CPSC 540 — Assignment 1

Aaron Berk

January 20, 2017

1 Fundamentals

1.1 Matrix notation

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

$$\begin{aligned}\nabla_w f(w) &= \nabla(2\langle w, a\rangle + \alpha) = 2a \\ \nabla_w^2 f(w) &\equiv 0\end{aligned}$$

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

$$\begin{aligned}\nabla_w f(w) &= \nabla_w\big(\langle a, w\rangle + \langle w, A^T a\rangle + \langle w, A^T b\rangle\big) = a + A^T(a + b) \\ \nabla_w^2 f(w) &\equiv 0\end{aligned}$$

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

$$\begin{aligned} f(w) &= \|w\|_2^2 + \|Xw\|_2^2 + w^T A w \\ \implies \quad \nabla_w f(w) &= 2(1 + X^T X)w + (A + A^T)w \\ \nabla_w^2 f(w) &= 2(1 + X^T X) + A + A^T\end{aligned}$$

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

$$\begin{aligned} f(w) &= \frac{1}{2}(Xw - y)^T V(Xw - y) + \frac{\lambda}{2}\|w\|^2 \\ \implies \quad \nabla_w f(w) &= 2X^T V(Xw - y) + \lambda w \\ \nabla_w^2 f(w) &= 2X^T V X + \lambda I\end{aligned}$$

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

$$\begin{aligned}\nabla_w f(w) &= -\sum_{i=1}^{n} \frac{\phi(y^i w^T x^i)}{\Phi(y^i w^T x^i)} y^i x^i + Bw \\ \nabla_w^2 f(w) &= -\sum_{i=1}^{n} y^i x^i \partial_w \frac{\phi(y^i w^T x^i)}{\Phi(y^i w^T x^i)} = \sum_{i=1}^{n} x^i (x^i)^T \frac{\phi^2(y^i w^T x^i) - y^i w^T x^i \phi(y^i w^T x^i)\Phi(y^i w^T x^i)}{\Phi^2(y^i w^T x^i)}\end{aligned}$$

1.2 Regularization and cross-validation

1.
```
function [model] = leastSquaresRBFL2(X,y,lambda,sigma)
% Compute sizes
[n,d] = size(X);
% Add bias variable
Z = rbfBasis(X, [], sigma);
% Solve least squares problem
w = (Z'*Z + lambda*eye(n))\Z'*y;

model.X = X;
model.w = w;
model.lambda = lambda;
model.sigma = sigma;
model.predict = @predict;
end

function [yhat] = predict(model,Xhat)
[t,d] = size(Xhat);
Zhat = rbfBasis(Xhat, model.X, model.sigma);
yhat = Zhat*model.w;
end

function [Z] = rbfBasis(X1,X2,sigma)
% X1 = query points
% X2 = training points, or X2 = X1
if isempty(X2)
    X2 = X1;
end
if isempty(sigma)
    sigma = 1;
end
n1 = size(X1,1);
n2 = size(X2,1);
d = size(X1,2);
den = 1/sqrt(2*pi*sigma^2);
D = X1.^2*ones(d,n2) + ones(n1,d)*(X2').^2 - 2*X1*X2';
Z = den*exp(-D/(2*sigma^2));
end
```

2. 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 $n \times d$ matrix costs $O(nd)$ and that inverting a $d \times d$ linear system costs $O(d^3)$.

3. Hand in your cross-validation procedure and the plot you obtain with the best values of λ and σ.

   Using the grid

   ```
   lambda = logspace(-7, -5, 30);
   sigma = logspace(-1,0,30);
   ```

   the best λ and σ found were $(\lambda, \sigma) = (4.894 \times 10^{-7}, 0.7279)$ giving a mean-squared test error on the test set of 59.956.

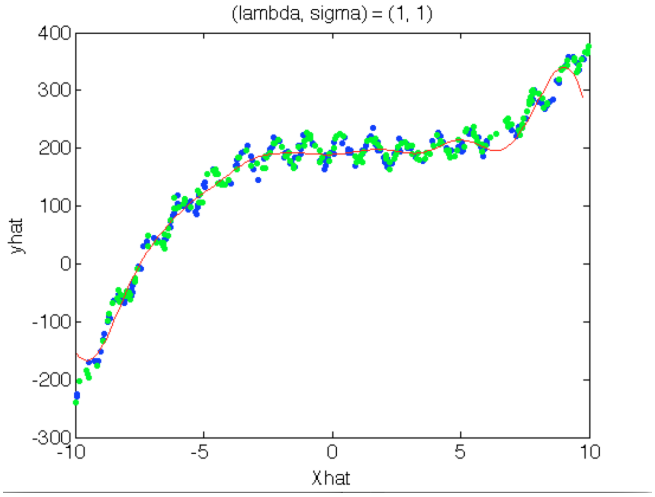

Figure 1: Plot of training data, test data and approximation to test data derived as the solution to the L^2-regularized NLS problem. Output: `squaredTestError = 636.0074`.

```
function [bestModel, score] = gridSearchCV(func, X, y, parms, folds)
% GRIDSEARCHCV performs nfold cross-validation of func on a paramater list ↩
    parms
% passed as a cell-array. Data and labels are given as X and y, resp.
parms = allcomb(parms{:});
[plLen, ~] = size(parms); % parms list length, num parameters (e.g. (lambda, ↩
    sigma))
modelList = cell(plLen,1);
errorList = zeros(1, plLen);
for j = 1:plLen
    display(sprintf('Iteration %d of %d.', j, plLen));
    parm = num2cell(parms(j,:));
    [modelList{j}, errorList(j)] = CV(func,X,y,parm,folds);
end
score = min(errorList);
bestModel = modelList{errorList == score};
end


function [model, score] = CV(func, X, y, parms, folds)
% CV performs nfolds-nfold cross validation
% Note: parms is a cell array
%       folds must be denoted using 1, 2, ..., nfolds

% n observations; d-dimensional feature space
[n, d] = size(X);
% default: 5-fold cross-validation
if isempty(folds) || (numel(folds)==1 && folds < 2)
    nfolds = 5;
    folds = getFolds(n, nfolds);
elseif numel(folds) > 1
    if length(folds) ~= n
        error('fold vector must have length equal to number of observations');
    end
    nfolds = max(folds);
```

```
elseif numel(folds) == 1 && folds >= 2
    nfolds = folds;
    folds = getFolds(n, nfolds);
end

squaredTestError = zeros(1, nfolds);

for j = 1:nfolds
    model = func(X(folds~=j,:), y(folds~=j), parms{:});
    yhat = model.predict(model, X(folds==j,:));
    squaredTestError(j) = sum((yhat-y(folds==j)).^2)/length(yhat);
end
score = mean(squaredTestError);
end

function folds = getFolds(n, nfolds)
% GETFOLDS returns a vector giving fold identities for each element.
folds = zeros(1, n);
perm = randperm(n);
for j = 1:nfolds
    j0 = round(1 + (j-1)*n/nfolds);
    j1 = round(j*n/nfolds);
    folds(perm(j0:j1)) = j;
end
end
```

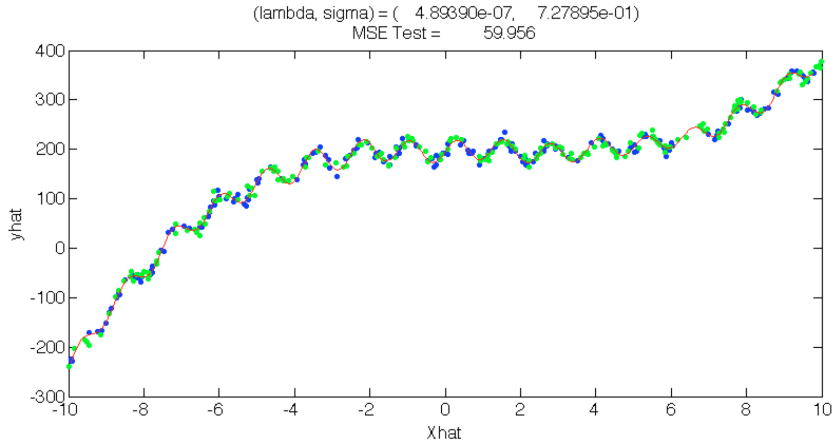

Figure 2: the best λ and σ found were $(\lambda, \sigma) = (4.894 \times 10^{-7}, 0.7279)$ giving a mean-squared test error on the test set of 59.956. Colours correspond to train, test, predict as above.

1.3 MAP estimation

For each of the alternate assumptions below, write it in the âĂIJloss plus regularizerâĂİ framework

1. Laplace distribution likelihoods and priors,

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

Solution

This implies that

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

Computing the negative log-likelihood function then gives

$$L(y^i \mid x^i) = -\log\frac{1}{2} - \log\frac{\lambda}{2} + |w^T x^i - y^i| + \frac{\lambda}{2}\|w\|_1$$

Hence, the function for the associated convex problem is given by

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

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

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

Solution

Using the same process as above, the function for the associated convex problem is given by

$$f(w) = (Xw - y)^T \Sigma^{-1}(Xw - y) + w^T \Lambda w, \quad \Sigma = \operatorname{diag}(\sigma_i^2), \quad \Lambda = \operatorname{diag}(\lambda_j)$$

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

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

Solution

Unpacking the definition of the distribution of y^i, it follows that the conditional probability for y^i and it's negative conditional log-likelihood are given by

$$p(y^i \mid x^i, w) = \frac{\exp(y^i w^T x^i)\exp(-\exp(w^T x^i))}{(y^i)!}$$
$$-\log p(y^i \mid x^i, w) = \log((y^i)!) - y^i w^T x^i + \exp(-w^T x^i)$$

Hence, the associated convex function is given by

$$f(w) = \log(y!) + YXw + \exp(-Xw) + \frac{\lambda}{2}\|w\|_2^2, \qquad z! := z_1!\, z_2! \cdots z_n!, \quad Y := \operatorname{diag}(y^i)$$

2 Convex functions

2.1 Minimizing strictly-convex quadratic functions

1. We seek the orthogonal projection of v onto $\mathbb{R}^n$: $\arg\min_{w\in\mathbb{R}^n} f(w) := \|w - v\|_2^2$. Notice that this is equal to $(\Re v_j)_{j=1}^n =: \Re v$.

2. Assume $f(w) := \frac{1}{2}\|Xw - y\|_2^2 + \frac{1}{2}w^T \Lambda w$. We want to find $w^* := \arg\min_w f(w)$. Taking derivatives, as the objective is smooth:

$$\partial_j f(w) = \partial_j\big(\frac{1}{2}\langle Xw, Xw\rangle - \langle Xw, y\rangle\big) + \lambda_j w_j = (X^T X w)_j - (X^T y)_j + \lambda_j w_j = 0$$

implying that

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

3. $\partial_j f(w) = \sum_{i=1}^n v_i(w_j x_{ij} - y_i)x_{ij} + \lambda(w_j - w^0)$
$0 = v^T X w - X^T \mathrm{diag}(v_i) y + \lambda(w - w^0)$ implying that $w = (v^T X + \lambda)^{-1}(\lambda w^0 + X^T \mathrm{diag}(v_i) y)$

3 Numerical Optimization

3.1 Gradient descent and Newton's method

Report the effect on performance (in terms of number of backtracking iterations and total number of iterations) of making the following changes to `findMin`:

1. When backtracking, replacing the cubic-Hermite interpolation with the simpler strategy of dividing α in half (as suggested in the Boyd & Vandenberghe book).

2. Instead of resetting α to one after the line-search, set it using the Barzilai-Borwein step-size, given by

$$\alpha \leftarrow -\alpha \frac{v^T \nabla f(w)}{v^T v},$$

where v is the new gradient value minus the old gradient value.

3. Fix the step-size α to $1/L$, where L is given by

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

which is the Lipschitz constant of the gradient.

4. Instead of using the gradient direction, set d to the Newton direction which is given by

$$d = [\nabla^2 f(w)]^{-1} \nabla f(w).$$

For the Newton direction, you'll need to make a new objective function that returns the Hessian in addition to the function and gradient, and modify *findMin* to use the Hessian.

The Student's answer:

Method	**funEvals**	**backtracks**	**time (sec)**
Cubic Hermite	23	13	0.049378
$\alpha \mapsto \alpha/2$	83	69	0.130050
Barzilai-Borwein	21	8	0.034763
Lipschitz	35	0	0.058531
Newton	5	0	0.009681

3.2 Hessian-free Newton

Use Matlab's `pcg` function to implement a "Hessian-free" Newton's method, where you use conjugate gradient to solve the Newton system. Report the output of `findMin` on `rcv1_train_binary.mat` when using this strategy and using `optTol` as the tolerance for `pcg`.

The Student's answer:

```
pcg converged at iteration 12 to a solution with relative residual 0.0071.
pcg converged at iteration 9 to a solution with relative residual 0.0089.
     2      0     1.00000e+00     5.18715e+03     6.38540e+01
pcg converged at iteration 9 to a solution with relative residual 0.0071.
     3      0     1.00000e+00     4.22805e+03     1.80539e+01
pcg converged at iteration 8 to a solution with relative residual 0.0088.
     4      0     1.00000e+00     4.09305e+03     1.02721e+01
pcg converged at iteration 8 to a solution with relative residual 0.0053.
     5      0     1.00000e+00     4.08556e+03     2.29591e+00
pcg converged at iteration 8 to a solution with relative residual 0.008.
     6      0     1.00000e+00     4.08547e+03     4.88609e-02
pcg converged at iteration 9 to a solution with relative residual 0.0091.
     7      0     1.00000e+00     4.08547e+03     7.96543e-05
Problem solved up to optimality tolerance
Elapsed time is 1.493054 seconds.
```

3.3 Multi-class logistic regression

Hand in the code and report the validation error