Homework 1.

Juan G. García

January 15, 2017

1 Fundamentals

1.1 Matrix Notation

We are going to find the gradient ∇f and the Hessian $\nabla^2 f$ of the following functions.

1.1.1 Linear Function I

Consider the function

$$f(w) = w^T a + \alpha + \sum_{j=1}^{d} w_j a_j.$$

Observe that this function can be written as

$$f(w) = \alpha + 2\sum_{j=1}^{d} w_j a_j.$$

With this, the derivative with respect to the $i-th$ derivative is given by

$$\frac{\partial f}{\partial w_i} = 2a_i,$$

this is the $i-th$ coordinate of the vector $2a$, hence

$$\nabla f(w) = 2a.$$

Since this function is independent of w we conclude that

$$\nabla^2 f(w) = 0.$$

1.1.2 Linear Function II

Now consider the function

$$f(w) = a^T w + a^T A w + w^T A^T b.$$

From the previous exercise we know that

$$\nabla(a^T w) = a. \tag{1}$$

On the other hand we have

$$\frac{\partial(a^T A w)}{\partial w_i} = \sum_{k=1}^{d}\sum_{l=1}^{d} a_k a_{kl} \frac{\partial w_l}{\partial w_i},$$

in the last equality we used the linearity of the $\frac{\partial}{\partial w_i}$ operator. Recalling that the coordinates w_i are independent we have that $\frac{\partial w_l}{\partial w_i} = \delta_{li}$, where

$$\delta_{li} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

With this we conclude

$$\frac{\partial(a^T A w)}{\partial w_i} = \sum_{k=1}^{d} a_k a_{ki}.$$

This is the $i - th$ component of the vector $A^T a$, hence we conclude

$$\nabla(a^T A w) = A^T a. \tag{2}$$

By doing the same steps as before, it is not hard to conclude that

$$\nabla(w^T A b) = Ab. \tag{3}$$

By combining equations (1),(2) and (3), we get

$$\nabla f = a + A^T a + Ab.$$

Since this last expression is independent of w we conclude

$$\nabla^2 f(w) = 0.$$

1.1.3 Cuadratic Function

Consider the function

$$f(w) = w^T w + w^T X^T X w + \sum_{i=1}^{d} \sum_{j=1}^{d} w_i w_j a_{ij}.$$

To find the gradient, first observe that if B is a $d \times d$ matrix and x is a $d \times 1$ vector then

$$\nabla(\|Bx\|_2^2) = 2B^T Bx.$$

With this in mind we get by setting $B = I$ where I is the $d \times d$ identity matrix

$$\nabla(w^T w) = \nabla(\|w\|_2^2) = 2I^T I w = 2w.$$

on the other hand by setting $B = X$ we get

$$\nabla(w^T X^T X w) = \nabla(\|Xw\|_2^2) = 2X^T X w.$$

Finally observe that

$$\sum_{i=1}^{d} \sum_{j=1}^{d} w_i w_j a_{ij} = w^T A w.$$

By doing the same component-wise analysis as before we conclude

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

The final statement is then

$$\nabla f(w) = 2w + 2X^T X w + (A + A^T) w = 2(I + X^T X + \frac{A + A^T}{2}) w.$$

Since the gradient of f is linear w.r.t w we conclude that

$$\nabla^2 f = 2(I + X^T X + \frac{A + A^T}{2}).$$

1.1.4 L2-regularized weighted least squares

Observe that the first summand can be written as

$$g(w) = \sum_{i=1}^{n} v_i (\sum_{j=1}^{d} w_j x_j^i - y^i)^2.$$

therefore by the chain rule we conclude

$$\frac{\partial g(w)}{\partial w_k} = 2\sum_{i=1}^{n} v_i(\sum_{j=1}^{d} w_j x_j^i - y^i)\sum_{j=1}^{d}\delta_{jk}x_j^i.$$

That is

$$\frac{\partial g(w)}{\partial w_k} = 2\sum_{i=1}^{n} v_i(\sum_{j=1}^{d} w_j x_j^i - y^i)x_k^i.$$

This is the $k-th$ component of the vector $\sum_{i=1}^{n} v_i(w^T x^i - y^i)x^i$. On the other hand we know that

$$\nabla(\|w\|_2^2) = 2w.$$

hence

$$\nabla f(w) = \sum_{i=1}^{n} v_i(w^T x^i - y^i)x^i + \lambda w.$$

To find the Hessian of f it is convenient to remember that if a, b, c are vectors then $(a^T b)c = (c \otimes a)b$. Hence

$$\nabla f(w) = \sum_{i=1}^{n}(x_i \otimes x_i)w - \sum_{i=1}^{n} y^i x^i + \lambda w.$$

Since this is an affine transfomration of w we conclude that

$$\nabla^2 f(w) = \sum_{i=1}^{n} x_i \otimes x_i + \lambda I.$$

1.1.5 Weighted L2-regularized probit regression

1.2 Cross-Validation

1.2.1 Plot leastSquaresRBFL2.m

Using the function leastSquaresRBFL2.m with the aid of the function rbf-Basis.m we obtain the following plot with $\sigma = 1$ and $\lambda = 1$.

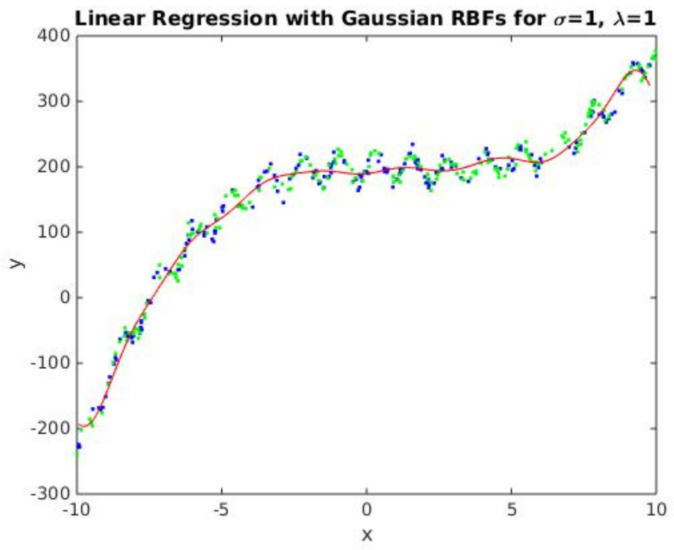

Figure 1: Linear regression with Gaussian Radial Basis Functions.

1.2.2 Computational Cost

- Linear Basis

For the linear with number of features d the feature matrix X is an $n \times d$ matrix. To train this case we need to calculate

$$w = (X^T X)^{-1} X^T y.$$

The multiplication of $X^T X$ to get a $d \times d$ matrix costs $\mathcal{O}(nd^2)$. The cost of inverting the system is $\mathcal{O}(d^3)$. The multiplication of $X^T y$ takes $\mathcal{O}(nd)$. Finally the multiplication of the $d \times d$ matrix $(X^T X)^{-1}$ with the $d \times 1$ vector $X^T y$ is $\mathcal{O}(d^2)$. Adding up all this contributions we conclude that

$$cost = \mathcal{O}(d^3) + \mathcal{O}(nd^2).$$

So in the previous exercise with $d = 2$ we get that the cost is $\mathcal{O}(n)$.

- Gaussian RBFs

If the number of features is d with n training points we can define the vector

$$\phi(x) = [1\ \phi_1(x), \ldots, \phi_n(x)],$$

where

$$\phi_j(x) = e^{-\frac{1}{2\sigma}\|x-x_j\|_2^2}.$$

Here $x_j \in \mathbb{R}^d$ is the $j-th$ row of the feature maxtrix X. With this we create the matrix Φ with entries

$$\Phi_{ij} = \phi_i(x_j) \qquad \text{and } \Phi_{i1} = 1,$$

hence Φ is an $n \times n+1$ matrix. Now, calculating the euclidean distance of a $d-$ dimensional vector takes $\mathcal{O}(d)$, and we need to calculate n^2 distances, hence the cost of constructing the matrix Φ would be $\mathcal{O}(n^2 d)$ (neglecting the cost of calculation the exponential).

To train the system we need to compute

$$w = (\Phi^T\Phi)^{-1}\Phi^T y.$$

The cost of the multiplication to get the $n+1 \times n+1$ matrix $\Phi^T\Phi$ is $\mathcal{O}(n^3)$. Inverting this matrix also costs $\mathcal{O}(n^3)$. The multiplication of the $n+1 \times n$ matrix Φ^T with the $n \times 1$ vector y costs $\mathcal{O}(n^2)$. Finally the multiplication of the matrix $(\Phi^T\Phi)^{-1}$ with the vector $\Phi^T y$ is $\mathcal{O}(n^2)$. Hence the cost of training with RBFs is

$$cost = \mathcal{O}(n^3) + \mathcal{O}(n^2 d).$$

- Classifying new examples

If we have t examples, in the linear case we need to compute Xw, where X is a $t \times d$ matrix hence the cost is $\mathcal{O}(td)$. For the RBFs we need to calculate Φw where Φ is a $t \times n$ matrix. Calculating this matrix takes $\mathcal{O}(ndt)$. In this case the cost is $\mathcal{O}(ndt)$.

1.2.3 Cross-Validation to set λ and σ

1.3 MAP estimation

1.3.1 Laplace distribution

We have

$$y^i \sim \mathcal{L}(w^T x^i, 1), \qquad w_j \sim \mathcal{L}(0, \frac{1}{\lambda}).$$

In this case the posterior looks like (assuming independence of the variables)

$$p(w|D) \propto \prod_{i=1}^{n} p(y^i|x^i, w) \prod_{j=1}^{d+1} p(w_j).$$

Recalling the definition of laplaces distribution we get

$$p(w|D) \propto e^{-\sum_{i=1}^{n} |y^i - w^T x_i|} \frac{\lambda}{2} e^{-\sum_{j=1}^{d+1} \lambda |w_j|}.$$

This means that the negative log likelihood is given by (omitting constants that don't depend on w)

$$-\log(p(w|D)) = \sum_{i=1}^{n} |y^i - w^T x_i| + \lambda \sum_{j=1}^{d+1} |w_j|.$$

Recalling the definition of the $L1$ norm we get that minimizing the log likelihood is equivalent to minimize

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

1.3.2 Gaussians

We now assume

$$y^i \sim \mathcal{N}(w^T x^i, \sigma_i^2), \qquad w_j \sim \mathcal{N}(0, \frac{1}{\lambda_j}).$$

As before we have

$$p(w|D) \propto \prod_{i=1}^{n} p(y^i|x^i, w) \prod_{j=1}^{d+1} p(w_j).$$

By changing products into sums in the exponenential and taking $-\log$, it is not hard to check that

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

If we define $\Sigma = diag(\sigma_1, \sigma_2, \dots, \sigma_n)$ and $\Lambda = diag(\lambda_1, \dots, \lambda_{d+1})$ then the above equation can be written as

$$-log(p(w|D)) := f(w) = \|\Sigma(Xw - y)\|_2^2 + \|\Lambda w\|_2^2.$$

1.3.3 Poisson Likelihood

For this exercise we have

$$y^i \sim \mathcal{P}(e^{w^T x^i}), \qquad w_j \sim \mathcal{N}(0, \frac{1}{\lambda}).$$

In this case by taking $-\log$ in the expression

$$p(w|D) \propto \prod_{i=1}^{n} p(y^i|x^i, w) \prod_{j=1}^{d+1} p(w_j).$$

and recalling that $\mathcal{P}(y; \lambda) = \frac{\lambda^y e^{-\lambda}}{y!}$ we get

$$f(w) = \underbrace{\sum_{i=1}^{n} (e^{w^T x^i} - y^i w^T x^i)}_{\text{Data-fitting term}} + \frac{\lambda}{2} \|w\|_2^w.$$

2 Convex Functions

2.1 Minimizing Strictly-Convex Quadratic Functions

2.1.1 Projection

Consider

$$f(w) = \frac{1}{2}\|w - v\|^2.$$

Since there are no restrictions on w the obvious solution is $w = v$, then

$$f(v) = 0.$$

2.1.2 Least Square

For the function

$$\frac{1}{2}\|Xw - y\|^2 + \frac{1}{2}w^T\Lambda w.$$

It is convenient to remember the identities

$$\nabla(\|Ax - b\|^2) = 2(A^TAx - A^Tb), \tag{4}$$

and

$$\nabla x^TAx = (A + A^T)w.$$

With this it is not hard to see that

$$\nabla(f(w)) = X^TXw - X^Tb + \frac{\Lambda + \Lambda^T}{2}w.$$

Setting this equation equal to zero and solving for w we get

$$w = \left(X^TX + (\Lambda + \Lambda^T)/2\right)^{-1} X^Tb.$$

2.1.3 Weighted Least Squares Shruking to $w^{(0)}$

In matrix notation the function of interest can be written as

$$f(w) = \frac{1}{2}(Xw - y)^TV(Xw - y) + \frac{\lambda}{2}\|w - w^0\|^2.$$

Where V contains the values $v_i \geq 0$ in the diagonal. If we define

$$\sqrt{V} = diag(\sqrt{v_1}, \ldots, \sqrt{v_n}).$$

Then the above equation can be written as

$$f(w) = \frac{1}{2}\|\sqrt{V}(Xw - y)\|^2 + \frac{\lambda}{2}\|w - w^0\|^2,$$

or

$$f(w) = \frac{1}{2}\|\sqrt{V}Xw - \sqrt{V}y)\|^2 + \frac{\lambda}{2}\|w - w^0\|^2.$$

By setting $\sqrt{V}X = A$ and $\sqrt{V}y = b$ we can use equation (4) to get

$$\nabla(\frac{1}{2}\|\sqrt{V}Xw - \sqrt{V}y)\|^2) = ((\sqrt{V}X)^T(\sqrt{V}X)w - (\sqrt{V}X)^T\sqrt{V}y),$$

Since $\sqrt{V}$ is simmetric then $\sqrt{V}^T\sqrt{V} = \sqrt{V}^2 = V$, therefore

$$\nabla(\frac{1}{2}\|\sqrt{V}Xw - \sqrt{V}y)\|^2) = X^TVXw - X^TVy.$$

On the other hand for the term $\|w-w^0\|^2$ we can use equation (4) by setting $A = I$ and $b = w^0$. By doing so we get

$$\nabla(\|w - w^0\|^2) = 2(w - w^0).$$

Hence we conclude

$$\nabla f(w) = X^TVXw - X^TVy + \lambda(w - w^0).$$

Equating to zero and solving for w we obtain

$$w = \left(X^TVX + \lambda I\right)^{-1}\left(X^TVy + \lambda w^0\right).$$

2.2 Proving Convexity

2.2.1 Negative Log

The domain of the function $-\log(aw)$ is $(0, \infty)$ which is clearly convex, on the other hand it is straight forward to see that

$$f''(w) = \frac{1}{w^2} > 0,$$

hence f is convex.

2.2.2 Quadratic with positive semi-definite A

for the function

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

its domain is all the space, hence convex domain. The gradient is given by

$$\nabla f = \frac{A + A^T}{2} w + b,$$

hence its Hessian is

$$\nabla^2 f = \frac{A + A^T}{2}.$$

Since A is possitive semi-definited, A^T and $\nabla^2 f$ are. Hence f is convex.

2.2.3 Any norm

The domain of any norm is the whole space, hence is convex. On the other hand if $\lambda \in [0, 1]$ and $w, v \in \mathbb{R}^d$, then by the triangle inequality we get

$$\|\lambda w + (1 - \lambda)v\|_p \leq \lambda\|w\|_p + (1 - \lambda)\|v\|_p.$$

which is the definition of convexity.

2.2.4 Logistic Regression

Finally we proved in class that the function

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

Has as Hessian

$$\nabla^2 f = X^T DX.$$

Where D is a diagonal matrix with elements

$$D_{ii} = sgm(y_i w^T x_i) sgm(-y_i w^T x_i).$$

Since the sigmoind function is positive we conclude that D is positive definite hence the inner product

$$a^t Db,$$

induces the norm

$$\|a\|_D := a^T Da.$$

With this in mind if $z \in \mathbb{R}^d$ then

$$z^T X^T D X z = \|Xz\|_D \geq 0.$$

hence $\nabla^2 f \succeq 0$, this shows that f is convex. Needless to say, to domain of f is the whole space, hence convex.

Now we are going to prove convexity using operations that preserve convexity

2.2.5 regularized regression with arbitrary p-norm

The functions

$$\|Xw - y\|_p,$$

and

$$\lambda \|Aw\|_q,$$

are composition of affine maps along with multtiplication with a positive scalar, since any norm is convex we conclude that the function (whose domain is the whole space)

$$f(w) = \|Xw - y\|_p + \lambda \|Aw\|_q,$$

is convex.

2.2.6 Support Vector Regression

constants are trivially convex, hence 0 and $-\epsilon$ are convex,also the absolute value is convex, this means that

$$|y_i - w^T x_i| - \epsilon,$$

is convex. Since max and addition preserves convexity we conclude

$$\sum_{i=1}^{n} \max\{0, |w^T x_i - y_i| - \epsilon\},$$

As shown before norms are convex and since $\lambda \geq 0$ we get that

$$f(w) = \sum_{i=1}^{n} \max\{0, |w^T x_i - y_i| - \epsilon\} + \frac{\lambda}{2}\|w\|_2^2,$$

is convex. Once again the domain is the whole space, which is convex.

2.2.7 3 largest-magnitude elements

As explained before, addition, and maximum preserves convexity and since the absolute value is convex then

$$f(x) = \max_{ijk}\{|x_i| + |x_j| + |x_k|\},$$

is convex as well. In this case we assume the domain to be convex.

2.3 Robust Regression

2.3.1 robustRegression.m

Below it is shown the script robustRegression.m

```
%Script to perform a robust regresion

%Author: Juan Garcia
%Date: Jan 14 2017
%email:jggarcia@sfu.ca


function coef=robustRegression(X,y)

        %Arguments: X is the training input data
        % y is the training output data

        %Output: coef is a strct coef.w is the vector
            (w0,w1,r1,r2,...,rn)
        %coef.predict predicts an output y given an
            input x


        [n,d]=size(X);

        %Adding the bias variable

        one=ones(n,1);
        Z=[one X];

        %Setting up the linprog problem


        f=[0;0;one];

        aux=-eye(n);
        A=[Z aux;-Z aux];
        b=[y;-y];

        coef.w=linprog(f,A,b);
```

```
            coef.predict=@predict;

end

function  yhat=predict(coef,Xhat)
            [test ,d]=size(Xhat);
            Zhat=[ones(test ,1)  Xhat];
            yhat=Zhat*coef.w(1:2);
end
```

Using this script on the data outliersData.mat we get the following results

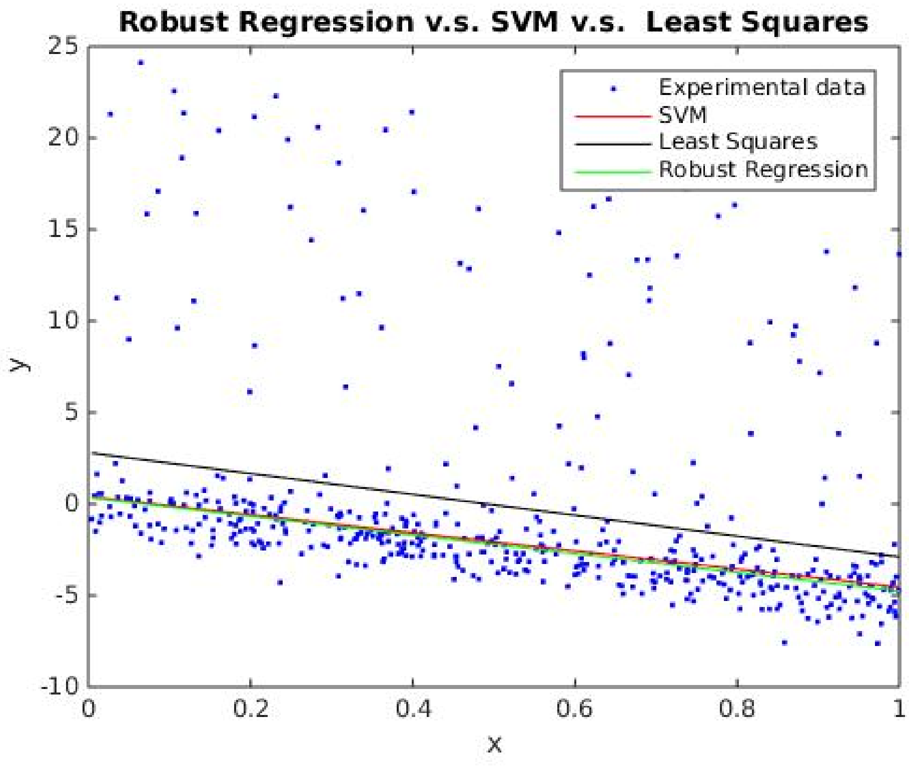

Figure 2: Comparison for the robust regression with SVM and least squares.

In this case the training error was

$$\|\frac{\hat{y} - y_{test}}{500}\|_1 = 3.0666.$$

Now we are going to analyze how SVM behaves.

2.3.2 SVM as a Linear Program

For the regression problem with SVM we are going to minimize the function

$$f(w) = \sum_{i=1}^{n} \max\{0, |w^T x_i - y_i| - \epsilon\}.$$

We are going to convert this into a linear program. first we define the variables r_i to be such that

$$r_i \geq |w^T x_i - y_i| - \epsilon,$$

and $r_i \geq 0$. Since $|a| = \max\{a, -a\}$ we can write the condition above as

$$r_i \geq max\{w^T x_i - y_i, y_i - w^T x_i\} - \epsilon,$$

or equivalently

$$\begin{aligned} r_i &\geq w^T x_i - y_i - \epsilon, \\ r_i &\geq y_i - w^T x_i - \epsilon. \end{aligned}$$

With this our linear problem reads as (Organizing terms to be used as inputs in Matlab)

$$\min_{r \in \mathbb{R}^n, w \in \mathbb{R}^d} \sum_{i=1}^{n} r_i,$$

s.t.

$$\begin{aligned} -r_i &\leq 0, \\ w^T x_i - r_i &\leq y_i + \epsilon, \\ -w^T x_i - r_i &\leq -y_i + \epsilon. \end{aligned}$$

2.3.3 SVM Regression

Now we do a regression using the script svRegression.m shown below

```
%Script to perform a SVM regresion

%Author: Juan Garcia
%Date: Jan 14 2017
%email:jggarcia@sfu.ca


```

```
function coef=svRegression(X,y,epsilon)

        %Arguments: X is the training input data
        % y is the training output data
        % epsilon is the sensitivity loss (real number
           )

        %Output: coef is a strct coef.w is the vector
           (w0,w1,r1,r2,...,rn)
        %coef.predict predicts an output y given an
           input x


        [n,d]=size(X);

        %Adding the bias variable

        one=ones(n,1);
        Z=[one X];

        %Setting up the linprog problem


        f=[0;0;one];

        aux=-eye(n);
        aux2=[Z aux;-Z aux];
        A=[zeros(n,2) aux;aux2];

        b=[zeros(n,1);y+epsilon;-y+epsilon];

        coef.w=linprog(f,A,b);
        coef.predict=@predict;

end

function yhat=predict(coef,Xhat)
        [test,d]=size(Xhat);
        Zhat=[ones(test,1) Xhat];
        yhat=Zhat*coef.w(1:2);
```

```
end
```

The regression obtained for $\epsilon = 1$ is shown in Figure 2. For this case the training error was

$$\|\frac{\hat{y} - y_{test}}{500}\|_1 = 3.0746.$$

Using SVM with $\epsilon = 1$ was slightly less accurate than robust regression.

3 Numerical Optimization

3.1 Gradient Descent and Newton's Method

By modifying the script findMin.m to count the number of iterations and backtracking we can assess the performance under changes in how to pick the parameter alpha in the backtracking line-search.

3.1.1 Replacing $\alpha \leftarrow \alpha/2$

1. Number of Backtracking: 69
2. Number of Iterations: 13

3.1.2 Reseting $alpha$ to $\alpha \leftarrow -\alpha\frac{v^T\nabla f(w)}{v^Tv}$

In this case by reseting α as $\alpha \leftarrow -\alpha\frac{v^T\nabla f(w)}{v^Tv}$ we have to posibilities, we can let the update of α to be the third order Hermite polynomial interpolation or as before $\alpha \leftarrow \frac{\alpha}{2}$. We are going to examine both cases.

- Cubic Hermite polynomial interpolation for updating α.
  1. Number of Backtracking: 6
  2. Number of Iterations: 10
- Updating $\alpha \leftarrow \frac{\alpha}{2}$.
  1. Number of Backtracking: 6
  2. Number of Iterations: 10

As we can see reseting α in this way allows the possibility to choose a simple update as $\alpha \leftarrow \frac{\alpha}{2}$ without compromising efficiency.

3.1.3 Fixing the steps size to $\alpha = \frac{1}{L}$ with $L = \frac{1}{4}\max\{eig(X^T X)\} + \lambda$

In this case, where α is not being updated we have that the Armijo condition is not satisfied.

3.1.4 Changing the direction to $d = (\nabla^2 f(w))^{-1}\nabla f)$

As before we have to choices, by letting α to update according to the interpolation of the Hermite polynomial or by splitting α by 2 every time. We are going to examine both.

- Cubic Hermite polynomial interpolation for updating α.
    1. Number of Backtracking: 0
    2. Number of Iterations: 5
- Updating $\alpha \leftarrow \frac{\alpha}{2}$.
    1. Number of Backtracking: 0
    2. Number of Iterations: 5

In this case, since there is no backtracking is irrelevant what α update we use. As we can see there is a tradeoff. The number of iterations before reaching the tolerance is very low compared to the other methods, but there is the extra cost of calculating the Hessian and solving the system $\nabla^2 f(w)d = \nabla f$ using Cholesky factorization (Taking advantage of the convexity of the objective function).