CSCI567 Fall 2013 - Assignment 2
Due date: Oct. 4 , 2013

September 20, 2013

Algorithm Component

1 Linear regression

In this section, we study a few interesting properties of linear regression.

Review In the lectures, we have described the least mean square (LMS) solution for linear regression $y = \sum_d w_d x_d$:

$$\boldsymbol{w}^{\text{LMS}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{1}$$

where $\boldsymbol{X}$ is our design matrix (N rows, D columns) for the training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ and $\boldsymbol{y}$ is the N-dimensional column vector of the true values in the training data set. (Note that, for convenience, we have assumed the input has been appended with the constant 1 so we do not have to explicitly introduce w_0.)

To invert the matrix $\boldsymbol{X}^T\boldsymbol{X}$, the computational complexity is $O(D^3)$. On the other hand, if $D = 1$, the training data set $\mathcal{D}$ would be $\{(x_n, y_n)\}_{n=1}^N$, and the solution is simply

$$w^{\text{LMS}} = \frac{\sum_n x_n y_n}{\sum_n {x_n}^2} \tag{2}$$

Note that the computation is quite straightforward and no matrix inversion is involved.

1.1 Question 1 (5 points)

Let us consider a special way of doing linear regression for $D > 1$. Instead of using the input $\boldsymbol{x}_n$ in eq.(1), we are to use each dimension of $\boldsymbol{x}_n$ separately in eq. (2). Namely,

- For each dimension d, we (imaginarily) construct a new training dataset $\mathcal{D}_d = \{(x_{nd}, y_n)\}_{n=1}^N$.
- With $\mathcal{D}_d$, we use the one-dimensional input to predict y_n. The LMS solution is thus computed according to eq. (2). Let us call the resulting parameter v_d:

$$v_d = \frac{\sum_n x_{nd} y_n}{\sum_n x_{nd}^2}$$

- We combine all v_d together to form a parameter vector $\boldsymbol{v}$.

Note that every step above is very simple — we definitely do not have to compute $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ and invert it, as in eq. (1).

However, the questions is: is $\boldsymbol{v}$ going to be the same as $\boldsymbol{w}^{\text{LMS}}$ as computed by eq. (1)?

Unfortunately, the two are *not* the same in the most general case. Nonetheless, the two are the same if the columns of $\boldsymbol{X}$ are orthogonal:

$$\sum_n x_{nd} x_{nd'} = 0, \quad \forall \quad d \neq d'$$

This condition sometimes can be referred as "the dimensions d and d' are uncorrelated". Prove that under the condition, $\boldsymbol{w}^{\text{LMS}}$ and $\boldsymbol{v}$ are indeed the same.

1.2 Question 2. (15 points)

In real-world data sets, features are rarely uncorrelated. Thus, the "trick" in Question 1 does not apply. However, we can transform the data such that features become uncorrelated. In the following, we will build towards that goal.

2.1 (2 points) Consider univariate linear regression $y = wx$. For a training data set $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$, after computing the LMS solution w^{LMS}, we obtain the residual error

$$e_n = y_n - w^{\text{LMS}} x_n$$

Prove that e_n and x_n are uncorrelated, i.e., $\sum_n e_n x_n = 0$. (Note this result has a very nice intuition: there is no more "information" we could extract from x in order to reduce the residual error further.)

2.2 (5 points) Suppose $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ is our design matrix. Let $\boldsymbol{u}_d \in \mathbb{R}^N$ denote $\boldsymbol{X}$'s d-th column. We will transformer $\boldsymbol{u}_d$ into $\boldsymbol{z}_d$ such that those dimensions become uncorrelated. We define our transformation in the following recursive way:

- $\boldsymbol{z}_1 = \boldsymbol{u}_1$. Namely, the first dimension is unchanged.
- The d-th $\boldsymbol{z}_d$ is given by

$$\boldsymbol{z}_d = \boldsymbol{u}_d - \sum_{d'=1}^{d} \gamma_{dd'} \boldsymbol{z}_{d'}$$

  where the coefficient $\gamma_{dd'}$ is the parameter of the following univariate problem: $\boldsymbol{z}_{d'}$ is the input and $\boldsymbol{u}_d$ is the target output.

Let us pause a second and think about what $\boldsymbol{z}_d$ is. First, let us say $d = 2$ and we will be looking at $\boldsymbol{z}_2$, which is given by

$$\boldsymbol{z}_2 = \boldsymbol{u}_2 - \gamma_{21} \boldsymbol{z}_1$$

Namely, $\boldsymbol{z}_2$ is the *residual error* of using $\boldsymbol{z}_1$ to predict $\boldsymbol{u}_2$! Thus, by the statement in Question 2.1, this error $\boldsymbol{z}_2$ is uncorrelated with $\boldsymbol{z}_1$.

Then what is $\boldsymbol{z}_3$? By the definition, we have

$$\boldsymbol{z}_3 = \boldsymbol{u}_3 - \gamma_{31} \boldsymbol{z}_1 - \gamma_{32} \boldsymbol{z}_2$$

Now, we already know $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are uncorrelated. Then, by the statement in Question 1, we know that we can use $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ to predict $\boldsymbol{u}_3$ by using $\boldsymbol{z}_1$ to predict $\boldsymbol{u}_3$ and using $\boldsymbol{z}_2$ to predict $\boldsymbol{u}_3$ *separately*, which give rise to γ_{31} and γ_{32} (Please check the definition of $\gamma_{dd'}$ in the above).

In other words $\boldsymbol{z}_3$ is the residual error for using $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ to predict $\boldsymbol{u}_3$. Once we have this observation, we are ready to prove some interesting results. Please prove

- $\boldsymbol{z}_3$ and $\boldsymbol{z}_1$ are uncorrelated, so are $\boldsymbol{z}_3$ and $\boldsymbol{z}_2$.
- Extending our argument, (show rigorously that) all pairs of $\boldsymbol{z}_d$ and $\boldsymbol{z}_{d'}$ are uncorrelated, as along as $1 \leq d \neq d' \leq D$.

2.3 (3 points) We are almost there! Now, let us look at $\boldsymbol{z}_D$, which is given by

$$\boldsymbol{z}_D = \boldsymbol{u}_D - \sum_{d'=1}^{D-1} \gamma_{Dd'} \boldsymbol{z}_{d'}$$

We will use $\boldsymbol{z}_D$ to predict $\boldsymbol{y}$. This is a univariate linear regression and we can compute the parameter as

$$\beta_D = \frac{\sum_n z_{Dn} y_n}{\sum_n z_{Dn}^2}$$

where z_{Dn} is the n-th element of $\boldsymbol{z}_D$.

Prove β_D is the same as w_D^{LMS}, the D-th element of $\boldsymbol{w}^{\text{LMS}}$. In other words, while in Question 2.2, we have been using univariate regressions to transform the data to $\boldsymbol{z}_d$ and now use univariate regression to get β_D, we arrive at the same solution as if we have used eq. (1).

2.4 (5 points) Note that in Question 2.3, we only state how to compute w_D^{LMS} using β_D, corresponding to the D-th dimension x_D. What if we want w_d^{LMS} for other dimensions $d \neq D$?

The strategy is surprisingly simple. Suppose we have 3 dimensions x_1, x_2, and x_3. Following the procedure outlined in Question 2.3, we will get β_3 (i.e., w_3^{LMS}). To get w_1^{LMS}, we use the same procedure but work in the following sequence: x_2, x_3, and x_1.

Namely, we "shuffle" the dimensions of the data so that the first dimension is now the last dimension. It is not difficult to see that, for this shuffled data, the new β_3 will be the same as w_1^{LMS}, corresponding to the original data. We can then use the same procedure to get w_2^{LMS}.

To summarize, our new method of doing linear regression when $D > 1$ is to use univariate regression many times — in Question 2.3 to transform the data, in Question 2.4 to get the parameter for the last dimension, and in the above to get the parameter for every dimension.

Please use the big O-notation to analyze the computational complexity of this strategy. The advantage of using univariate regression is to avoid the matrix inversion in eq. (1). What is the computational complexity of the new method? Is it $O(D^\alpha)$ with $\alpha < 3$, thus, more efficient than directly computing $\boldsymbol{w}^{\text{LMS}}$? If not, (i.e., for the new method α is 3 or greater than 3), then what is the advantage of this new method?

1.3 Question 3 (10 points)

We now look at how to overcome the computational difficulty in large-scale linear progression from another angle. Suppose our D is very large. However, let us say we are more interested in finding a subset of dimensions (i.e., features) that can predict well.

Why that makes sense? First, if the subset is small, then making prediction can be less costly because we do not have to use a very big parameter vector $\boldsymbol{w} \in \mathbb{R}^D$. This is very important if you need to make prediction fast (for example, to make decision in a split of second). Second, using a small number of features can curb overfitting. Third, a model using a small number of features is easier to examine and to interpret.

The tricky part is, of course, how do we know which subset of features we should use? Suppose we constrain ourselves to consider only $K \ll D$ features. There are $\left(\begin{array}{c} D \\ K \end{array}\right)$ different subsets! Obviously, enumerating each one of them and checking which subset of features will predict well is not practical.

Perhaps we should relax a bit. Instead of looking for the *best* subset of features, we could be happy if we could find a reasonably good solution. The simplest solution would be randomly choose K features from all D of them.

Can you design a strategy that does better than this random strategy? Please present your strategy with your arguments. You do not have to rigorously prove how well your strategy beats the random one or how close your strategy gets close to the best subset. Both your TAs and instructor are qualified to judge whether your strategy is reasonable. So please be creative!

Hint. You might want to think about what "predict well" means. For example, if you need to choose a subset of features to predict, you probably want to choose them such that the residual sum square gets reduced as much as possible. Thus, you might want to start with a model with only one feature, and then increase to two features, etc. Then, your decision problem would be which feature should you add and why adding that feature is a good idea?

2 Logistic Regression

2.1 Question 4. (3 points)

We have shown that for two Gaussian distributions, if their covariances are the same, then the decision boundary between them is linear. This linear decision boundary can be estimated discriminatively — it leads to logistic regression.

We did not derive how to estimate the parameters of the Gaussians generatively, though. You are going to do that. Specifically, suppose we have two classes C_1 and C_2. Each class has a class conditional probability

$$p(x|y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu_y)^2}{2\sigma^2}}$$

Note that y takes the value of 1 or 2, corresponding to C_1 and C_2 respectively. In the class conditional probability, note that while the means μ_y are different, the covariance is the same.

For a training data set $\mathcal{D} = \{(\boldsymbol{x}_n y_n)\}_{n=1}^N$, please use the generative approach to estimate μ_1, μ_2 and σ.

2.2 Question 5. (5 points)

We have described multinomial logistic regression

$$p(C_k|\boldsymbol{x}; \boldsymbol{w}_k) = \frac{e^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}}}{\sum_{k'} e^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}}}$$

Prove that this model is invariant under a constant shift to the parameter vectors

$$\boldsymbol{v}_k \leftarrow \boldsymbol{w}_k + \boldsymbol{c}, \quad \forall \;\; k$$

where $\boldsymbol{c}$ does not depend on k. Under this shift, the posterior probabilities do not change

$$p(C_k|\boldsymbol{x}; \boldsymbol{w}_k) = p(C_k|\boldsymbol{x}; \boldsymbol{v}_k), \quad \forall \;\; k$$

This means the conditional likelihood solution (i.e., the parameter vectors) for multinomial logistic regression is *not* unique. Shifting parameters by a constant will not change the likelihood value.

Thus, please present a strategy (of constraining parameters) such that the solution is unique.

2.3 Question 6. (10 points)

Prove that the cross-entropy error function for multinomial logistic regression is convex by showing the Hessian matrix is positive semidefinite.

Submission instruction

Your answers must be typeset with LaTeXand generated as a PDF file. No handwritten submission will be permitted. A template is provided for you. Please consult it for additional instructions. There are many free integrated LaTeXeditors that are convenient to use, please google search them and choose the one(s) you like the most. This `http://www.andy-roberts.net/writing/latex` seems like a good tutorial.

Due Date and Time

3:15pm, Friday, Oct. 4, 2013. You need to logon to Blackboard (for the time being) to submit your solutions. Your PDF file should be named as abcd_hw2_AB.pdf where abcd is the last 4 digits of your USC ID and AB are the initials of your first and last names

A single two-day extension or two one-day extensions are allowed. You need to apply for it by Thursday Oct 3 2013, 3:15pm and get my approval. Please send your application to me via email. Late submissions will not be accepted if not previously approved. My email address is `feisha@usc.edu`

Collaboration

You may collaborate. However, you need to write your own solution and submit separately. You also need to list with whom you have discussed. For any team, at most 4 members can participate. You cannot participate more than 1 team.

Programming Component

3 Digit Classification using K-Nearest Neighbors

You are required to implement a K-Nearest Neighbor (KNN) classifier in Matlab and use it to classify images of digits to their correct class labels. Your classifier should support different distance functions.

Data

The file *'USPS-split1.mat'* contains a randomly partitioned and noised up version of the USPS dataset into training, development and test sets. Once you load the .mat file (using Matlab's 'load' command) your Matlab environment should contain two variables - X and y. The training, development and test sets are stored as $D\times$*(number of samples)* matrices in $X.train$, $X.devel$ and $X.test$ with $D = 256$. The corresponding digit labels are similarly stored in y.

Displaying the digits: If you'd like to take a look at the digit grayscale image, you may convert a data point (column) to 16×16 matrix and display it using image display function - for example, if the target digit is stored in $3rd$ column of $X.devel$, it can be displayed in Matlab using the command

$$imshow(double(reshape(X.devel(:,3), 16, 16)));$$

Implementation

You should implement the KNN algorithm yourselves. You are NOT allowed use any matlab toolboxes (e.g., *knnclassify*, *knnsearch*, or what not).

Your implementation should support different distance functions - for example, for the report you should use the distance function $distance(u, v, p) = ||u - v||_p$ that computes the L_p norm of the difference between two vectors. Additionally, you should experiment with other general distance metrics (e.g., Jaccard, Hamming, etc).

Optional: Extra credit will be given to student groups that come up with an "interesting" distance function and a proof that it is indeed a distance function. An "interesting" distance function is one that was neither taught in class (or used in this homework) nor commonly used in the literature.

Experiment

You are asked to compare the training/development/test classification accuracy rates for $K = 1, 3, \ldots, K_{max} = 19$ and for various L_p norm based distances with $p = [0.5, 1, 1.5, 2, 3, \infty]$ (note that $p = 2$ is called the Euclidean distance and $p = 1$ is known as the city-block, or Manhattan distance).

For each K and p, run your KNN classifier to classify $X.train$, $X.devel$ and $X.test$ (always use $X.train$ as the training set for the KNN classifier). You should obtain label predictions for each of the sets. To calculate the accuracy for training/development/test, compute the average number of correctly classified data points - for example, if the predicted labels for the development set are stored in *pred.devel*, the accuracy can be computed in Matlab using

$$mean(pred.devel == y.devel)$$

Notes:

1. Use the data as is. You are NOT allowed to preprocess the data in any way.
2. When computing the labels for the **training** set you must use the leave-one-out strategy - Specifically, when classifying a data point from $X.train$, it should be regarded as a development set, and the rest should be used as a training set.

3. When selecting the optimal K to maximize classification accuracy, pay attention that you should tune K based on **development data set** rather than training data set. Also, **never** tune K on the test data set.

What to submit for KNN

3.1 Reports (21 points)

Question 7 (10 points) For each $p = [0.5, 1, 1.5, 2, 3, \infty]$ report the following:

1. An accuracy table as below for $K = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]$ (you may use http://truben.no/latex/table/ or better yet, have your code generate the latex code for the table).

K	1	3	5	7	9	...	15	17	19
Training Acc (%)						...			
Development Acc(%)						...			
Test Acc(%)						...			

2. The K that maximizes the training set accuracy and the K that maximizes the development set accuracy.

Question 8 (2 points) Report the pair of (p, K) that maximize training accuracy and the (p, K) pair that maximizes development accuracy.

Question 9 (2 points) Explain why the leave-one-out strategy is used for the training set.

Question 10 (2 points) Explain why K should be tuned based on the development set rather than training set and why K should never be tuned on the test set.

Question 11 (5 points) **Extra credit/Optional**: If you used an 'interesting' distance function, write down:

1. The distance function form and a proof that it is indeed a distance function (or give reference to one).
2. The training/development/test accuracy results you obtained (they don't have to beat your best results for the best L_p norm)
3. A matlab function run_1_KNN_interesting(K) that executes your code with your distance function.

3.2 Code(5 points)

Your .m code files - submit your KNN implementation. Your code should be runnable using a **function** named *run_1_KNN.m* that has the following interface:

pred=run_1_KNN(K,p,set)

Where 'set' is one of three strings *'train'* or *'devel'* or *'test'*; and 'p' is the L_p norm to be used. The output *pred* is the predicted labels (a column vector) for the set you specified in 'set'.

3.3 Predictions

- A single .mat file named *predictionKNN.mat* containing the predicted labels for training/development/test sets using the best K and p on the development set. These should be stored in variables *predTrain*, *predDevel* and *predTest* (using Matlab's *save* command).

4 Linear Regression

The UCI "Wine Quaity" dataset (wine.mat) lists 11 chemical measurements of roughly 5000 wine samples as well as an overall quality per sample, as determined by wine connoisseurs. Your goal is to model wine quality using the linear regression model. Since linear regression is quite easy to implement (a few lines of code in matlab), we shall consider two different settings for experimentation.

As a reminder, in linear regression we model the target variable y as $y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x_1 + ... + w_D x_D = w^T(1; x)$ and proceed to find the best $\boldsymbol{w}$ in the least squares sense:

$$\boldsymbol{w}^{\text{LMS}} = \min_{\boldsymbol{w}} \frac{1}{2} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2$$

(Restricted) Feature Design

Linear regression is a linear model which means that w, the hypothesis linear regression finds, lies in the feature space of the dataset. When there are few features, it may be the case that the hypothesis space (the set of all possible hypothesis) is too restrictive.

One way to mitigate this problem is to non-linearly map the original (small) feature space to a large one. Namely, we can map each D-dimensional datapoint $\boldsymbol{x}$ to a M dimensional datapoint $\phi(\boldsymbol{x})$ where the mapping function $\phi(\cdot)$ is non-linear in its input $\boldsymbol{x}$ and such that $D < M$.

For example, a simple quadratic mapping appends all features of the form $x_i \cdot x_j$ for $i, j \in \{1, \ldots, D\}$ yielding:

$$\phi(\boldsymbol{x}) = (1, x_1, \ldots, x_D, x_1^2, x_1 x_2, \ldots x_D^2)$$

We can easily extend this to cubic mapping by further appending features of the form $x_i x_j x_k$ and so on for general polynomial mapping of higher order. *Note that we should remove redundant features. For example, we need only one of $x_1 x_2$ and $x_2 x_1$.*

Generally, recall the geometric interpretation of the least squares solution - it is the projection of the labels $\boldsymbol{y}$ onto the subspace spanned by the design matrix's columns. By adding linearly independent features we are increasing the rank of this subspace and thus the projection can become closer to $\boldsymbol{y}$. This may however result in overfitting.

Regularization

The motivation behind regularization is to introduce prior assumptions on the output model $\boldsymbol{w}$. This may help control the model complexity and prevent overfitting with respect to the training data. For example, one possible assumption is that most entries of $\boldsymbol{w}$ should be exactly 0 (corresponding to question 3 in the theory part) and another is that no entry in w should be too large in absolute value. A complete introduction and discussion of regularization will be presented in class.

For now, we will focus on L_2 regularization. In this case we change the optimization problem to:

$$\boldsymbol{w}^{\lambda} = \min_{\boldsymbol{w}} \frac{1}{2} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 + \frac{\lambda}{2} ||\boldsymbol{w}||_2^2$$

where $\lambda \geq 0$ is referred to as a *hyper-parameter* used to control the complexity of the resulting model. When $\lambda = 0$, the model reduces to the usual (unregularized) linear regression. For $\lambda > 0$ the objective function balances between two terms: (1) The data-dependent quadratic loss function $||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2$, and (2) A function of the model parameters only $||\boldsymbol{w}||_2^2$. Intuitively, for larger λ the new objective function is less sensitive to bad estimations of the target y.

Implementation and Experiment

You should:

1. Find the closed form solution for $\boldsymbol{w}^{\lambda}$ - For a fixed λ the objective function admits a closed form solution.

2. Write a matlab function that computes $\boldsymbol{w}^{\lambda}$:

$$w = RLS(X, y, lambda)$$

   Here RLS stands for regularized least squares.

3. Use K-fold cross validation on the training set to determine the best $\lambda \in \{0, 0.5, 1, 1.5, \ldots, 10\}$. That is,

   (a) Randomly split the training data into K disjoint parts, each about $1/K$ of the original size.

   (b) For each λ,

      i. Train the model on $K - 1$ parts and use the resulting model to classify the remaining part. Record the classification accuracy on that part.

      ii. Compute the average classification accuracy rate over the K folds.

   (c) Determine the accuracy maximizing λ, denoted λ^{CV}.

What to submit for Linear Regression

4.1 Report (20 points)

Question 12 Feature Design (8 points) For $\lambda = 1$, report the training and testing accuracy using linear (i.e., original), linear + quadratic and linear + quadratic + cubic features. Among the 3, which feature set is the best ?

Question 13 Regularization (8 points) Using 10-fold cross validation and linear features, report the accuracy maximizing $\lambda \in [0 : 0.5 : 10]$, as well as plot $||\boldsymbol{w}^{\lambda}||_2^2$ and $||\boldsymbol{X}^{train}\boldsymbol{w}^{\lambda} - \boldsymbol{y}||_2^2$.

Question 14 Analyze the role of the regularizer in the model (4 points)

4.2 Code (5 points)

-Your .m code files - submit your linear regression implementation. Your code should be runnable using a **function** named *run_2_LR.m* which has no input arguments (It simply loads the data and uses $\lambda = 1$) and a single output variable *pred* with the predicted values for the test set.

Blackboard Submission Instructions

You are required to submit both runnable Matlab code files, result mats (.mat) and a formatted report (PDF) via the blackboard system.

Upload a single zip archive file *containing a single folder*. **The name of the zip file should be your abcd_hw2_AB.zip** where abcd is the last 4 digits of your USC ID and AB is your initials. **The folder should be named as results**. If you are working in a group, only one student from the group should upload the files. Please use the uploading student's ID number for the folder name.

The folder should contain all your .m files and a single PDF report named *HW2Progmming.pdf.* The report should contain the names and USC student IDs of all the group members. It should answer all programming questions — make you mark clearly each question you are answering. Since you will have plots, you will need to refer to the plots (such as "Please see Fig. 1" — LaTeX does not place plots next to texts so you need to number the plots and refer that way.)

Due Date and Time

The programming assignment is due on blackboard the same way as the algorithm component.

Collaboration

For the coding part of the assignment, you may collaborate in groups of up to 4. Each group needs to submit to blackboard only once. Group members share the same credits. Changing group membership throughout the semester is strongly discouraged.