}(x) \leq \|x\|_2 \|\text{sign}(\text{x})\|_2 = \sqrt{d}\|x\|_2,$$
   where the inequality is Cauchy-Schwartz. Second we have
$$\|x\|_2^2 = \sum_i |x_i|^2 \leq \sum_i \max_i |x_i|^2 = d \max_i |x_i|^2 = d\|x\|_\infty^2.$$
   which if we take the square root and multiply by $\sqrt{d}$ gives the result.

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

   Answer: By expanding the left side this is equivalent to
$$\frac{1}{2}\|x\|^2 + x^T y + \frac{1}{2}\|y\|^2 \leq \|x\|^2 + \|y\|^2.$$
   Moving everything to the left side we get
$$-\frac{1}{2}\|x\|^2 + x^T y - \frac{1}{2}\|y\|^2 \leq 0,$$
   and completing the square we have
$$-\|x-y\|^2 \leq 0,$$
   which is true by non-negativity of norms.

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 and/or to use that $\|x\|_1 = x^T \text{sign}(x)$.

2.3 MAP Estimation

In 340, we showed that under the assumptions of a Gaussian likelihood and Gaussian prior,
$$y^i \sim \mathcal{N}(w^T x^i, 1), \quad w_j \sim \mathcal{N}\left(0, \frac{1}{\lambda}\right),$$
that the MAP estimate is equivalent to solving the L2-regularized least squares problem
$$f(w) = \frac{1}{2}\sum_{i=1}^n (w^T x^i - y^i)^2 + \frac{\lambda}{2}\sum_{j=1}^d w_j^2,$$
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. Gaussian likelihood with separate variance for each training example and Laplace prior
$$y^i \sim \mathcal{N}(w^T x^i, \sigma_i^2), \quad w_j \sim \mathcal{L}\left(0, \frac{1}{\lambda}\right).$$

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

or in matrix notation as

$$f(w) = \frac{1}{2}(Xw - y)^T \Sigma^{-1}(Xw - y) + \lambda \|w\|_1,$$

where Σ has the σ_i^2 along the diagonals.

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

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

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

   Answer:

$$f(w) = \frac{\nu+1}{2}\sum_{i=1}^{n} \log\left(1 + \frac{(w^T x^i - y^i)^2}{\nu}\right) + \frac{\lambda}{2}\sum_{j=1}^{d}(w_j - u_j)^2,$$

   or you could write the second term in matrix notation as $\frac{\lambda}{2}\|w - u\|^2$.

3. Time-independent censored survival analysis likelihood with per-variable Gaussian prior,

$$p(y^i, v^i|x^i, w) = \exp(v^i w^T x^i)\exp(-y^i \exp(w^T x^i)), \quad w_j \sim \mathcal{N}\left(0, \frac{1}{\lambda_j}\right).$$

   Here, y^i is a positive number giving the latest time that we observed patient i, and $v^i = 1$ if patient i has quit the study while $v^i = 0$ if they are still in it.[2]

   Answer:

$$f(w) = \sum_{i=1}^{n}\left(-v^i w^T x^i + y^i \exp(w^T x^i)\right) + \frac{1}{2}\sum_{j=1}^{d}\lambda_j w_j^2,$$

   or in matrix notation as

$$f(w) = -VXw + \sum_{i=1}^{n} y^i \exp(w^T x^i) + \frac{1}{2}w^T \Lambda w.$$

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. The quadratic function

$$f(w) = w^T u + u^T Aw + \frac{1}{2}w^T w + w^T Aw,$$

   wher u is $d \times 1$ and A is $d \times d$ (not necessarily symmetric).

   Answer:

$$\nabla f(w) = u + A^T u + w + Aw + A^T w,$$

[1]This likelihood is more robust than the Laplace likelihood, but leads to a non-convex objective.
[2]This likelihood can be used to estimate survival times when some patients are still alive.

or if you want to factorize you can write

$$\nabla f(w) = (I + A^T)u + (I + A + A^T)w.$$

$$\nabla^2 f(w) = I + A + A^T.$$

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

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

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:

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

or if you factorize you get

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

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

3. Weighted L2-regularized probit regression,

$$f(w) = -\sum_{i=1}^{n} \log p(y^i | x^i w) + \frac{1}{2}\sum_{j=1}^{d} u_j w_j^2.$$

where u is $d \times 1$, $y^i \in \{-1, +1\}$, and the likelihood of a single example i is given by

$$p(y^i | x^i, w) = \Phi(y^i w^T x^i).$$

where Φ is the cumulative distribution function (CDF) of the standard normal distribution.

Answer: Define the vector r with elements

$$r_i = \frac{p_i}{c_i}$$

which is y^i times the derivative of the log(CDF) with respect to its input. We then have

$$\nabla f(w) = -X^T(y \circ r) + Uw,$$

where I used $\circ$ as element-wise multiplication. (There are many variations like defining $r_i = -y_i \frac{p_i}{c_i}$ which would give $\nabla f(w) = X^T r$.) Let's apply the quotient rule to r_i to define a diagonal matrix D with elements

$$D_{ii} = -\frac{p_i'}{c_i} + \frac{p_i^2}{c_i^2},$$

where the derivative of the standard normal PDF would be

$$p_i' = -y^i w^T x^i p_i.$$

(I've actually taken the negative of the derivative, since I want the element D_{ii} to be positive.) This lets us write the Hessian as

$$\nabla^2 f(w) = X^T D X + U.$$

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, it is useful to define a vector c containing the CDF $\Phi(y^i w^T x^i)$ as element c_i and a vector p containing the corresponding PDF as element p_i. For the probit question you'll need to define new vectors to express the gradient and Hessian in matrix notation (and remember the relationship between the PDF and CDF). As a sanity check, make sure that your results have the right dimension.

3 Coding Questions

3.1 Regularization and Hyper-Parameter Tuning

Download *a1.zip* from the course webpage, and start Julia (verison of 0.6) 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:[3]

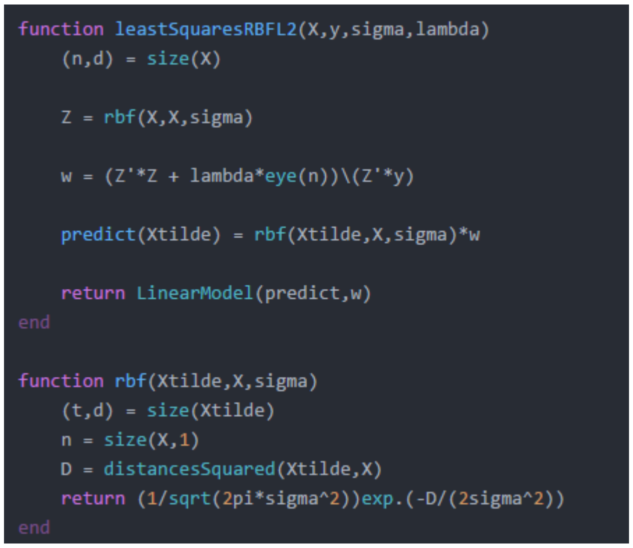
```
Pkg.add("JLD")
Pkg.add("PyPlot")
```

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 $\sigma = 1$.

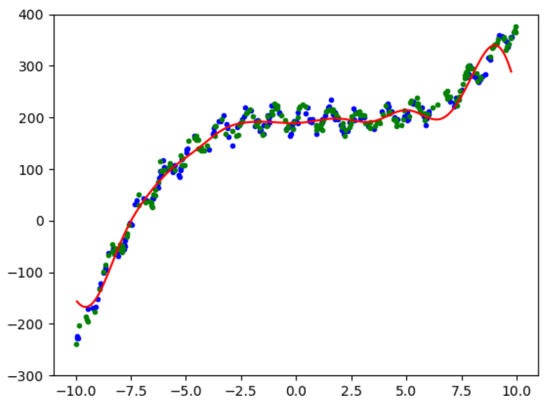
   Answer: The code could look like this:

[3]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`



```
function leastSquaresRBFL2(X,y,sigma,lambda)
    (n,d) = size(X)

    Z = rbf(X,X,sigma)

    w = (Z'*Z + lambda*eye(n))\(Z'*y)

    predict(Xtilde) = rbf(Xtilde,X,sigma)*w

    return LinearModel(predict,w)
end

function rbf(Xtilde,X,sigma)
    (t,d) = size(Xtilde)
    n = size(X,1)
    D = distancesSquared(Xtilde,X)
    return (1/sqrt(2pi*sigma^2))exp.(-D/(2sigma^2))
end
```

The plot should look like this:

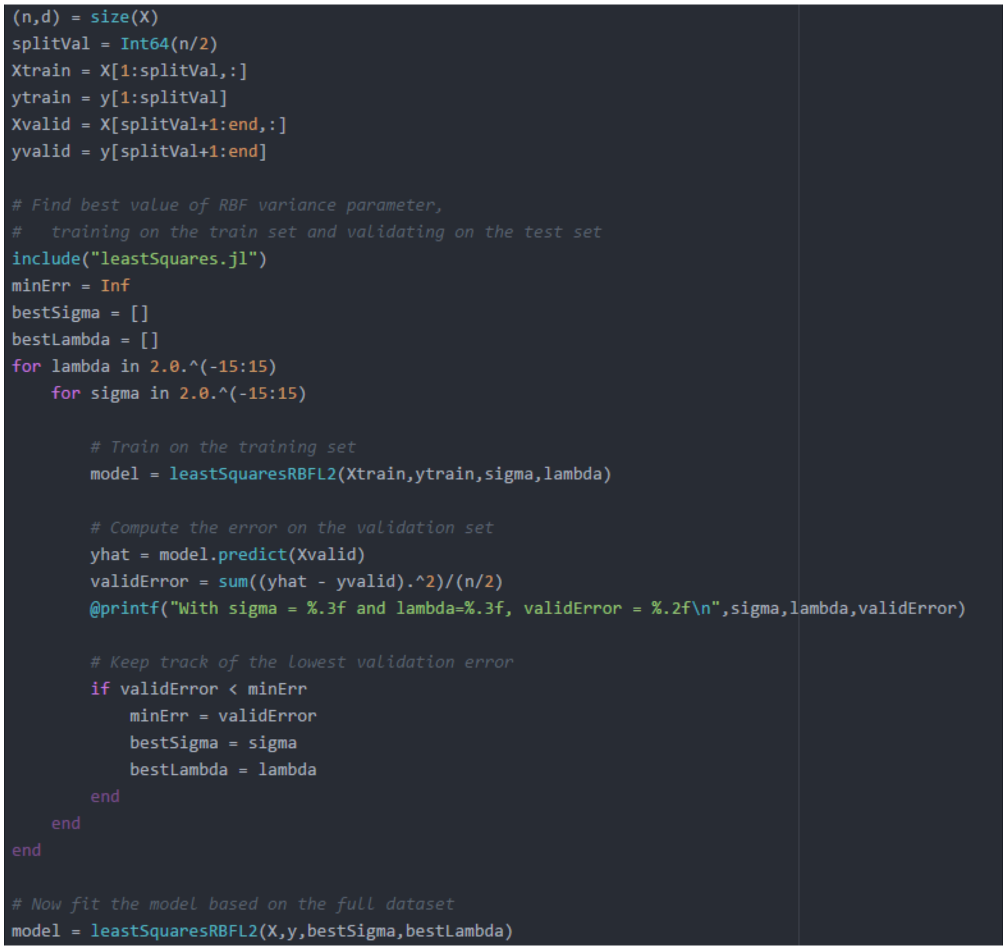


2. When dealing with larger datasets, an important issue is the dependence of the computational cost on the number of training examples n and the number of features d. What is the cost in big-O notation of training the model on n training examples with d features under (a) the linear basis, and (b) Gaussian 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: Under the linear basis, the dominant costs are: computing X^TX which costs $O(nd^2)$ to compute $O(d^2)$ inner products at a cost of $O(n)$ each, and inverting X^TX which costs $O(d^3)$ using standard methods. This gives a cost of $O(nd^2+d^3)$. Classifying t new examples costs $O(td)$ to compute t inner products at cost of $O(d)$ each. Using RBFs, forming Z costs $O(n^2d)$ to compute $O(n^2)$ distances at a cost of $O(d)$ each. Computing Z^TZ costs $O(n^3)$ while inverting it also costs $O(n^3)$ using standard methods. This gives a cost of $O(n^2d+n^3)$. Classifying a new example costs $O(tnd)$.

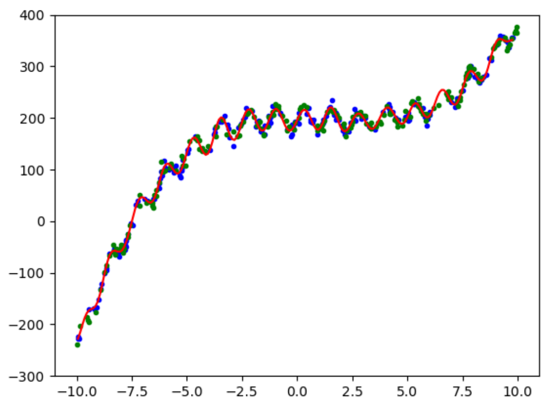
3. Modify the script to split the training data into a "train" and "validation" set (you can use half the examples for training and half for validation), and use these to select λ and σ. Hand in your modified script and the plot you obtain with the best values of λ and σ.

   Answer: Here is my implementation of the training/validation:



```
(n,d) = size(X)
splitVal = Int64(n/2)
Xtrain = X[1:splitVal,:]
ytrain = y[1:splitVal]
Xvalid = X[splitVal+1:end,:]
yvalid = y[splitVal+1:end]

# Find best value of RBF variance parameter,
#   training on the train set and validating on the test set
include("leastSquares.jl")
minErr = Inf
bestSigma = []
bestLambda = []
for lambda in 2.0.^(-15:15)
    for sigma in 2.0.^(-15:15)

        # Train on the training set
        model = leastSquaresRBFL2(Xtrain,ytrain,sigma,lambda)

        # Compute the error on the validation set
        yhat = model.predict(Xvalid)
        validError = sum((yhat - yvalid).^2)/(n/2)
        @printf("With sigma = %.3f and lambda=%.3f, validError = %.2f\n",sigma,lambda,validError)

        # Keep track of the lowest validation error
        if validError < minErr
            minErr = validError
            bestSigma = sigma
            bestLambda = lambda
        end
    end
end

# Now fit the model based on the full dataset
model = leastSquaresRBFL2(X,y,bestSigma,bestLambda)
```

The best values seem to be something like $\sigma = \frac{1}{2}$ and $\lambda \approx 10^{-5}$, which gives a plot that looks like this:



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: There are many possible answers and we'll accept anything reasonable. I don't think that the answers will depend on whether you focus on 1 or 2. Some possible answers are:

- Try to somehow define local values of σ.
- Use a basis that is only defined locally, as is done with splines.
- Abandon linear regression and use other non-parametric methods like locally-linear regression.

Later in the course, we'll see that Bayesian methods provide an elegant way to implement the first idea.

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,

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

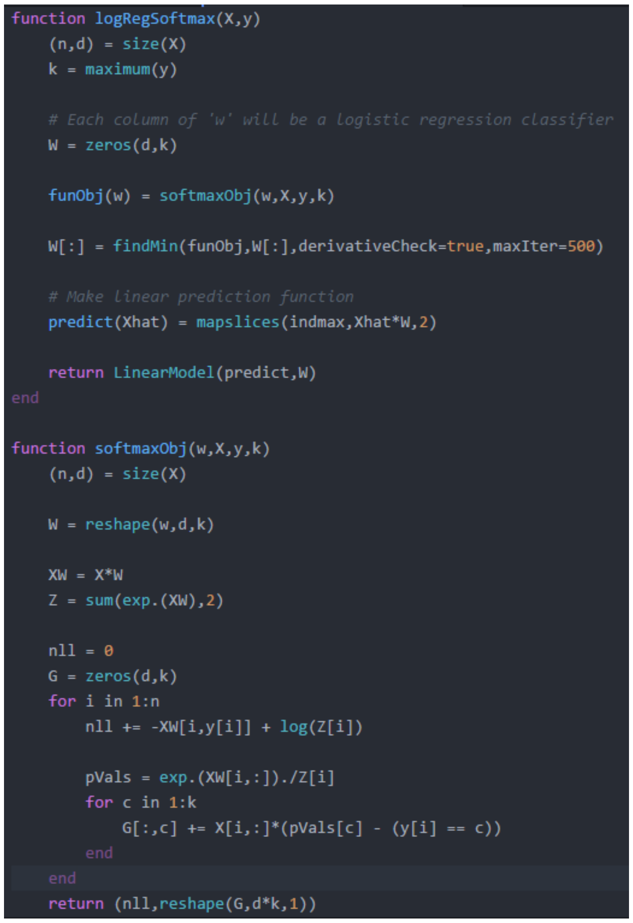
Here c is a possible label and w_c is column c of W. Similarly, y^i is the training label, w_{y^i} is column y^i of W. The loss function corresponding to the negative logarithm of the softmax probability is given by

$$f(W) = \sum_{i=1}^n \left[-w_{y^i}^T x^i + \log\left(\sum_{c'=1}^k \exp(w_{c'}^T x^i) \right) \right].$$

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* function to help you debug the gradient of the softmax loss.

Answer: The code should look something like this:



```
function logRegSoftmax(X,y)
    (n,d) = size(X)
    k = maximum(y)

    # Each column of 'w' will be a logistic regression classifier
    W = zeros(d,k)

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

    W[:] = findMin(funObj,W[:],derivativeCheck=true,maxIter=500)

    # Make linear prediction function
    predict(Xhat) = mapslices(indmax,Xhat*W,2)

    return LinearModel(predict,W)
end

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

    W = reshape(w,d,k)

    XW = X*W
    Z = sum(exp.(XW),2)

    nll = 0
    G = zeros(d,k)
    for i in 1:n
        nll += -XW[i,y[i]] + log(Z[i])

        pVals = exp.(XW[i,:])./Z[i]
        for c in 1:k
            G[:,c] += X[i,:]*(pVals[c] - (y[i] == c))
        end
    end
    return (nll,reshape(G,d*k,1))
```

The validation error depends on how precisely you solve the optimization problem, but it decreases to something very small like 0.01 or 0.03.

3.3 Robust and Brittle Regression

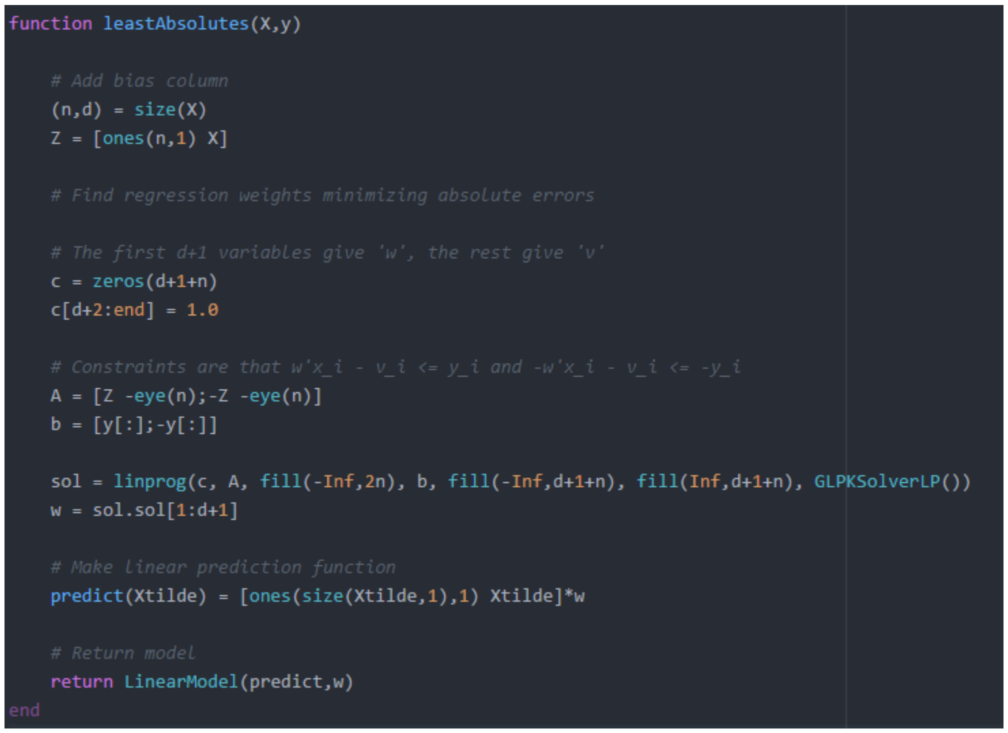
The script *example_outliers.jl* loads a one-dimensional regression dataset that has a non-trivial number of "outlier" data points. These points do not fit the general trend of the rest of the data, and pull the least squares model away from the main cluster of points. One way to improve the performance 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,

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

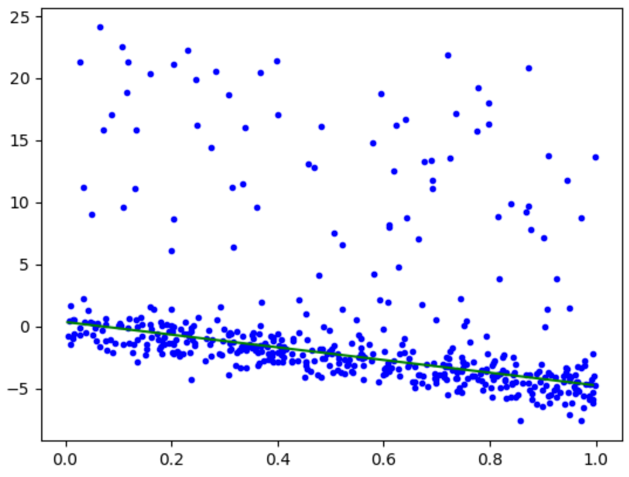
   You should turn this into a *linear program* as shown in class, and you can solve this linear program using the *linprog* function the *MathProgBase* package. Hand in the new function and the updated plot.

   Answer: The code could look like this:



```
function leastAbsolutes(X,y)

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

    # Find regression weights minimizing absolute errors

    # The first d+1 variables give 'w', the rest give 'v'
    c = zeros(d+1+n)
    c[d+2:end] = 1.0

    # Constraints are that w'x_i - v_i <= y_i and -w'x_i - v_i <= -y_i
    A = [Z -eye(n);-Z -eye(n)]
    b = [y[:];-y[:]]

    sol = linprog(c, A, fill(-Inf,2n), b, fill(-Inf,d+1+n), fill(Inf,d+1+n), GLPKSolverLP())
    w = sol.sol[1:d+1]

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

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

The plot should look like this:



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,

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

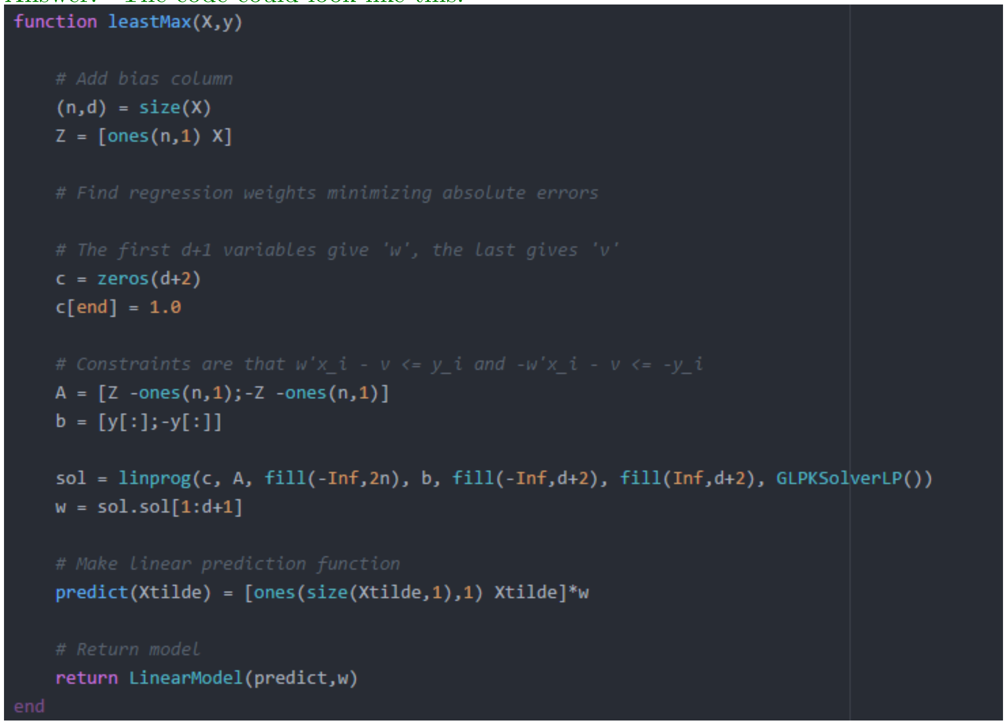
This objective function is non-smooth because of the absolute value function as well as the max function. Show how to formulate this non-smooth optimization problem as a linear program.

Answer: The linear program could look something like

$$\underset{w \in \mathbb{R}^d, v \in \mathbb{R}}{\arg\min}\ v, \quad \text{s.t.} \forall_i\ v \geq w^T x_i - y_i,\ v \geq y_i - w^T x_i.$$

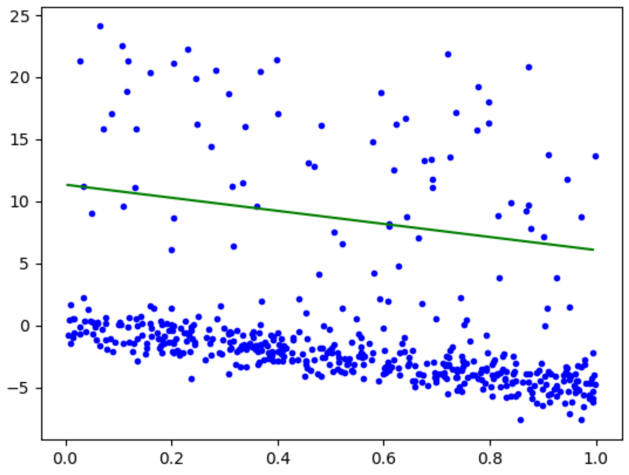
3. Write and hand in a function, *leastMax*, that fits this model using *linprog* (after adding a bias variable). Hand in the new function and the updated plot.

Answer: The code could look like this:



```
function leastMax(X,y)

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

    # Find regression weights minimizing absolute errors

    # The first d+1 variables give 'w', the last gives 'v'
    c = zeros(d+2)
    c[end] = 1.0

    # Constraints are that w'x_i - v <= y_i and -w'x_i - v <= -y_i
    A = [Z -ones(n,1);-Z -ones(n,1)]
    b = [y[:];-y[:]]

    sol = linprog(c, A, fill(-Inf,2n), b, fill(-Inf,d+2), fill(Inf,d+2), GLPKSolverLP())
    w = sol.sol[1:d+1]

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

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

The plot should look like this:



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 $c^T x$ 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 appropirate infinite values if there are no lower/upper bounds. The vectors $c/d/b/lb/ub$ should all be lists.