CPSC 540 — Assignment 1

Chaurette, Laurent	Knill, Stephanie	Vincart-Emard, Alexandre
84060128	54882113	85135127

1 Fundmanentals

The purpose of this question is to give you practice using the mathematical and coding notation that we will adopt in the course.

1.1 Matrix Notation

For this question we'll use the following Householder-like notation:

1. α is a scalar.
2. w, a, and b are d by 1 column-vectors.
3. y and v are n by 1 column-vectors (with elements y^i and v_i).
4. A is a d by d matrix, not necessarily symmetric (with elements a_{ij}).
5. V is a diagonal matrix with v along the diagonal.
6. B is a diagonal matrix with b along the diagonal.
7. X is a n by d matrix (with rows $(x^i)^T$).

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 linear function*

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

Gradient

$$\begin{aligned} f(w) &= w^T \alpha + \alpha + w^T \alpha \\ &= 2w^T \alpha + \alpha \\ \therefore \nabla f(w) &= 2a \end{aligned}$$

Hessian

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

2. *The linear function*

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

Gradient

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

Hessian

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

3. *The quadratic function*

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

Gradient

$$\begin{aligned} f(w) &= w^T w + w^T X^T X w + w^T A w \\ \therefore \nabla f(w) &= 2w + 2(X^T X)w + (A + A^T)w \end{aligned}$$

Hessian

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

4. *L2-regularized weighted least squares,*

$$f(w) = \frac{1}{2}\sum_{i=1}^{n} v_i (w^T x^i - y^i)^2 + \frac{\lambda}{2}\|w\|^2$$

Gradient

$$\begin{aligned} f(w) &= \frac{1}{2}\sum_{i=1}^{n} v_i [w^T x^i w^T x^i - w^T x^i y^i - 2y^i w^T x^i + y^i y^i] + \frac{\lambda}{2}\|w\|^2 \\ \therefore \nabla f(w) &= \frac{1}{2}\sum_{i=1}^{n} v_i [2x^i x^{iT} w - 2y^i x^i] + \frac{\lambda}{2}\cdot 2w \\ &= \sum_{i=1}^{n} v_i [x^i x^{iT} w - y^i x^i] + \lambda w \end{aligned}$$

Hessian

$$\nabla^2 f(w) = \sum_{i=1}^{n} v_i [x^i x^{iT}] + \lambda I$$

5. *Weighted L2-regularized probit regression,*

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

where $y^i \in \{-1, +1\}$ *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.*

Gradient

$$\begin{aligned} f(w) &= -\sum_{i=1}^{n} \log \Phi(y^i w^T x^i) + \frac{1}{2} \sum_{j=1}^{d} b_j w_j^2 \\ &= -\sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{y^i w^T x^i} e^{-t^2/2} dt + \frac{1}{2} \sum_{j=1}^{d} b_j w_j^2 \end{aligned}$$

For ease of notation, let the vector c be defined by the CDF (i.e. $c = \int_{-\infty}^{y^i w^T x^i} e^{-t^2/2} dt$), with elements c_i. And so we have

$$f(w) = -\sum_{i=1}^{n} \log c_i + \frac{1}{2} \sum_{j=1}^{d} b_j w_j^2$$

Since the derivative of the CDF is the PDF, we will denote it as the vector p with elements p_i. Now computing the gradient

$$\nabla f(w) = -\sum_{i=1}^{n} \frac{1}{c_i} \cdot p_i y^i x^i + b^T w$$

Hessian

Since the PDF p unfortunately contains w, we must take the product rule

$$\nabla^2 f(w) = -\sum_{i=1}^{n} y^i x^i \Big[-\frac{1}{c_i^2} p_i p_i^T y_i^T x_i^T p_i + \frac{1}{c_i} p_i' + \frac{1}{c_i} p_i' y_i^T x_i^T \Big] + b$$

to give us our desired Hessian.

Hint: You can use the results we showed in class 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 fourth example to matrix notation first. For the fifth example, 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 fifth one 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.*

1.2 Regularization and Cross-Validation

Download *a1.zip* from the course webpage, and start Matlab 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 error.
4. Draw a figure showing the training/testing data and what the model looks like.

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 $\sigma = 1$.
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 inverting a d by d linear system costs $O(d^3)$.

3. Modify the training/validation procedure to use 10-fold cross-validation on the training set to select λ and σ. Hand in your cross-validation procedure and the plot you obtain with the best values of λ and σ

Note: If you find that calculating the Euclidean distances between all pairs of points takes too long, the following code will form a matrix containing the squared Euclidean distances between all training and test points:

```
[n,d] = size(X);
[t,d] = size(Xtest);
D = X.^2*ones(d,t) + ones(n,d)*(Xtest').^2 - 2*X*Xtest';
```

Element $D(i,j)$ gives the squared Euclidean distance between training point i and testing point j.

1.3 MAP Estimation

In class, we showed that under the assumptions

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

the MAP estimate is equivalent to solving the L2-regularized least squares problem

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

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, including converting to matrix notation):

1. Laplace distribution likelihoods and priors,

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

   Using the "loss plus regularizer" framework, we have

$$w^* \in \arg\min_{w \in W} \sum_{i=1}^{n} -\log p(y^i|w, x^i) - \log p(w)$$

   and the probabilities for the Laplacian distribution

$$p(y^i|w, x^i) = \frac{1}{2}\exp(-|y^i - w^T x^i|)$$
$$p(w_j) = \frac{1}{2}\exp(-\lambda|w|)$$

This gives us our desired minimization

$$\begin{aligned} w^* &\in \arg\min_{w\in W} \sum_{i=1}^{n} -\log\Big[\frac{1}{2}\exp(-|y^i - w^T x^i|)\Big] - \log\Big[\frac{1}{2}\exp(-\lambda|w|)\Big] \\ &\in \arg\min_{w\in W} \sum_{i=1}^{n} -\log\Big(\frac{1}{2}\Big) - \log\Big[\exp(-|y^i - w^T x^i|)\Big] - \log\Big(\frac{1}{2}\Big) - \log\Big[\exp(-\lambda|w|)\Big] \\ &\in \arg\min_{w\in W} \sum_{i=1}^{n} |y^i - w^T x^i| + \lambda|w| \\ &\in \arg\min_{w\in W} ||y - w^T x^i||_1 + \lambda||w||_1 \end{aligned}$$

or in matrix form

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

2. Gaussians with separate variance for each training example and variable,

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

Again using the "loss plus regularizer" framework, we have

$$w^* \in \arg\min_{w\in W} \sum_{i=1}^{n} -\log p(y^i|w, x^i) - \log p(w)$$

however, for our probabilities we now have

$$\begin{aligned} p(y^i|w, x^i) &= \frac{1}{\sqrt{2\sigma_i^2\pi}} \exp\Big[-\frac{(y^i - w^T x^i)^2}{2\sigma_i^2}\Big] \\ p(w_j) &= \frac{1}{\sqrt{\frac{2\pi}{\lambda_i}}} \exp\Big[-\frac{\lambda_i}{2} w_j^2\Big] \end{aligned}$$

This gives us our desired minimization

$$\begin{aligned} w^* &\in \arg\min_{w\in W} \sum_{i=1}^{n} -\log\Big[\frac{1}{\sqrt{2\sigma_i^2\pi}} \exp\Big[-\frac{(y^i - w^T x^i)^2}{2\sigma_i^2}\Big]\Big] - \log\Big[\frac{1}{\sqrt{\frac{2\pi}{\lambda_i}}} \exp\Big[-\frac{\lambda_i}{2} w_j^2\Big]\Big] \\ &\in \arg\min_{w\in W} \sum_{i=1}^{n} \frac{(y^i - w^T x^i)^2}{2\sigma_i^2} + \frac{\lambda_i}{2} w_j^2 \\ &\in \arg\min_{w\in W} \frac{1}{2}\Big|\Big|\frac{(y^i - w^T x^i)^2}{\sigma_i^2}\Big|\Big|^2 + \frac{1}{2}\Big|\Big|\sqrt{\lambda_i} w_i\Big|\Big|^2 \end{aligned}$$

or in matrix form

$$f(w) = \frac{1}{2}||\sigma^{-1}(Xw - y)||^2 + \frac{1}{2}||Dw||^2$$

where $\sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$ and $D = \text{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n})$

3. Poisson-distributed likelihood (for the case where y^i represents discrete counts) and Gaussian prior,

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

Using the "loss plus regularizer" framework, we have

$$w^* \in \arg\min_{w \in W} \sum_{i=1}^{n} -\log p(y^i|w, x^i) - \log p(w)$$

and the probabilities for the distributions

$$\begin{aligned} p(y^i|w, x^i) &= \frac{\exp(w^T x^i)^{y^i} \cdot \exp[-\exp(w^T x^i)]}{y_i!} \\ &= \frac{\exp(w^T x^i y^i) \cdot \exp[-\exp(w^T x^i)]}{y_i!} \\ p(w_j) &= \frac{1}{\sqrt{\frac{2\pi}{\lambda_i}}} \exp\Big[-\frac{\lambda_i}{2} w_j^2\Big] \end{aligned}$$

This gives us our desired minimization

$$\begin{aligned} w^* &\in \arg\min_{w \in W} \sum_{i=1}^{n} -\log\left[\frac{\exp(w^T x^i y^i) \cdot \exp[-\exp(w^T x^i)]}{y_i!}\right] - \log\left[\frac{1}{\sqrt{\frac{2\pi}{\lambda_i}}} \exp\Big[-\frac{\lambda_i}{2} w_j^2\Big]\right] \\ &\in \arg\min_{w \in W} \sum_{i=1}^{n} -w^T x^i y^i + \exp(w^T x^i) + \log(y_i!) + \frac{\lambda}{2} w_i^2 \\ &\in \arg\min_{w \in W} \sum_{i=1}^{n} -w^T x^i y^i + \exp(w^T x^i) + \frac{\lambda}{2} w_i^2 \\ &\in \arg\min_{w \in W} \sum_{i=1}^{n} \Big[-w^T x^i y^i + \exp(w^T x^i)\Big] + \frac{\lambda}{2}||w||^2 \end{aligned}$$

or equivalently

$$f(w) = \sum_{i=1}^{n} \Big[-w^T x^i y^i + \exp(w^T x^i)\Big] + \frac{\lambda}{2}||w||^2$$

2 Convex Functions

2.1 Minimizing Stricly Convex Quadratic Functions

1.

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

Taking the gradient of $f(w)$, we get

$$\begin{aligned} \nabla f(w) &= \nabla\left(\frac{1}{2}(w-v)^T(w-v)\right) \\ &= \frac{1}{2}\nabla\left(w^Tw - 2w^Tv + v^Tv\right) \\ &= w - v. \end{aligned} \tag{2}$$

We now set the gradient to 0 to find the critical points

$$\nabla f(w_{min}) = 0 = w_{min} - v, \tag{3}$$

implies

$$w_{min} = v. \tag{4}$$

2.

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{1}{2}w^T\Lambda w \tag{5}$$

Taking the gradient of $f(w)$, we get

$$\begin{aligned} \nabla f(w) &= \nabla\left(\frac{1}{2}\|Xw - y\|^2 + \frac{1}{2}w^T\Lambda w\right) \\ &= \frac{1}{2}\nabla\left[w^T\left(X^TX + \Lambda\right)w - 2w^TX^Ty + y^Ty\right] \\ &= \frac{1}{2}\left(2X^TX + \Lambda + \Lambda^T\right)w - X^Ty. \end{aligned} \tag{6}$$

Setting the gradient to 0, we find the critical point

$$0 = \frac{1}{2}\left(2X^TX + \Lambda + \Lambda^T\right)w_{min} - X^Ty \tag{7}$$

which gives,

$$\left(2X^TX + \Lambda + \Lambda^T\right)w_{min} = 2X^Ty \tag{8}$$

and finally,

$$w_{min} = 2\left(2X^TX + \Lambda + \Lambda^T\right)^{-1}X^Ty \tag{9}$$

3.

$$f(w) = \frac{1}{2}\sum_{i=1}^{n} v_i \left(w^T x_i - y_i\right)^2 + \frac{\lambda}{2}\|w - w^0\|^2 \tag{10}$$

Taking the gradient of $f(w)$, we get

$$\begin{aligned}\nabla f(w) &= \nabla\left(\frac{1}{2}\sum_{i=1}^{n} v_i \left(w^T x_i - y_i\right)^2 + \frac{\lambda}{2}\|w - w^0\|^2\right) \\ &= \frac{1}{2}\nabla\left(\sum_{i=1}^{n} v_i \left(w^T x_i x_i^T w - 2w^T x_i y_i + y_i^2\right) + \lambda\left(w^T w - 2w^T w^0 + {w^0}^T w^0\right)\right) \\ &= \sum_{i=1}^{n} v_i \left(x_i x_i^T w - x_i y_i\right) + \lambda\left(w - w^0\right).\end{aligned} \tag{11}$$

Setting the gradient to 0, we find the critical point

$$0 = \sum_{i=1}^{n} v_i \left(x_i x_i^T w_{min} - x_i y_i\right) + \lambda I\left(w_{min} - w^0\right), \tag{12}$$

where I is the $d \times d$ identity matrix. Thus,

$$\lambda I w^0 + \sum_{i=1}^{n} v_i x_i y_i = \left(\sum_{i=1}^{n} v_i x_i x_i^T + \lambda I\right) w_{min}. \tag{13}$$

We finally find

$$w_{min} = \left(\sum_{i=1}^{n} v_i x_i x_i^T + \lambda I\right)^{-1}\left(\lambda I w^0 + \sum_{i=1}^{n} v_i x_i y_i\right) \tag{14}$$

2.2 Proving Convexity

1.

$$f(w) = -\log(aw) \tag{15}$$

As the log function is twice differentiable, we will prove convexity by calculating the Hessian matrix

$$f''(w) = \frac{1}{w^2} > 0, \quad \forall\, w \in \mathcal{R}^+ \tag{16}$$

The hessian is therefore positive definite over the domain of $f(w)$ which means f is convex.

2.
$$f(w) = \frac{1}{2} w^T A w + b^T w + \gamma \tag{17}$$

Once again, we will use the Hessian matrix criteria,

$$\nabla^2 f(w) = \frac{1}{2}\left(A + A^T\right) \succeq 0. \tag{18}$$

As A is positive semi-definite, its transpose is also positive semi-definite and so is the sum $A + A^T$. The Hessian is therefore positive semi-definite, which implies f(w) is convex.

3.
$$f(w) = \|w\|_p \tag{19}$$

Here, we will choose two points, $(v, f(v))$ and $(w, f(w))$ and show that the line always lies above the function itself. The line joining the two points can be written parametrically as

$$\theta \left(\sum_{i=1}^{n} \mid w_i \mid^p \right)^{1/p} + (1-\theta) \left(\sum_{i=1}^{n} \mid v_i \mid^p \right)^{1/p}, \quad 0 \leq \theta \leq 1 \tag{20}$$

while the function in between those two points can be expressed as

$$\left(\sum_{i=1}^{n} \mid \theta w_i + (1-\theta) v_i \mid^p \right)^{1/p}. \tag{21}$$

However, the triangle inequality gives us directly that

$$\begin{aligned} f(\theta w + (1-\theta)v) &= \left(\sum_{i=1}^{n} \mid \theta w_i + (1-\theta) v_i \mid^p \right)^{1/p} && (22) \\ &\leq \theta \left(\sum_{i=1}^{n} \mid w_i \mid^p \right)^{1/p} + (1-\theta) \left(\sum_{i=1}^{n} \mid v_i \mid^p \right)^{1/p} \\ &= \theta f(w) + (1-\theta) f(v), && (23) \end{aligned}$$

showing that the line joining two points is always above the function itself and therefore $f(w)$ is convex.

4.
$$f(w) = \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i w^T x_i\right)\right) \tag{24}$$

Once again, we will compute the Hessian matrix. First of all, we find the gradient

$$\nabla f(w) = \sum_{i=1}^{n} \frac{-y_i x_i}{1 + \exp\left(-y_i w^T x_i\right)}. \tag{25}$$

The Hessian is therefore

$$\nabla^2 f(w) = \sum_{i=1}^{n} \frac{y_i^2 x_i x_i^T}{\left[1 + \exp\left(-y_i w^T x_i\right)\right]^2}, \tag{26}$$

which is a real symmetric matrix. This implies the Hessian is positive semi-definite and the function $f(w)$ is convex.

5.

$$f(w) = \|Xw - y\|_p + \lambda \|Aw\|_q \tag{27}$$

The first term on the right hand side of equation (27) is simply the affine composition of the $l_p - norm$ which we have showed is convex in 3. That term is therefore convex.

The second term of the expression is the scalar product of the affine composition of the $l_q - norm$ and is also convex.

The sum of two convex functions is convex and therefore $f(w)$ is convex.

6.

$$f(w) = \sum_{i=1}^{N} \max\{0, \mid w^T x_i - y_i \mid -\epsilon\} + \frac{\lambda}{2}\|w\|_2^2 \tag{28}$$

The second term on the right hand side is convex as it is the scalar product of the $l_2 - norm$ which is convex.

As of the first term on the right hand side, the function $g(w) = 0$ is convex. The function $h(w) = w^T x_i - y_i$ is convex as it is a linear function. taking the absolute value of $h(w)$ can be written as $|h(w)| = \max\{-h(w), h(w)\}$ and the maximum of two convex functions is convex. Substracting a constant ϵ remains convex as the addition of convex functions is convex. The maximum of $g(w)$ and $|h(w)| - \epsilon$ is therefore still convex and so is summing over i as it is the sum of convex functions.

As both terms on the right hand side are convex and $f(w)$ is the sum of those two terms, $f(w)$ is thus convex.

7.

$$f(w) = \max_{ijk}\{|x_i| + |x_j| + |x_k|\} \tag{29}$$

Once again, the absolute value $|x_i| = \max\{x_i, -x_i\}$ is convex as it is the maximum of two convex functions. The sum $|x_i| + |x_j| + |x_k|$ is convex as it is the sum of convex functions. The maximum over ijk preserves convexity and therefore $f(w)$ is convex.

2.3 Robust Regression

1. RobustRegression(X,y)

```
(* ::Package:: *)

function [model] = RobustRegression(X,y)
% Solve L1-norm problem through linear program
[n,d] = size(X);
b = zeros(2*n,1);
beq = y(:);
A = [zeros(n,2), eye(n), -eye(n); zeros(n,2) -eye(n) -eye(n)];
Aeq = [X, ones(n,1), eye(n), zeros(n,n)];
f = [0;0;zeros(n,1); ones(n,1)];
w = linprog(f,A,b,Aeq,beq);
model.w = w(1);
model.beta = w(2);

model.predict = @predict;

end

function [yhat] = predict(model,Xhat)
w = model.w;
beta = model.beta;
yhat = Xhat*w+beta;
end
```

This functions obtains an average absolute error of

$$errL1 = 3.0666. \tag{30}$$

2. SVM Regression The goal is to find the minimizer w to the function

$$f(w) = \sum_{i=1}^{n} \max\{0, |w^T x_i - y_i| - \epsilon\}. \tag{31}$$

We transform this into a linear program by introducing two sets of slack variables $\xi_i, \xi_i^* \geq 0$. Our linear program will be the following:

objective function:

$$\sum_{i=1}^{n} \xi_i + \xi_i^* \tag{32}$$

Constraints:

$$w^T x_i - y_i \leq \epsilon + \xi_i \tag{33}$$
$$y_i - w^T x_i \leq \epsilon + \xi_i^* \tag{34}$$
$$\xi_i, \xi_i^* \geq 0. \tag{35}$$

basically, when $w^T x_i - y_i$ is greater(smaller) then 0, we want to minimize $\xi_i(\xi_i^*)$ while $\xi_i^*(\xi_i)$ will go to zero.

3. svRegression(X,y,ϵ)

```
(* ::Package:: *)

function [model] = svRegression(X,y, epsilon)
% Solve epsilon incensitive SVM problem through linear program
[n,d] = size(X);
b = [epsilon*ones(n,1);epsilon*ones(n,1);zeros(2*n,1)]+[y;-y;zeros(2*n,1)];
A = [X, ones(n,1),-eye(n), zeros(n,n);-X,-ones(n,1),zeros(n,n),-eye(n);zeros
f = [0;0;ones(2*n,1)];
w = linprog(f,A,b);
model.w = w(1)
model.beta = w(2)

model.predict = @predict;

end

function [yhat] = predict(model,Xhat)
w = model.w;
beta = model.beta;
yhat = Xhat*w+beta;
end
```

This function obtains an average absolute error of

$$errL1 = 3.0746 \tag{36}$$

3 Numerical Optimization

3.1 Gradient Descent and Newton's Method

In this section, we investigate the effect on performance of various changes to the function *findMin*, which implements a gradient descent complemented by a backtracking line-search to find the step size α that satisfies the Armijo "sufficient decrease" condition

$$f(x^{t+1}) \leq f(x^t) - \gamma\alpha\nabla f(x^t)^T d^t, \quad \gamma = 10^{-4}, \tag{37}$$

where d^t is the direction taken by the gradient during the descent. In particular, we will look at both *funEvals*, the number of function and gradient evaluations needed throughout the optimization process, as well as *btEvals*, the number of backtracking iterations needed to satisfy condition (37).

1. When backtracking, replacing the cubic-Hermit interpolation by

$$\alpha \leftarrow \frac{\alpha}{2} \tag{38}$$

   yields *funEvals* = 83 and *btEvals* = 69, which is evidence of a sharp decline in performance.

2. Keeping the cubic-Hermit interpolation during the backtracking portion of the algorithm, and instead updating α for the next iteration to be the Barzilai-Borwein step-size

$$\alpha \leftarrow -\alpha\frac{v^T\nabla_{\text{new}}f(w)}{v^T v}, \qquad v = \nabla_{\text{new}}f(w) - \nabla_{\text{old}}f(w), \tag{39}$$

   we obtain the best performance with *funEvals* = 21 and *btEvals* = 8.

3. Fixing the step size to be constant and equal to $\alpha = 1/L$, where

$$L = \frac{1}{4}\max\left(\text{eig}(X^T X)\right) + \lambda, \tag{40}$$

   gives us *funEvals* = 35 and *btEvals* = 0.

4. Using the Newton direction

$$d = \left[\nabla^2 f(w)\right]^{-1}\nabla f(w), \tag{41}$$

   performs well, with *funEvals* = 31 and *btEvals* = 0.

Thus our results suggest that (2) > (4) > (3) > (0) $\gg$ (1), where (0) represents the gradient descent routine *findMin* without modification.

3.2 Hessian-Free Newton

The Newton method requires the formation and inversion of the Hessian matrix, which do not come cheap. Instead, it is possible to implement a "Hessian-free" Newton's method by solving

$$\nabla f(w) + \nabla^2 f(w)\, d = 0 \tag{42}$$

for the direction of descent d by using a *conjugate gradient* algorithm, implemented by the function *pcg* in MATLAB.

The first step is to define a function *Hvfunc* that calculates the Hessian-vector product

$$\nabla^2 f(w)\, d = X^T(D(Xd)), \tag{43}$$

where D is a diagonal matrix with elements

$$D_{ii} = \sigma(y^i w^T x^i)\sigma(-y^i w^T x^i), \qquad \sigma(z) = \frac{1}{1+e^{-z}}. \tag{44}$$

With *Hvfunc* in hand, it is straightforward to calculate d via *pcg*, which we may write in MATLAB as

```
>> Hv = @(v) Hvfunc(w,v,X,y,lambda);
>> d = pcg(Hv,-g,optTol);
```

The output of *findMin* on the dataset *rcv1_train_binary.mat* reads

```
>> pcg converged at iteration 12 to a solution with relative
residual 0.0071.
>> Backtracking...
>>      3     -1.13091e+00      4.82515e+03      4.01011e+01
>> pcg converged at iteration 9 to a solution with relative
residual 0.0085.
>> Backtracking...
>>      5     -8.00255e-01      4.23711e+03      1.73080e+01
>> pcg converged at iteration 9 to a solution with relative
residual 0.0077.
>> Backtracking...
>>      7     -8.56343e-01      4.10167e+03      9.88761e+00
>> pcg converged at iteration 9 to a solution with relative
residual 0.0051.
>> Backtracking...
>>      9     -9.23189e-01      4.08601e+03      3.51711e+00
>> pcg converged at iteration 8 to a solution with relative
```

```
residual 0.008.
>> Backtracking...
>>      11      -9.77696e-01       4.08547e+03       2.82335e-01
>> pcg converged at iteration 8 to a solution with relative
residual 0.0051.
>> Backtracking...
>>      13      -9.98461e-01       4.08547e+03       1.28566e-03
>> Problem solved up to optimality tolerance
```

Therefore we conclude that the Hessian-free Newton's method performs remarkably well, using only *funEvals* = 13 and *btEvals* = 6 before reaching a minimum despite X having dimension $20,242 \times 47,236$.

3.3 Multi-Class Logistic Regression

In this section, we consider the softmax probability

$$p(y^i|W, x^i) = \frac{\exp(w_{y^i}^T x^i)}{\sum_{c=1}^k \exp(w_c^T x^i)}, \tag{45}$$

which yields the loss function

$$f(W) = \sum_{i=1}^n \left[-w_{y^i}^T x^i + \log\left(\sum_{c'=1}^k \exp(w_c^T x^i) \right) \right]. \tag{46}$$

In this notation, w_c is column c of the matrix W, and each column acts as the weights of a classifier for one of the k classes, each assigning a probability $p(y^i|W, x^i)$ to example i.

Given that W is a $d \times k$ matrix, we can also express its gradient as a $d \times k$ matrix; its components can be written as

$$\frac{\partial f}{\partial W_{ac}} = -\sum_{i=1}^n \left[X_{ia} \left(\delta_{y^i\, c} - \frac{\exp(XW)_{ic}}{\sum_{c'=1}^k \exp(XW)_{ic'}} \right) \right] \tag{47}$$

$$= -\sum_{i=1}^n \left[X_{ia} \left(\delta_{y^i\, c} - p(y^i = c|W, x^i) \right) \right], \tag{48}$$

where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise.

The loss function and its gradient are straightforward to vectorize in MATLAB, and one needs only to reshape the weights and gradient matrices into vectors in order to use gradient

descent as before. Using the *findMin* function without any of the modifications of Section 3.1 implemented, we reach a validation error of 0.024, or in other words a success rate of 97.6% on the validation set. This error is obtained after the maximum number of function evaluations, $funEvals = 500$, was reached, together with $btEvals = 258$. We note that for $funEvals = 5{,}000$, the validation error drops to 0.01 to yield a 99% success rate. We invite the reader to read our code in the appendix in order to assess our results.