Basic Information

1. Name: Cody Griffith
2. Student ID: 88416169
3. Faculty: Applied Science
4. Department: Mathematics

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?
   - IID assumptions allow us to say that the training data is well representative of the testing data as well as cheaply estimate a conditional probability.
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?
   - You should never put together your training and test data to avoid bias, this would be a valid approach if you formed the k-means from only the training, evaluated its validation error on the test data and then continued with the supervised learning.
3. What is the difference between a validation set error and the test error?
   - The validation error is an approximation to the test error and should only be used once to get an unbiased estimate.
4. Describe a setting where using a validation set to choose hyper-parameters can lead to overfitting.
   - Polynomial fitting up to degree p and using the using the validation set to choose p can cause the polynomial to be too high of order and hence overfit the data.
5. What is the effect of the number of features d that our model uses on the training error and on the approximation error?
   - As we increase the number of features, we will see the training error decrease (possibly to zero) but our approximation error will likely increase due to overfitting.

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

   - This value is likely to be very small, if not zero. A regression model need not actually pass through any points on the validation set and hence using the Hamming distance is not a good measure of error.

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

   - If the matrix $X^T X$ has a large condition number, then the normal equation approach will be very unstable and more expensive in comparison to gradient descent.

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

   - As we allow λ to increase, we will see the sparsity increase (more weights sent to zero) but this will cause the training error to likely increase but perhaps also decrease the approximation error since we are preventing overfitting. As λ decreases, then the sparsity decreases (more weights are nonzero) and the opposite effect on the errors will likely occur.

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

   - That this problem is very expensive to the point where it may be impossible to even get a solution due to the non-convexity.

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?

    - The approximation error will increase as the inverse matrix becomes more unstable with a large condition number.

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 instead of the squared error?

    - The squared error returns massive values of error as $w^T x^i$ can be as large or small with regards to the threshold boundary, logistic loss is a smooth approximation to the 0-1 loss and hence the errors make more sense for this y^i.

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

    - A "one vs. all" logistic regression will return the label that had the largest positive value, where softmax loss will return the label with the highest probability of being correct.

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

    - For Laplace likelihood would translate to a L1-regression model, which is a situation that is less sensitive to outliers. The Laplace prior would lead to a L1- regularization which is especially good at feature selection.

14. What do we use the backpropagation algorithm for?

    - Backpropagation is used to calculate the gradient of our loss function and hence used to minimize our cost function.

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

    - When the problem is recursive in nature (same situation arises and a similar decision or method can be used) and when the problem can be broken into smaller sub-problems and solved independently.

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

    - This is parametric as we have a fixed number of hidden units, the parameters do not grow with more data.

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

    - CNN will force most weights to zero and hence reduce overfitting, but also forces neurons to be locally connected and hence share weights as well. Both of these cause the needed weights to be much smaller and hence overfitting is less likely.

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).

    - Begin by expanding the equation,

    $$f(w) = \frac{1}{2}\left[w^T\Sigma w - u^T\Sigma w - w^T\Sigma u + u^T\Sigma u\right].$$

    Which then has the following gradient with Σ symmetric,

    $$\nabla f(w) = \Sigma w - \Sigma u$$

    Setting this equal to zero to minimize gives $w = u$ as the minimizer.

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

   - Once more, expanding gives,

   $$\frac{1}{2\sigma^2}\left[w^T X^T X w - w^T X^T y - y^T X w - y^T y\right] + \frac{\lambda}{2} w^T w.$$

   Which would have the following gradient,

   $$\nabla f(w) = \frac{1}{\sigma^2}\left[X^T X w - X^T y\right] + \lambda w.$$

   Since we have that $\frac{1}{\sigma^2} X^T X + \lambda I$ is invertible (seen from a simple in-class proof of why $X^T X + \lambda I$ is invertible), we can find the minimizer,

   $$w = \left(\frac{1}{\sigma^2} X^T X + \lambda I\right)^{-1} \left(\frac{1}{\sigma^2} X^T y\right).$$

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

   - Taking the gradient,

   $$\begin{aligned}\nabla f(w) =& \sum_i v^i (w^T x^i - y^i) x^i + \frac{1}{2}\left(\Lambda + \Lambda^T\right)(w-u),\\ =& X^T V X w - X^T V y + \frac{1}{2}\left(\Lambda + \Lambda^T\right)(w-u).\end{aligned}$$

   Solving for the local optima, and using that Λ is positive semidefinite gives the invertibility criteria,

   $$w = \left(X^T V X + \Lambda\right)^{-1}\left(X^T V y + \Lambda u\right)$$

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.

2.2 Norm Inequaliies

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).

   - Here we will prove this for the square norms, but taking the root on both sides would lead to the actual norm. Notice for the square infinity norm, the maximum occurs in the vector. Call it w_k, then.

   $$\|w\|_\infty^2 = \max\{|w_i|\}^2 = |w_k|^2 \leq \sum_i |w_i|^2 = \|w\|_2^2.$$

For the final inequality, let $x_i = \frac{|w_i|}{\|w\|_1^2}$. Then we have that $x_i \leq 1$ for all i, and hence,

$$x_i^2 \leq x_i \Rightarrow \frac{\|w\|_2^2}{\|w\|_1^2} \leq 1 \Rightarrow \|w\|_2^2 \leq \|w\|_1^2.$$

Which finally gives,

$$\|w\|_\infty^2 \leq \|w\|_2^2 \leq \|w\|_1^2.$$

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

   - Once more, we will use the square norms. For the first inequality, recall Cauchy-Schwarz,

   $$\|w\|_1^2 = \left(\sum_i |w_i| \cdot 1\right)^2 \leq \left(\sum_i |w_i|^2\right) \cdot \left(\sum_i 1^2\right) = d\|w\|_2^2.$$

   For the next inequality, we just notice that a sum will be bound by a sum of its max,

   $$\|w\|_2^2 = \sum_i |w_i|^2 \leq \sum_i \max\{|w_i|\}^2 = d\|w\|_\infty^2 \Rightarrow d\|w\|_2^2 \leq d^2\|w\|_\infty^2.$$

   Putting this together,

   $$\|w\|_1^2 \leq d\|w\|_2^2 \leq d^2\|w\|_\infty^2.$$

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

   - Note that $2ab \leq a^2 + b^2$ for all $a, b \in \mathbb{R}$ which follows directly from $(a-b)^2 \geq 0$. With this,

   $$\begin{aligned}\frac{1}{2}\|w+u\|_2^2 =& \frac{1}{2}\|w\|_2^2 + \|w\|_2\|u\|_2 + \frac{1}{2}\|w\|_2^2, \\ \leq& \|w\|_2^2 + \|u\|_2^2.\end{aligned}$$

You should use the definitions of the norms, but should not use the known equivalences between these norms (since these are the things you are trying to prove). Hint: for many of these it's easier if you work with squared values (and you may need to "complete the square"). Beyond non-negativity of norms, it may also help to use the Cauchy-Schwartz inequality 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 seperate variance for each training example and Laplace prior,

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

   - With a Gaussian Likelihood and Laplace prior, we follow the form of MLE and MAP estimators,

$$f(w) = \sum_i -\ln(P(y_i|x_i, w) + \sum_j -\ln(P(w_j)).$$

     Define $K_1^i = -\frac{1}{2}\ln(2\pi\sigma_i)$ and $K_2 = \ln(\lambda/2)$, then from the above equation we have,

$$f(w) = \sum_i -K_1^i + \frac{1}{2\sigma_i^2}(w^T x^i - y^i)^2 + \sum_j -K_2 + \lambda|w_j|.$$

     To clean this up a bit, we just combine the all constants and allow for $\Sigma = (1/\sigma_i)I \in \mathbb{R}^{n\times n}$ to be a diagonal matrix,

$$f(w) = K + \frac{1}{2}\left\|\Sigma X w - \Sigma y\right\|_2^2 + \lambda\|w\|_1.$$

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

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

   - Using the same equation as above, let $K_1 = -\ln(\sqrt{\nu}B\left(\frac{1}{2}, \frac{\nu}{2}\right))$ and $K_2 = \frac{1}{2}\ln(\lambda/2\pi)$ then we have,

$$f(w) = \sum_i -K_1 + \frac{\nu+1}{2}\ln\left(1 + \frac{(w^T x^i - y^i)^2}{\nu}\right) + \sum_j -K_2 + \frac{\lambda}{2}(w_j - u_j)^2.$$

Once again, mostly combining all constants and simplifying a bit,

$$f(w) = K + \frac{\nu+1}{2}\sum_i \ln\left(1 + \frac{(w^T x^i - y^i)^2}{\nu}\right) + \frac{\lambda}{2}\|w-u\|_2^2.$$

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 has quit the study while $v^i = 0$ if they are still in it.

- One more time, let $K_j = \frac{1}{2}\ln(\lambda_j/2\pi)$, then,

$$f(w) = \sum_i -v^i w^T x^i + y^i e^{w^T x^i} + \sum_j -K_j + \frac{\lambda_j}{2} w_j^2.$$

Simplifying with $\Lambda \in \mathbb{R}^{n\times n}$ the diagonal matrix containing $\sqrt{\lambda_j}$ and $V \in \{0,1\}^{d\times d}$ the diagonal matrix containing v_i,

$$f(w) = K + \|VXw - e^{Xw} y\|_1 + \frac{1}{2}\|\Lambda w\|_2^2.$$

Note: I choose to put the square root inside the matrix to avoid confusion, although $\Lambda^{1/2}$ would make sense for a diagonal.

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,

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

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

- Taking the gradient,

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

Appling the gradient again to get the Hessian,

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

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

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

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

- Define $V = v_i I$ where this is a diagonal matrix with v_i along the diagonal, then the gradient is,

$$\nabla f(w) = \sum_i v_i(w^T x^i - y^i)x^i + \Lambda w = X^T V X w - X^T V y + \Lambda w.$$

The Hessian follows easily as $X^T V X$ is symmetric,

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

3. Weighted L2-regularized probit regression,

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

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

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

Here Φ is the CDF of the standard normal distribution.

- Denote $c_i = \Phi(y^i w^T x^i$ and $p_i = \phi(y^i x^T x^i)$ as the standard normal cdf and pdf respectively, we can look at the w_kth partial of our loss function,

$$f_{w_k}(w) = -\frac{p_i}{c_i} y^i x_k^i + u_k w_k$$

Define the diagonal matrix U with the entires u_j, the gradient is,

$$\nabla f(w) = -\sum_i y^i \frac{p_i}{c_i} x^i + Uw.$$

Following the same approach, noticing that y^i squared is 1 and to any odd power is itself,

$$\nabla^2 f(w) = \sum_i \left(y^i \frac{p_i}{c_i} w^T x^i + \frac{p_i^2}{c_i^2} \right) x^i (x^i)^T + U$$

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 Cross-Validation

Download *a1.zip* from the course webpage and start Julia (V0.6) 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:

```
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 λ=1 and σ=1.

- *rbfBasis.jl*:

```
include("misc.jl")

function rbfBasis(X,Y,sigma)
    # Take matrix and create Gaussian Distance matrix
    (n,d) = size(X)
    distances = distancesSquared(X,Y)
    Gauss = exp.(-distances/(2*sigma^2))
    return Gauss
end
```

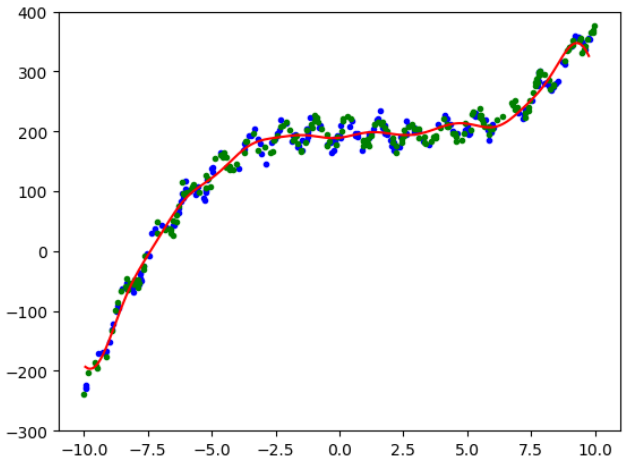
leastSquaresRBFL2.jl:

```
include("misc.jl")
include("rbfBasis.jl")

function leastSquaresRBFL2(X,y,lambda,sigma)

# Shift basis, add bias and linear columns
(n,d) = size(X)
Z = rbfBasis(X,X,sigma)
    Z = [ones(n,1) Z]

# Find regression weights minimizing squared error
w = (Z'*Z+lambda*eye(n+1))\(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
```

Plot code:

```
include("leastSquaresRBFL2.jl")
# Load X and y variable
lambda=1; sigma=1;
using JLD
data = load("nonLinear.jld")
(X,y,Xtest,ytest) = (data["X"],data["y"],data["Xtest"],data["ytest"])

# Compute number of training examples and number of features
(n,d) = size(X)

# Fit least squares model
model = leastSquaresRBFL2(X,y,lambda,sigma)
```

```
# Report the error on the test set
t = size(Xtest,1)
yhat = model.predict(Xtest)
testError = sum((yhat - ytest).^2)/t
@printf("TestError = %.2f\n",testError)

# Plot model
using PyPlot
figure()
plot(X,y,"b.")
plot(Xtest,ytest,"g.")
Xhat = minimum(X):.1:maximum(X)
Xhat = reshape(Xhat,length(Xhat),1) # Make into an n by 1 matrix
yhat = model.predict(Xhat)
plot(Xhat,yhat,"r")
ylim((-300,400))
```



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 $O(d^3)$.

    - Linear regression: $X^T y \sim O(nd)$, $X^T X \sim O(nd^2)$, solving $X^T X w = X^T y \sim O(d^3)$ and hence the total complexity of this problem is $O(nd^2 + d^3)$.

Gaussian RBF: $Z \sim O(n^2 d)$, $Z^T Z \sim O(nd)$, $Z^T y \sim O(n^2)$, $Z^T Z v = Z^T y \sim O(n^3)$. Thus the entire problem is $O(n^2 d + n^3)$. For classifying t more examples, Linear: $\tilde{X} w \sim O(td)$. Gaussian RBF: $\tilde{Z} \sim O(tnd)$ and $\tilde{Z} v \sim O(td)$, hence our total cost is $O(tnd)$.

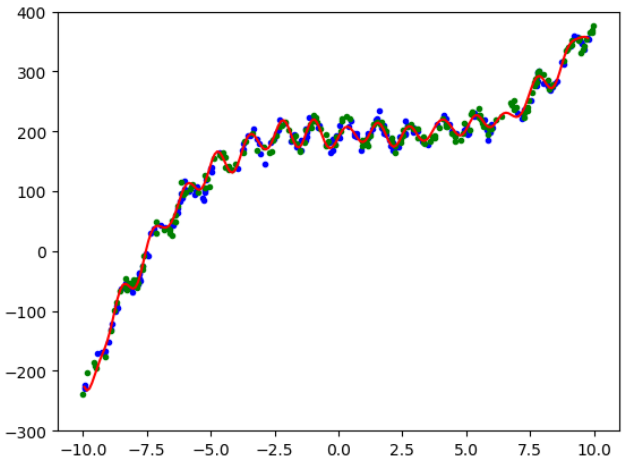
3. Modify the script to split the training data into a "train" and "validation" set (you can use half the examples for training and half for validation), and use these to select λ and σ. Hand in your modified script and the plot you obtain with the best values of λ and σ.

   - Modified code: Will choose from a range of powers of 2, here $\lambda = 0.000031$ and σ=0.5.

```
include("leastSquaresRBFL2.jl")
# Load X and y variable
using JLD
data = load("nonLinear.jld")
(X,y,Xtest,ytest) = (data["X"],data["y"],data["Xtest"],data["ytest"])

# Compute number of training examples and number of features
(n,d) = size(X)

# Split into training and validation sets
floorN = floor(Int8,n/2)
Xtrain = X[1:floorN,:]
ytrain = y[1:floorN]
Xval = X[floorN+1:n,:]
yval = y[floorN+1:n]

(n,d) = size(Xval)

# Generate a range of values
range1 = -4:2
sigmavec = 2.0.^range
lambdavec = 2.0.^range
m = length(range)

valErrorMatrix = zeros(m,m)

# Create a model for each pair
for i = 1:m
    for j = 1:m
        lambda = lambdavec[i]
        sigma = sigmavec[j]

        # Fit least squares model
        model = leastSquaresRBFL2(Xtrain,ytrain,lambda,sigma)
```

```
        # Find lowest validation error
        t = size(Xval,1)
        yhat = model.predict(Xval)
        valErrorMatrix[i,j] = sum((yhat - yval).^2)/n
    end
end

# Find smallest validation error and choose that model
s = indmin(valErrorMatrix)
divisor = floor(Int8,s/m)
remainder = rem(s,m)
if remainder == 0
    sigma = sigmavec[divisor]
    lambda = lambdavec[m]
else
    sigma = sigmavec[divisor+1]
    lambda = lambdavec[remainder]
end
# This will choose lambda=0.0625 and sigma=0.5.

model = leastSquaresRBFL2(Xtrain,ytrain,lambda,sigma)

# Report the error on the test set
t = size(Xtest,1)
yhat = model.predict(Xtest)
testError = sum((yhat - ytest).^2)/t
@printf("TestError = %.2f\n",testError)

# Plot model
using PyPlot
figure()
plot(X,y,"b.")
plot(Xtest,ytest,"g.")
Xhat = minimum(X):.1:maximum(X)
Xhat = reshape(Xhat,length(Xhat),1) # Make into an n by 1 matrix
yhat = model.predict(Xhat)
plot(Xhat,yhat,"r")
ylim((-300,400))
```

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.



- For (i) we could allow for an additional linear term to drive the direction of the regression and take less effect from the oscillations. For (ii) we could allow the variance to depend on the the data point, σ_i^2. This could allow us to choose where we want to allocate more weight in the model.

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,

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

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.

- *softmaxClassifier.jl*:

```
include("findMin.jl")
function softmaxReg(X,y)
    # Set up dimensions
    (n,d) = size(X)
    k = maximum(y)
    W = zeros(d,k)

    funObj(W) = softmaxObj(W[:],X,y)
    W[:] = findMin(funObj,W[:],verbose=false,derivativeCheck=true)
    W=reshape(W,d,k)

# Make prediction from maximum over each row
predict(Xhat) = maxRow(Xhat*W)


# Return model
return LinearModel(predict,W)
end

function maxRow(X)
    n= size(X,1)
    labels = []
    for i= 1:n
        (maxprob,nextlabel) = findmax((X)[i,:])
        labels = append!(labels, nextlabel)
    end
    return labels
end


function softmaxObj(W,X,y)
    # Parameter input can be a vector
    (n,d) = size(X)
    k = maximum(y)
    W = reshape(W,d,k)

    # Compute the function value
    f = 0
    for i = 1:n
        f = f-W[:,y[i]]'*X[i,:]
    end
```

```
    f = f+sum(log.(sum(exp.(X*W),2)))

    # Gradient
    g=zeros(d,k)
    for j = 1:k
        for i = 1:n
            if j == y[i]
                g[:,j] = g[:,j]-X[i,:]
            end
            g[:,j] = g[:,j]+X[i,:]*exp(W[:,j]'*X[i,:])./(sum(exp.(X*W),2)[i])
        end
    end
    g=g[:]
    return (f,g)
end
```

Validation error = 0.026

3.3 Robust Regression

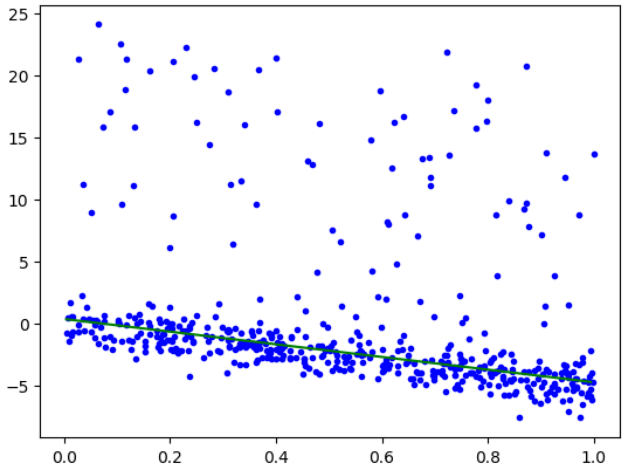
The script example *outliers.jl* loads a one-dimensional regression dataset that has a non-trivial number of "outlier" data points. These points do not fit the general trend of the rest of the data, and pull the least squares model away from the main cluster of points. One way to improve the performance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

   - *leastAbsolutes.jl*:

```
function leastAbsolutes(X,y)
    (n,d) = size(X)
    # Add a bias
    Z = [ones(n,1) X]
    (n,d) = size(Z)

    # Find regression weights using linprog
    c = [zeros(d,1);ones(n,1)]
```

```
    c=vec(c)
    A = [ -Z -eye(n) ; Z -eye(n)]
    b = [-y;y]
    uppervec = [-y;y]
    uppervec = vec(uppervec)
    lowervec = -Inf*ones(2*n,1)
    lowervec = vec(lowervec)
    lowerbound = -Inf*ones(n+d,1)
    lowerbound = vec(lowerbound)
    upperbound = Inf*ones(n+d,1)
    upperbound = vec(upperbound)

    solution = linprog(c,A,lowervec,uppervec,lowerbound,upperbound,
    GLPKSolverLP())

    w = solution.sol[1:d]

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

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



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.

- The objective here is

$$\arg\min_w \|Xw - y\|_1 = \arg\min_w \max_i \{|w^T x^i - y^i|\}.$$

Where we take the interior and break this into the constraints for the problem if we notice,

$$r = \max_i |w^T x^i - y^i| = \max_i \{w^T x^i - y^i, y^i - w^T x^i\}$$

$$\begin{aligned} &\arg\min_{w,r} r, \\ &\quad r \geq w^T x^i - y^i, \\ &\quad r \geq y^i - w^T x^i. \end{aligned}$$

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.

- *leastMax.jl*:

```
function leastMax(X,y)
    (n,d) = size(X)
    # Add a bias
    Z = [ones(n,1) X]
    (n,d) = size(Z)

    # Find regression weights using linprog
    c = [zeros(d,1); 1]
    c=vec(c)
    A = [ -Z -ones(n,1) ; Z -ones(n,1)]
    uppervec = [-y;y]
    uppervec = vec(uppervec)
    lowervec = -Inf*ones(2*n,1)
```

```
    lowervec = vec(lowervec)
    lowerbound = -Inf*ones(1+d,1)
    lowerbound = vec(lowerbound)
    upperbound = Inf*ones(1+d,1)
    upperbound = vec(upperbound)

    solution = linprog(c,A,lowervec,uppervec,lowerbound,upperbound,
     GLPKSolverLP())
    w = solution.sol[1:d]

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

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