CPSC340A3

Qinglan Huang n6v9a Liu Yang v4d9

November 6, 2017

1 Convex Functions

1.$\frac{d}{dx}f'(w) = 2\alpha - \beta \geq 0$
Since the we can use second derivative and make the second derivative always greater or equal to zero for this entire domain therefore the function is always convex.

2.$\frac{d}{dx}f'(w) = \frac{1}{w} \geq 0$
Computing the second derivative for one-dimension function, the second derivative always greater or equal to zero for entire domain therefore the function is always convex.

3.$f(w) = \|Xw - y\|^2 + \lambda\|w\|$
$Since \|Xw - y\|^2$ is convex function since it is norm, and $\|w\|$ is convex since L1-regularized norm is tend to set variable to 0, sum of two convex function is convex.

4.Since $-y_i w^T x_i$ is linear function and scalar. Log(1+ $e^{-y_i w^T x_i}$) can be written as log(1+e^{n_i}), and second derivative of linear function is always greater or equal to zero thus n_i is convex.Second derivative of log(1+ e^{n_i}) is convex, since $\frac{d}{dx}f'(w) = \frac{e^{n_i}}{(1+e^{n_i})^2}$ is always greater or equal to zero thus it is convex. Therefore the sum of the convex function is convex function as well.

math is here:

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

Let $g(x) = \log(1+\exp(x))$

$$g'(x) = \frac{(1+\exp(x))'}{1+\exp(x)} = \frac{\exp(x)}{1+\exp(x)}$$

$$g''(x) = \frac{\exp(x)'(1+\exp(x)) - \exp(x)(1+\exp(x))'}{(1+\exp(x))^2}$$

$$= \frac{\exp(x)(1+\exp(x)) - \exp(x)\exp(x)}{(1+\exp(x))^2}$$

$$= \frac{\exp(x)}{(\exp(x)+1)^2} \quad \text{for all } x.$$

5.Since $w_0 - w^T x_i$ is linear function and convex because of the second derivative is greater or equal to 0, then the max of two convex function is convex and sum of the convex is convex. And $\|w\|^2$ is convex, thus the f(w) is convex which is sum of two convex function.

2 Gaussian RBFs and Regularization

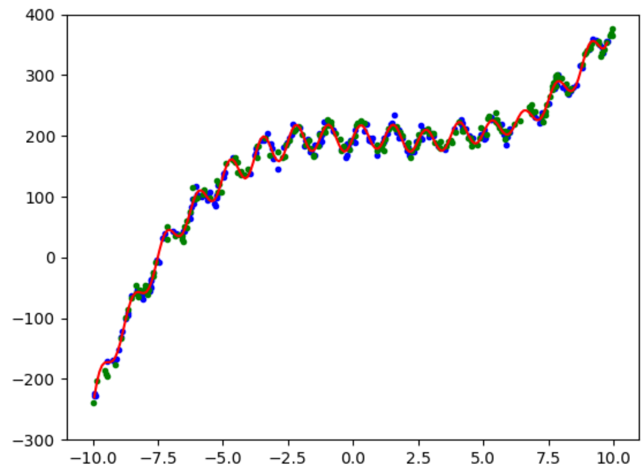
2.1 Regularization

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

    Z = rbf(X,X,sigma)


    v=ones(n)
    v=Diagonal(v)

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

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

    return LinearModel(predict,w)
end

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

2.2 Cross-Validation

```julia
using JLD
data = load("basisData.jld")
(X,y,Xtest,ytest) = (data["X"],data["y"],data["Xtest"],data["ytest"])

# Data is sorted, so *randomly* split into train and validation:
n = size(X,1)
perm = randperm(n)
#validStart = Int64(n/2+1) # Start of validation indices
#validEnd = Int64(n) # End of validation incides
#validNdx = perm[validStart:validEnd] # Indices of validation examples
#trainNdx = perm[setdiff(1:n,validNdx)] # Indices of training examples

stop = floor(Int,(n / 10))
println(stop)

validIndex = Matrix{Int64}(10, stop)
trainIndex = Matrix{Int64}(10,n-stop)

# 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 i in 1: 10
    validEnd = stop * i
    validStart = validEnd - stop + 1
    validIndex[i,:] = perm[validStart:validEnd]
    trainIndex[i,:] = perm[setdiff(1:n,validStart:validEnd)]
end
```

```julia

    for sigma in 2.0.^(-15:15)
        sumErr = 0
        for i in 1:10
            Xtrain = X[trainIndex[i,:],:]
            ytrain = y[trainIndex[i,:],:]
            Xvalid = X[validIndex[i,:],:]
            yvalid = y[validIndex[i,:],:]
            # Compute the error on the validation set
            model = leastSquaresRBF(Xtrain,ytrain,sigma,10^-12.0)
            yhat = model.predict(Xvalid)
            validError = sum((yhat - yvalid).^2)/(9*n/10)
            #@printf("With sigma = %.3f, validError = %.2f\n",sigma,validError)
            # Keep track of the lowest validation error
            sumErr += validError
        end

        avgErr = sumErr/10
        if avgErr < minErr
            minErr = avgErr
            bestSigma = sigma
        end
    end
```



For plot picture is like:

2.3 Cost of Non-Parametric Bases

1.Computing distance between the x_i and x_j cost O(n^2d).Then computing the function $(Z^TZ + \lambda)w = Z^Ty$ which cost O(n^3), sum of two cost is O($n^2d + n^3$)

2.First,Compute the distance between t objects and x_i with d features cost O(tnd).Then, Computing prediction with the cost with Z*w with txn matrix from Z and nx1 matrix from w, we can get tx1 matrix, for one example cost only O(tn). Therefore the cost of classifying t new example with this model cost O(tnd).

3.
Runtime for building RBFs: O($n^3 + n^2d$)
Runtime for building linear basis:O($nd^2 + d^3$).
math:
$n^3 + n^2d > nd^2 + d^3$
$n^2 > d^2$
Thus, when $n > d$ we choose RBFs than linear basis

4.
Runtime for testing RBFs: O(tnd)
Runtime for testing linear basis: O(td)
Thus, linear basis is always cheaper than RBFs.

3 Logistic Regression with Sparse Regularization

3.1 Logistic Regression

According to data report that we know with the condition that the number of features is the same. TrainError is decreasing and the validError is decreasing.

3.2 L2-Regularization

```
function logRegL2(X,y,lambda)
    (n, d) =size(X)

    #Initial guess
    w = zeros(d,1)

    #Function we'ra going to minimize(and that computes gradient)
    funObj(w) = logisticObjL2(w,X,y)

    # Solve least squares problem
    w = findMin(funObj,w,derivativeCheck=true)
    # Make linear prediction function
    predict(Xhat) = sign.(Xhat*w)

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

```
function logisticObjL2(w,X,y)
    yXw = y.*(X*w)
    f = sum(log.(1 + exp.(-yXw))) + lambda/2 * norm(w)^2
    g = -X'*(y./(1+exp.(yXw))) + lambda* w
    return (f,g)
end
```

Training error increases to 0.002 and validation error decreasing to 0.074 with the same number of the feature, there are 101 features in the L2-regularization model. And the largest gradient descent iteration is going to be t=O(max(eig(X'X+λ))/min(eig(X'X+λ))) = 6

3.3 L1-Regularization

```
function logRegL1(X,y,lambda)
    (n, d) =size(X)

    #Initial guess
    w = zeros(d,1)

    #Function we'ra going to minimize(and that computes gradient)
    funObj(w) = logisticObjL1(w,X,y)

    # 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 logisticObjL1(w,X,y)
    yXw = y.*(X*w)
    f = sum(log.(1 + exp.(-yXw)))
    g = -X'*(y./(1+exp.(yXw)))
    return (f,g)
end
```

Training error decrease to 0 and validation error decreasing to 0.052, the number of the feature in L1-regularization model is 71.

3.4 L0-Regularization

```

        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]
            w = zeros(length(Sj), 1)
            w = findMin(funObj,w,verbose= false)
            (f,~) = funObj(w)
            score = f + lambda*length(Sj)

            if score < minScore
                minScore = score
                minS = Sj
            end
            # PUT YOUR CODE HERE
        end
        S = minS
    end

```

Training error maintain to 0 and validation error decreasing to 0.018,the number of the feature in L0-regularization model is 24

4 Logistic Regression with Sparse Regularization

1. Because validation error will try to eliminate noise in order to achieve a better score but noise should be part of true model. In one word, validation error tends to choose a larger parameter or a more complicated model.Also BIC can find a true model as n goes to infinity large.

2. Exhaustive search has 2^d models and is hard to find an efficient score to choose a best model (optimization bias is huge if we choose validation error). Therefor we use forward selection which substantially decreases number of possible models. Also, forward selection reduces false positives.

3. When λ is small, model we get will be complicate and train error is small but approximation error is large.
When λ is large or, model is simple and therefore large train error and small approximation error.

4. When there is bunch of irrelevant features, L1-regularization is preferred because it is robust to irrelevant features and can do feature selection.
L2-regularization is preferred over L1 because it is differentiable and therefore can be computed easily, also L2 has a unique solution.

5. Generate several bootstrap samples, run feature selection using L1 as score, record selected features, find the union of these selected features.

6. Least squares penalize for being too right

7. SVM will found a classifier with largest margin, but perceptron will be satisfied as long as a classifier is found.

8. When d is large, polynomial basis (Z) would be too large to be stored. Kernel trick avoids storing Z but stores $K = ZZ^T$ because K is what really needed to make prediction and K's size is independent of d.

9. The perceptron algorithm, SVMs,logistic regression.