CS273a Homework #1

Introduction to Machine Learning: Fall 2012

Due: Friday October 12th, 2012

Write neatly (or type) and show all your work!

This may be your first homework using MATLAB ; please see the course webpage for links to tutorials to help you start using it, or the recitation lecture from the first week. In matlab, "help <functionname>" is often a good idea; e.g., "help plot", "help find"...

For MATLAB reports such as this one, I recommend that you use Word or OpenOffice to create a document with your answers, export the finished report as a PDF file, and upload it to EEE. In Windows, you can import MATLAB figures into your document directly using copy/paste operations, or you can export a figure from MATLAB directly to pdf using e.g. "print -dpdf <filename>". You can also use this to create and import JPEG or PNG files, but if you do so, please ensure that they are of sufficient resolution to be clear in the resulting document ("-r<value>" controls the resolution).

I prefer to receive a *single* electronic document when possible; for some homeworks you may find it more convenient to do parts by hand, in which case a hard copy of that portion, or of the entire thing, is fine. Please *do not* upload a ZIP file containing multiple figures, as this is difficult to interpret without associated comments.

Problem 1: Data Exploration

In this problem, we will explore some basic statistics and visualizations of an example data set. First, download and load the "iris" data set into Matlab:

```
iris=load('data/iris.txt');     % load the text file
y = iris(:,end);           % target value is last column
X = iris(:,1:end-1);       % features are other columns
whos                       % show current variables in memory and sizes
```

The Iris data consist of four real-valued features used to predict which of three types of iris flower was measured (a three-class classification problem).

(a) For each feature, compute the mean and variance of the data values

(b) For each feature, plot a histogram ("`hist`") of the data values

(c) For each pair of features (1,2), (1,3), and (1,4), plot a scatterplot ("`plot`") of the feature values, colored according to their target value (class). (For example, plot all data points with $y = 0$ as blue, $y = 1$ as green, etc.) You may find the commands "`find`" and "`hold on`" useful for this.

Problem 2: K-Nearest Neighbors and Validation

In this problem, you will continue to use the Iris data and explore a KNN classifier using provided code. First, shuffle and split the data into training and test subsets:

```
iris=load('data/iris.txt'); y=iris(:,end); X=iris(:,1:end-1);
[X y] = reorderData(X,y);                  % shuffle data randomly
[Xtr Xte Ytr Yte] = splitData(X,y, .75);  % split data into 75/25 train/test
```

Class Objects We will use Matlab classes to implement our learner methods. Matlab classes are a bit annoying to use, particularly the "old style" that are compatible with Octave. However, the usefulness outweighs the flaws.

An old-style class is created using a directory preceded by . For example, included in the code is a kNN classifier, `knnClassify`. The methods associated with this class are the Matlab `.m` files located within it. The constructor is `knnClassify`; all the other functions are called by providing a `knnClassify` object as the first argument. (That tells Matlab / Octave where to look for the function.) So, you can build and "train" a kNN classifier on `Xtr,Ytr` and make predictions on some data `Xte` with it using e.g.,

```
knn = knnClassify( Xtr, Ytr, K );  % replace or set K to some integer
YteHat = predict( knn, Xte );      % make predictions on Xtest
```

If your data are 2D, you can visualize a data set and a classifier's decision regions using e.g.,

```
plotClassify2D( knn, Xtr, Ytr );  % make 2D classification plot with data (Xtr,Ytr)
```

(a) Modify the above code to use only the first two features of X, and visualize (plot) the classification boundary for varying values of $K = [1, 5, 10, 50]$.

(b) Again using only the first two features, compute the error rate on both the training and test data as a function of $K = [1, 2, 5, 10, 50, 100, 200]$. You can do this most easily with a for-loop:

```
K=[1,2,5,10,50,100,200];
for k=1:length(K)
  learner = knnClassify(...    % train model
  Yhat = predict(...           % predict results on training data
  err(k) = ...                 % count what fraction of predictions are wrong
end;
figure; plot(...               % average and plot results
```

Plot the resulting error rate functions using a semi-log plot ("`semilogx`"), with training error in red and test error in green. Based on these plots, what value of K would you recommend?

(c) Repeat your error rate computation using all four features of X, and comment on any changes.

(d) Many algorithms, including kNN, are sensitive to the *scale* or magnitude of the feature values. Often one tries to normalize for this by rescaling or otherwise transforming the data first. Repeat the first part of this question (visualizing the boundary) after first (1) rescaling the data to unit variance (see the provided "`rescale`" function) and (2) "whitening" or "sphereing" the data (see Wikipedia and the provided "`whiten`" function). Best practice is to estimate the transform on the training data and then apply it to the test data, since this mimics not having the test data available when the model is constructed, e.g.,

```
[Xtr,T] = rescale(Xtr);   % learn scaling transform T on train data
Xte=rescale(Xte,T);       % apply same transform to test data
```

Comment on any changes you notice in the classifier boundaries or results.

Problem 3: Linear Regression

For this problem we will explore linear regression and the creation of additional features.

(a) Load the "`data/mcycleTrain.txt`" and "`data/mcycleTest.txt`" training and test sets, respectively. The first column `data(:,1)` are the target (y) values; the second column `data(:,2)` is the scalar feature x for each example. Plot a scatter plot of the data; you should be able to discern the implicit functional relationship between x and y.

(b) Use the provided `linearRegress` class to create a linear regression predictor of y given x. You can plot the resulting function by simply evaluating the model at a large number of x values, `xs`:

```
lr = linearRegress( Xtr, Ytr );  % create and train model
xs = [0:.01:2]';                 % densely sample possible x-values
ys = predict( lr, xs );          % make predictions at xs
```

Plot the training data along with your learned prediction function in a single plot. Also calculate and report the mean squared error in your prediction at the training and test data.

(c) Try fitting $y = f(x)$ using a polynomial function $f(x)$ of increasing order. Do this by the trick of adding additional polynomial features before constructing and training the linear regression object. You can do this easily yourself; you can add a "constant" feature and a quadratic feature of Xtr with

```
Xtr2 = [ones(size(Xtr,1),1), Xtr, Xtr.^2];
```

A function "`fpoly`" is also provided to more easily create such features:

```
XtrP = fpoly(Xtr, degree);       % create polynomial features up to given degree
[XtrP, T] = rescale(XtrP);       % often a good idea to scale the features
lr = linearRegress( XtrP, Ytr ); % create and train model
% ...
```

Train models of degree $d = 3, 5, 7, 10, 18$ and (1) plot their learned prediction function $f(x)$ and (2) their training and test errors. For (1), don't forget to also expand and scale the features of `xs` using `fpoly` and `rescale`. For (2), plot the resulting training and test errors as a function of polynomial degree.

(d) (Extra credit) The `linearRegress` class can also take a regularization parameter such as I discussed in class. Experiment with this parameter and comment on the results.