CPSC 340 Assignment 4 (due Friday November 1 at 11:55pm)

1 Gaussian RBFs and Regularization

Unfortunately, in practice we often don't know what basis to use. However, if we have enough data then we can make up for this by using a basis that is flexible enough to model any reasonable function. These may perform poorly if we don't have much data, but can perform almost as well as the optimal basis as the size of the dataset grows. In this question you will explore using Gaussian radial basis functions (RBFs), which have this property. These RBFs depend on a parameter σ, which (like p in the polynomial basis) can be chosen using a validation set. In this question, you will also see how cross-validation allows you to tune parameters of the model on a larger dataset than a strict training/validation split would allow.

1.1 Regularization

If you run the demo *example_RBF.jl*, it will load a dataset and randomly split the training examples into a "train" and a "validation" set (it does this randomly since the data is sorted). It will then search for the best value of σ for the RBF basis. Once it has the "best" value of σ, it re-trains on the entire dataset and reports the training error on the full training set as well as the error on the test set.

A strange behaviour appears: if you run the script more than once it might choose different values of σ. Sometimes it chooses a large value of σ (like 32) that follows the general trend but misses the oscillations. Other times it sets $\sigma = 1$ or $\sigma = 2$, which fits the oscillations better but overfits so achieves a similar test error.[1] Modify the *leastSquaresRBF* function so that it allows a regularization parameter λ and it fits the model with L2-regularization. Hand in your code, and report and describe how the performance changes if you use a regularized estimate with $\lambda = 10^{-12}$ (a very small value). Hint: to construct an identity matrix in Julia, use the linear algebra package (*using LinearAlgebra*) and then use I to make an identity matrix of the appropriate size (Julia figures out the dimensions for you).

```
function leastSquaresRBFWithL2(X,y,sigma,lambda)
    (n,d) = size(X)

    Z = rbf(X,X,sigma)

    w = (Z'*Z + lambda * I)\(Z'*y)

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

    return LinearModel(predict,w)
end
```

Answer:
Running the script multiple times, the optimal σ will be less likely to change from 1.

[1] This behaviour seems to be dependent on your exact setup. Because the $Z^T Z$ matrix with the RBF matrix is really-badly behaved numerically, different floating-point and matrix-operation implementations will handle this in different ways: in some settings it will actually regularizer for you!

1.2 Cross-Validation

Even with regularization, the randomization of the training/validation sets has an effect on the value of σ that we choose (on some runs it still chooses a large σ value). This variability would be reduced if we had a larger "train" and "validation" set, and one way to simulate this is with *cross-validation*. Modify the training/validation procedure to use 10-fold cross-validation to select σ, and hand in your code. How does this change the performance when fixing $\lambda = 10^{-12}$?[2]

```
Xtrain = [];
ytrain = [];
Xvalid =  [];
yvalid =  [];
for i in 1:10
    validStart = (i-1)*Int64(n/10) + 1 # Start of validation indices
    validEnd = i * Int64(n/10) # End of validation incides
    validNdx = perm[validStart:validEnd] # Indices of validation examples
    trainNdx = perm[setdiff(1:n,validStart:validEnd)] # Indices of training examples
    push!(Xtrain,X[trainNdx,:])
    push!(ytrain , y[trainNdx])
    push!(Xvalid , X[validNdx,:])
    push!(yvalid , y[validNdx])
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 = []
for sigma in 2.0.^(-15:15)
    errors = zeros(10)
    for i in 1:10
        # Train on the training set
        model = leastSquaresRBFWithL2(Xtrain[i],ytrain[i],sigma, 1e-12)

        # Compute the error on the validation set
        yhat = model.predict(Xvalid[i])
        validError = sum((yhat - yvalid[i]).^2)/(n/2)
        #@printf("With sigma = %.3f, validError = %.2f\n",sigma,validError)
        errors[i] = validError
    end
    validError = mean(errors)
    # Keep track of the lowest validation error
    if validError < minErr
        global minErr = validError
        global bestSigma = sigma
    end
end
```

Answer: The algurithm will now always choose 1 as σ after run the code several times.

1.3 Cost of Non-Parametric Bases

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.

1. What is the cost in big-O notation of training a linear regression model with Gaussian RBFs on n training examples with d features (for fixed σ and λ)?

   Answer: $O(n^2d)$

2. What is the cost of classifying t new examples with this model?

   Answer: $O(ndt)$

3. When is it cheaper to train using Gaussian RBFs than using the original linear basis?

   Answer: When feature is nore than number of examples.

4. When is it cheaper to predict using Gaussian RBFs than using the original linear basis?

   Answer: Never

[2] In practice, we typically use cross-validation to choose both σ and λ

2 Logistic Regression with Sparse Regularization

If you run the function *example_logistic.jl*, it will:

1. Load a binary classification dataset containing a training and a validation set.
2. "Standardize" the columns of X and add a bias variable.
3. Apply the same transformation to $Xvalidate$.
4. Fit a least squares model, using the sign of $w^T x_i$ to make predictions.
5. Report the number of features selected by the model (number of non-zero regression weights).
6. Report the error on the training and validation sets.

Least squares does ok as a binary classifier on this dataset, but it uses all the features (even though only the prime-numbered features are relevant) and the validation error is above the minimum achievable for this model (which is 1 percent, if you have enough data and know which features are relevant). In this question, you will modify this demo to use the logistic loss and to use different forms of regularization to improve on these aspects.

2.1 Logistic Regression

Instead of least squares, modify the script to use logistic regression. You can use the *logReg.jl* file, which implements the training and prediction function for a logistic regresion classifier (using a version of the *findMin* function that does derivative checking for you and that uses more-clever choices of step-sizes). When you switch to using logistic regression, report how the following quantities change: the training error, validation error, and number of features.

Answer: The training error goes to 0; and validation error is 0.082 lower than before. but number of none zeros remains unchanged.

2.2 L2-Regularization

Make a new function, *logRegL2*, that takes an input parameter λ and fits a logistic regression model with L2-regularization. Specifically, while *logReg* computes w by minimizing

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

your new function *logRegL2* should compute w by minimizing

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \frac{\lambda}{2}\|w\|^2.$$

Hand in the objective function that your updated code minimizes, and using $\lambda = 1.0$ report how the following quantities change: the training error, the validation error, the number of features used, and the number of gradient descent iterations.

```
function logisticObj(w,X,y, lambda)
    yXw = y.*(X*w)
    f = sum(log.(1 .+ exp.(-yXw)))+ sum(lambda/2 * (w'* w))
    g = -X'*(y./(1 .+ exp.(yXw))) + lambda * w
    return (f,g)
end
```

Answer: numberOfNonZero = 101 trainError = 0.002 validError = 0.074 The training Error goes up but the validation error goes down, and Nomber of non zero stays the same.

2.3 L1-Regularization

Make a new function, *logRegL1*, that takes an input parameter λ and fits a logistic regression model with L1-regularization,

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \lambda \|w\|_1.$$

Hand in your *logRegL1* code. Using this new code and $\lambda = 1$, report the following quantities: the training error, the validation error, and the number of features the model uses.

You should use the function *findMinL1*, which implements a proximal-gradient method to minimize the sum of a differentiable function g and $\lambda\|w\|_1$,

$$f(w) = g(w) + \lambda \|w\|_1.$$

This function has a similar interface to *findMin*, except that you (a) only provide the code to compute the function/gradient of the differentiable part g and (b) need to provide the value λ.

```
function logReg_L1(X,y, lambda)

    (n,d) = size(X)

    # Initial guess
    w = zeros(d,1)

    # Function we're going to minimize (and that computes gradient)
    funObj(w) = logisticObj(w,X,y,lambda)

    # Solve least squares problem
    w = findMinL1(funObj,w, lambda)

    # Make linear prediction function
    predict(Xhat) = sign.(Xhat*w)

    # Return model
    return LinearModel(predict,w)
end

function logisticObj(w,X,y, lambda)
    yXw = y.*(X*w)
    f = sum(log.(1 .+ exp.(-yXw)))+ sum(lambda * w)
    g = -X'*(y./(1 .+ exp.(yXw))) + lambda * ones(size(w))
    return (f,g)
end
```

Answer: numberOfNonZero = 72 trainError = 0.0 validError = 0.048 The number of feature used is smaller. The test error and validation error also goes down.

2.4 L0-Regularization

The function *logRegL0* contains part of the code needed to implement the *forward selection* algorithm, which approximates the solution with L0-regularization,

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \lambda \|w\|_0.$$

The 'for' loop in this function is missing the part where we fit the model using the subset *Sj*, then compute the score and updates the *minScore/minS*. Modify the 'for' loop in this code so that it fits the model using only the features *Sj*, computes the score above using these features, and updates the *minScore/minS* variables

(if you want to turn off the diagonistics generated by *findMin*, you can use *verbose = false*).[3] Hand in your updated code. Using this new code, set $\lambda = 1$ and report: the training error, the validation error, and the number of features used.

Note that the code differs a bit from what we discussed in class, since we assume that the first feature is the bias variable and assume that the bias variable is always included. Also, note that for this particular case using the L0-norm with $\lambda = 1$ is equivalent to what is known as the Akaike information criterion (BIC) for variable selection.

```
for j in setdiff(1:d,S)
    # Fit the model with 'j' added to the feature set 'S'
    # then compute the score and update 'minScore' and 'minS'
    Sj = [S;j]
    Xs = X[:,Sj]

    # PUT YOUR CODE HERE
    # Start out just using the bias variable (assumed to be in first column),
    # and record 'score' which is the loss plus regularizer
    w = zeros(length(Sj),1)
    w = findMin(funObj,w,verbose=false)
    (f,~) = funObj(w)
    score = f + lambda*length(Sj)
    #@show score
    if minScore > score
        minScore = score # Lowest score we've found
        minS = Sj # Best set of features we've found
    end
end
```

Answer: numberOfNonZero = 24 trainError = 0.0 validError = 0.018 The validation error goes down a lot, and number of features used is 24. They are all relevent. (1 2 3 5 7 11 13 17 19 23 31 37 41 43 47 53 59 67 71 73 79 83 89 101)

3 Multi-Class Logistic

The function *example_multiClass* loads a multi-class classification datasetwith $y_i \in \{1, 2, 3, 4, 5\}$ and fits a 'one-vs-all' classification model using binary logistic regression, 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 predicts that examples will be in class 1 (the green class).

3.1 Softmax Classification

Linear classifiers make their decisions by finding the class label c maximizing the quantity $w_c^T x_i$, so we want to train the model to make $w_{y_i}^T x_i$ larger than $w_{c'}^T x_i$ for all the classes c' that are not the true label y_i. Here, c is a possible label and $w_{c'}$ is row c' of W. Similarly, y_i is the training label, w_{y_i} is row y_i of W, and in this setting we are assuming a discrete label $y_i \in \{1, 2, \ldots, k\}$. Before we move on to implementing the softmax classifier to fix the issues raised in the introduction, let's do a simple example:

Consider the dataset below, which has 10 training examples, 2 features, and 3 class labels:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 3 \\ 3 \\ 3 \\ 3 \end{bmatrix}.$$

[3] Note that Julia doesn't like when you re-define functions, but if you change the variable *Xs* it will actually change the behaviour of the *funObj* that is already defined.

Suppose that you want to classify the following test example:

$$\hat{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Suppose we fit a multi-class linear classifier using the softmax loss, and we obtain the following weight matrix:

$$W = \begin{bmatrix} +2 & -1 \\ +2 & +2 \\ +3 & -1 \end{bmatrix}$$

1. Why are the weights of this model a 3×2 matrix?

   Answer: Because there are 3 kinds of labels in the training set. And there are 2 freatures for one example.

2. Under this model, what class label would we assign to the test example? (Show your work.)

   Answer: $Wx = \begin{bmatrix} +2+-1 \\ +2++2 \\ +3+-1 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix}$
   So it will presict 2 as it has the largest value.

3.2 Softmax Loss

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 loss probability,

$$p(y_i \mid W, x_i) = \frac{\exp(w_{y_i}^T x_i)}{\sum_{c=1}^{k} \exp(w_c^T x_i)}.$$

The loss function corresponding to the negative logarithm of the softmax probability for n training examples is given by

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

Derive the partial derivative of this loss function with respect to a particular element W_{cj}. Try to simplify the derivative as much as possible (but you can express the result in summation notation).

Answer: The diravitive respect to w_{cj} is $\sum_{i=1}^{n}[-X_{ij} * I(y_1 = c) + \frac{1}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)} * (exp(w_c^T x_i)) * x_{ij}]$

Hint: for the gradient you can use x_{ij} to refer to element j of example i. For the first term you can use an 'indicator' function, $I(y_i = c)$, which is 1 when $y_i = c$ and is 0 otherwise. Note that you can use the definition of the softmax probability to simplify the derivative.

3.3 Softmax Classifier

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 will want to use the *derivativeCheck* option in *findMin.jl* to check that your gradient code is correct. Also, note that *findMin.jl* expects that the parameter vector and gradient are *column vectors*. The easiest way to work around these issues is to use the *reshape* command: call *findMin.jl* with a $dk \times 1$ vector w and at the start of your objective function reshape w to be a $d \times k$ matrix W, then compute the $d \times k$ matrix of partial derivatives and finally reshape this to be the $dk \times 1$ gradient vector.

```
function softMaxClassifier(X,y)
    (n,d) = size(X)
    k = maximum(y)
    W = zeros(k*d,1)
    funObj(w) = softMaxObjObj(w,X,y)
    W = findMin(funObj,W,verbose=false, derivativeCheck=true)I
    function predict(Xhat)
        w2 = reshape(W,k,d)
        return mapslices(argmax,Xhat*w2',dims=2)
    end
    return LinearModel(predict,W)
end

function softMaxObjObj(w,X,y)
    (n,d) = size(X)
    k = maximum(y)
    t = reshape(w,k,d)
    f = 0
    for i in 1:n
        f = f - sum(t[y[i],:].*X[i,:])
        r = 0
        for c in 1:k
            r = r + exp(sum(t[c,:].*X[i,:]))
        end
        f = f + log(r)
    end
    g = zeros(k,d)
    for c in 1:k
        for j in 1:d
            for i in 1:n
                if y[i] == c
                    g[c,j] = g[c,j] - X[i,j]
                end
                t1 = 0
                for c1 in 1:k
                    t1 = t1 + exp(sum(t[c1,:].*X[i,:]))
                end
                t2 = 0
                for c2 in 1:k
                    if c2 == c
                        t2 = t2 + exp(sum(t[c2,:].*X[i,:])) * X[i,j]
                    end
                end
                g[c,j] = g[c,j] +1/t1 *t2
            end
        end
    end
    rg = reshape(g,k*d,1)
    return (f,rg)
end
```

Answer:
validError = 0.026

3.4 Cost of Multinomial Logistic Regression

Assuming that we have

- n training examples.
- d features.
- k classes.
- t testing examples.
- T iterations of gradient descent for training.

1. In $O()$ notation, what is the cost of training the softmax classifier?

4 Very-Short Answer Questions

1. Consider performing feature selection by measuring the "mutual information" between each column of X and the target label y, and selecting the features whose mutual information is above a certain threshold (meaning that the features provides a sufficient number of "bits" that help in predicting the label values). Without delving into any details about mutual information, what is a potential problem with this approach?

   Answer: This method might make errors to select features having a high correlation but not necessary relevent.

2. Why do we use forward selection instead of exhaustively search all subsets in search and score methods?

   Answer: Because the total number of subset are huge it is 2^d possibilities.

3. What is a setting where you would use the L1-loss, and what is a setting where you would use L1-regularization?

   Answer: We would use L1 loss when we have outliers. We would use L1 regularization if we want to select important feateure from the regression(sparse solusion).

4. Among L0-regularization, L1-regularization, and L2-regularization: which yield convex objectives? Which yield unique solutions? Which yield sparse solutions?

   Answer: L2 and L1 will have convex objectives. L2 will have unique solution. L0 and L1 will have sparse solution.

5. What is the effect of λ in L1-regularization on the sparsity level of the solution? What is the effect of λ on the two parts of the fundamental trade-off?

   Answer: The bigger the lambda is, the more sparse the value will be. The bigger the lambda is, the training Error will go up and approximation error will go down as the model will be less sensitive to training data and training error will be more accurate approximating the test error.

6. Suppose you have a feature selection method that tends not generate false positives but has many false negatives (it misses relevant variables). Describe an ensemble method for feature selection that could improve the performance of this method.

7. How does the hyper-parameter σ affect the shape of the Gaussian RBFs bumps? How does it affect the fundamental tradeoff?

   Answer: The σ affects the wideness of the normal distribution, the narrower the distribution, the training error will go down but the approximation error will go up, as it is more sensitive to trainging data.

8. What is the main problem with using least squares to fit a linear model for binary classification?

   Answer: We have to make sure that the two groups are be able to split by a hyper plane.

9. Suppose a binary classification dataset has 3 features. If this dataset is "linearly separable", what does this precisely mean in three-dimensional space?

   Answer: There is a four dimentional hyper plane that can seperate two class.

10. When searching for a good w for a linear classifier, why do we use the logistic loss instead of just minimizing the number of classification errors?

    Answer: Because the function will be non-convex in w.

11. For a linearly-separable binary classification problem, how does an SVM classifier differ from a classifier found using the perceptron algorithm?

    Answer: The perceptron algorithm and SVM are gareenteed to find a perfect classifier if they exist, but SVM will make sure it also have a big margin.

12. Which of the following methods produce linear classifiers? (a) binary least squares as in Question 3, (b) the perceptron algorithm, (c) SVMs, (d) logistic regression, and (e) KNN with $k = 1$ and $n = 2$.

    Answer: a, b, c,d

13. Why do we use the polynomial kernel to implement the polynomial basis when d and p (degree of polynomial) are large?

    Answer: Because it is fast, we don't have to calculate Z for all the examples, we can get k directly from x.

14. Suppose we want to fit a multi-class logistic regression model, but the class labels do not form convex regions. How could we modify multi-class logistic regression to fit non-convex regions?

    Answer: We can make some basis transformation to make the regions convex.