CS74/174 Homework #2

Machine Learning and Statistical Data Analysis: Winter 2016

Due: Feb 13, 2016

The homework should be submitted electronically via Canvas. It should be a ZIP file which includes the main report in a PDF format, and a folder named as "code" that includes all the MATLAB functions and code. Please make sure your code is directly implementable, or you will not receive any credit otherwise. It is important that you include enough detail that we know how you solved the problem, since otherwise we will be unable to grade it. Please include all the figures in the main PDF document with enough comments.

Problem 1: Linear Regression

In this problem, we will implement linear regression with weighted L_2 regularization. Given observation $(x^{(j)}, y^{(j)})_{j=1}^{n}$, where $x^{(j)} = [x_1^{(j)}, \cdots, x_d^{(j)}]$ is the $d \times 1$ features. Linear regression assumes $\hat{y}(x;\ \theta) = \sum_i \theta_i x_i = \theta x^\top$, where $\theta = [\theta_1, \ldots, \theta_d]$ is the unknown parameter and is estimated by minimizing the mean square error with a weighted L_2 regularization:

$$\hat{\theta} = \arg\min_{\theta} \Big\{ \sum_j (y^{(j)} - \sum_i \theta_i x_i^{(j)})^2 + \alpha \sum_i w_i \theta_i^2 \Big\},$$

where α is the regularization coefficient and $w = [w_1, w_2, \ldots, w_d]$ is a set of weights that controls how much we want to regularize on each coordinate of θ. It is useful to re-write the optimization in a compact matrix form:

$$\hat{\theta} = \arg\min_{\theta} \big\{ ||Y^\top - \theta X^\top||_2^2 + \alpha \theta W \theta^\top \big\},$$

where

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}, \qquad X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \cdots & x_d^{(n)} \end{bmatrix}, \qquad W = \mathrm{diag}(w) = \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & w_d \end{bmatrix}.$$

We can use linear regression to fit nonlinear functions by adding artificial features that are nonlinear functions of the original features. For example, assume x_0 is our original feature (we assume x_0 is a scalar in this problem), and then we can construct a set of polynomial features by $x = [1, x_0, x_0^2, \ldots, x_0^\ell]$, and the predictor becomes a polynomial function $\hat{y}(x_0\ ;\ \theta) = \theta_0 + \theta_1 x_0 + \cdots + \theta_\ell x_0^\ell$. I have implemented a simple MATLAB function `poly_feature.m` to calculate the polynomial features.

(a) **[5 Points]** Prove that

$$\hat{\theta} = Y^\top X (X^\top X + \alpha W)^{-1}.$$

Please show your derivation.

(b) **[5 Points]** Please complete `linearRegWL2_train.m` for calculating $\hat{\theta}$ and `linearRegWL2_predict.m` for making the prediction. After you finish your code, you should be able to run `linearRegWL2_run.m`, which loads a small dataset `curveData.mat` and fit a polynomial curve using your linear regression code.

When you load `linearRegWL2_run.m`, you will find data $Xtrain$, $Ytrain$ and $Xtest$; the testing label $Ytest$, which is what you want to predict, is not given to you. Instead, there is a function `calculate_mse_Ytest_curveData.m` that can evaluate the mean square error $mean((Ytest - hatYtest)^2)$ for any given prediction $hatYtest$ that you produce. This is to ensure that you do not use the testing label $Ytest$ in your algorithm in any way, except for the evaluation.

(c) **[5 Points]** Please completing `cross_validation.m` to implement K-fold cross validation. To help you get started, I have already implemented the case of $K = 2$, and showed a simple demo in `linearRegWL2_run.m`. Please complete the code for general K.

(d) **[5 Points]** Run cross validation when $\alpha = [0, 0.1, 1, , 10, 100]$, and decide which α is the best. Please use the following setting (code also included in `linearRegWL2_run.m`):

```
% use 5-fold cross validation
Nfold = 5;
% fix the order of the polynomial feature to be 1
poly_order = 10;
% fix the weights in L2 norm; w(1)=0 means no regularization on the constant term
w = [0, ones(1,poly_order)];
% try different values of alpha
alphaVec = [0,0.1,1,10,100];
for i = 1:length(alphaVec)
    alpha  = alphaVec(i);
    %... please complete the code
    err_xVar_Vec(i) = ... %please complete the code
end
alphabest = ...% please complete the code
% Rerun your model on the whole training data
% with the best alpha and evaluate your performance
hatYtest = ....  %please complete the code
% calculate mse = mean((Ytest-hatYtest).^2);
err_test = calculate_mse_Ytest_curveData(hatYtest);
```

After you find the optimal α, you need to re-run the linear regression on the whole training set and evaluate your perform on the test data using `calculate_mse_Ytest_curveData`. To complete this problem, please finish the code in `linearRegWL2_run.m` and report the optimal α and the testing error.

(e) **[5 Points]** Run cross validation to select the best polynomial order. Please use the following setting (code also included in `linearRegWL2_run.m`):

```
Nfold = 5;
alpha = .1; % fix alpha
poly_order_Vec = [1,2,3,4,5,10,20];
for i = 1:length(alphaVec)
    poly_order  = poly_order_Vec(i);
    % weights in L2 norm; w(1)=0 means no regularization on the constant term
    w = [0, ones(1,poly_order)];
    %... please complete the code
    err_xVar_Vec(i) = ... % please complete the code
```

```
end
poly_order_best = ....
% Rerun your model with the best poly_order and evaluate the testing error
hatYtest = ....  %please complate the code
% calculate mse = mean((Ytest-hatYtest).^2);
err_test = calculate_mse_Ytest_curveData(hatYtest);
```

Similar to Problem (d), you need to complete the code in `linearRegWL2_run.m` and report the optimal `poly_order` and the corresponding testing error.

(f) **[5 Points]** Try several other combinations of `alpha`, `poly_order` and `w`, and report the one that gives the best testing error. You should avoid to call `calculate_mse_Ytest_curveData` too many times (even though I have no way to enforce it), since otherwise your algorithm would overfit to this particular testing data (and does not generalize well if I give you another different testing data); for more information, check out this news post http://www.technologyreview.com/view/538111/why-and-how-baidu-cheated-an-artificial-intelligence-tes

Problem 2: Logistic Regression

(a) Implement `logsumexp.m` to calculate the log-sum-exp function $\Phi(t) = \log \sum_k \exp(t_k)$ and its gradient in MATALB. In `logsumexp.m`, I have implemented it in the naïve fashion that has the overflow/underflow problem that we discussed in the class. Please modify it to make it numerically stable. Here is some code to help you verify your result:

```
% Test your logsumexp function
% (currently my naive implementation has overflow/underflow problems):
%*** if your code is correct, it should return 1.0017e+03
logsumexp([1000,1001.2, 999.9])

%*** if your code is correct, it should return -999.6867
logsumexp([-1000,-1001])
```

(b) **[5 Points]** Implement `LogisticReg_negloglikelihood.m` which calculates the *negative* of the log-likelihood of multi-class logistic regression and its gradient. One useful way to debug your code is to test it on simulated data; please check `LogisticReg_debug.m` for details. Please include the figures generated by `LogisticReg_debug.m` in your PDF report.

(c) **[5 Points]** Implement `LogisticReg_predict.m` to make prediction for testing data. It should return both the "hard label" (the prediction $hatYtest$), as well as the "soft label" (the prediction probability $p(y|x)$).

(d) **[5 Points]** Implement `LogisticReg_GD.m` which minimizes the negative log-likelihood by gradient descent with constant step size μ. Set a reasonable step size so that the gradient descent algorithm decreases the objective function monotonically on the simulated data generated by `LogisticReg_debug.m`. Please report the step size that you recommend and show the trajectory plot of your algorithm, that is, plot $(t, f(\theta_t))$ where $f(\theta_t)$ is the value of our objective function at the t-th iteration.

(e) **[5 Points]** The naive gradient descent method requires step size tuning and can be slow in practice. Limited-memory BFGS (L-BFGS) is a much more efficient optimization algorithm and is widely used in practice. You can find a MATLAB implementation of L-BFGS `minFunc` from http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html. Please download the package (you do not need to include the download files in your homework submission) and learn how to use it; for the purpose of this homework, you do not need to actually understand how L-BFGS works. Please complete `LogisticReg_LBFGS.m` which uses minFunc to minimize the negative log-likelihood.

(f) **[5 Points]** We now test your logistic regression code on the Wine dataset available at `https://archive.ics.uci.edu/ml/datasets/Wine`, which uses chemical analysis to determine the origin of wines. Please load `wine.mat`, in which I have split the data into training and testing; again, I have held out $Ytest$, and instead provide a function `cal_err_Ytest_wine` that evaluates the prediction error rate for a given $hatYtest$, that is, it calculates $mean(Ytest \sim= hatYtest)$. Because the features are not in the same scale, you should normalize the features:

```
load wine.mat
X = [Xtrain; Xtest];
mu = mean(X,1); sigma = std(X,[],1);
Xtrain = (Xtrain - repmat(mu,size(Xtrain,1),1)) ./ repmat(sigma, size(Xtrain,1),1);
Xtest = (Xtest - repmat(mu,size(Xtest,1),1)) ./ repmat(sigma, size(Xtest,1),1);

hatYtest = randi(10,size(Xtest,1),1); % predicting by random guess
testerror  = cal_err_Ytest_wine(hatYtest); % evaluate the testing error
```

Please run logistic regression on this dataset using `LogisticReg_LBFGS.m` and `LogisticReg_predict.m` that you just implemented. Use cross validation to select the best regularization coefficient α in $[0.1, .5, 1, 5]$. Show your code and report the α you pick and the testing error returned by `cal_err_Ytest_wine`.

Problem 3: SVM

Consider a toy dataset with one features and three data points

$$(x^{(1)}, y^{(1)}) = (-1, \ +1)$$
$$(x^{(2)}, y^{(2)}) = (0, \ +1)$$
$$(x^{(3)}, y^{(3)}) = (1, \ -1).$$

In this case, the linear predictor is $\hat{y} = \text{sign}(wx + b)$, where w, b, x are all scalars.

(a) **[2 Points]** Is this dataset linearly separable? Why?

(b) **[5 Points]** Write down the primal form of linear SVM for finding the optimal $[w, b]$ that maximizes the geometric margin.

(c) **[5 Points]** Introduce the dual (or slack) variable α using the Lagrangian multiplier method, and derive the dual form of SVM in this case. In addition, represent the optimal w using the dual variable α.

(d) **[2 Points]** Which points are the support vectors in this case? Explain why.

(e) **[5 Points]** Assume we use kernel SVM with kernel $K(x, x')$ instead, write down the corresponding optimization problem for finding the optimal α and describe how we make predictions for testing data based on α and the support vectors (Hint: you need to rewrite both wx and b in terms of the kernel function $K(x, x')$).