

Problem Set 1: Density Estimation, Regression, and Classification

Due: 5PM Friday, Oct. 10, 2008

In this problem set, you will use FASTlib, a C++ library for machine learning algorithm development, to implement several algorithmically straightforward methods for the general problems of density estimation, regression, and classification.

You will also compare different methods for each problem by analyzing their results on challenging, real-world datasets. The datasets and some necessary MATLAB code can be found [here](#). For any problems that ask for optimal bandwidths, you need only provide a bandwidth that is near-optimal; we do not expect for you to differentiate the objective functions with respect to the bandwidth to find the precise optimum, though if you are able to do so successfully we may award extra credit.

Also, for the kernel methods, you should scale the dimensions as necessary such that they all have the same variance. This way, for 2 dimensional data the bandwidth for each dimension is the same, and so using isotropic (i.e., radially symmetric) kernels is justified. For the histogram, the data should be scaled such that each dimension has the same range; this way square bins make sense.

For all of the requested density estimate and prediction files, please write one estimate per line and ensure the order of the predictions is the same as the order of the test points. Most (probably all) of the programs will accept command-line parameters, and we let you know precisely what to call those. You should use the FASTlib `fx_param` functions for handling your parameters (the tutorial explains this). It's incredibly important to stick with the naming conventions for the executable names, parameters, and output files, as this will let us efficiently test your code to make sure it produces the correct output.

For your submission, create a directory with the same name as your GT user account. Put into this directory all of the files necessary to build your code using FASTlib (including `build.py`, we will compile and run your code!), along with a document with answers to the questions and the requested figures. Compress the directory using tar or zip, name the archive with the same name as your GT username and the appropriate file extension, and email it to the TA at `niche@cc.gatech.edu`. The subject line should be "CSE6740 - PS1" without the quotes.

1 Install and test FASTlib

a) Take a look at the [FASTlib tutorial](#), which should provide a clear introduction to using the FASTlib library for coding machine learning methods in C++. For a somewhat more in-depth look, you can check out the [documentation](#). Note that MLPACK (with the exception of a SVM) and PHYSPACK are not present in this release of FASTlib, but they will be made available to you in time for the project.

b) Install FASTlib and compile the provided example in `examples/platonic_allnn`. Post the output of the program. All of the code below must be implemented in FASTlib, so be sure to let us know well in advance of the deadline if you encounter any problems using the library.

2 Density Estimation, 10 points

The first part of this problem set explores density estimation. We have provided some synthetic data generated from a mixture of Gaussians (MoG) model where each Gaussian produces points $x_i \in \mathbb{R}^2$. The data points are in `datasets/synth/X.txt` and the true densities at those points are in `datasets/synth/true_density.txt`. In `code/synth.m`, you will find a MATLAB script that draws points from a mixture of 2 Gaussians distribution. You might find this code useful to expedite initial testing of your density estimators, perhaps by generating a small number of points from simple distributions.

a) Using `code/hist2.m`, a MATLAB function for producing 2-dimensional histogram plots, plot a histogram of the data for a bin-width that undersmooths the data, a bin-width that oversmooths the data, and a bin-width which is optimal according to minimization of the least-squares cross-validation objective

$$CV_{L_2}(h) = \int \hat{f}_h^2 - 2 \frac{1}{N} \sum_{i=1}^N \hat{f}_h^{-i}(x_i).$$

You need not submit any code you write for histogram estimation and loss measures.

b) Implement a program that estimates a mixture of Gaussians model via expectation maximization (EM). The first part of your implementation should be k -means; the results of this clustering can be used to initialize the parameters for each distribution. The second part of the implementation is EM using those initial parameter estimates.

You should estimate full covariance matrices, rather than diagonal covariance matrices. The covariance matrix update in the maximization step is a natural generalization of the update rule presented in class; it is shown in Equation (9.25) of Pattern Recognition and Machine Learning. Name the executable `mog_density`, and hence in `build.py`, the `binrule's name` field should be set equal to “`mog_density`”.

Your program should accept a parameter `data_filename` for the data, a parameter `true_density_filename` (to evaluate L_2 loss) for the true densities file, and a parameter `k` (for k Gaussians). It should output to two files:

`mog_model.txt` - the estimated parameters (π_i, μ_i, Σ_i) for each $1 \leq i \leq k$

`mog_density_estimates.txt` - your density estimates for the test points

What are the estimated model parameters (π_i, μ_i, Σ_i) for $1 \leq i \leq k$ for three choices of k ? Also, what are the model parameters for k^* , the k that minimizes the L_2 loss between the points' estimated and true densities?

c) Implement kernel density estimation (KDE) using the Epanechnikov kernel. Name the executable `kde`.

Your program should accept a parameter `data_filename` for the data, a parameter `true_density_filename`, for the true densities file, and a parameter `h` for the bandwidth. The true densities need not be used by your program, though we recommend that accept them as a parameter so that you can calculate the L_2 loss within your program. The program should output to one file:

`kde_density_estimates.txt` - your density estimates for the test points

Provide the L^2 loss and the leave-one-out cross-validation (LOOCV) likelihood for four bandwidths including: one that undersmooths, one that oversmooths, the bandwidth that minimizes the LOOCV negative log-likelihood (i.e., maximizes the likelihood) of all of the points \mathbf{X} . Note that we are not asking for the mean integrated squared error (MISE), which is the expected risk under L_2 loss, but rather that you calculate L_2 loss using the provided true densities.

You may find the below information useful, along with the fact that the Epanechnikov kernel is implemented in FASTlib (the EpanKernel class), which will take care of kernel evaluations and normalization constants for you, and hence save you a bit of work.

In class we went over the Epanechnikov kernel for univariate data. Recall that for $x = \frac{x_i - x_j}{h}$ this formula is

$$K(x) = \frac{3}{4}(1 - x^2)I(x^2 \leq 1)$$

This can be generalized to higher dimensions, such that for points in \mathbb{R}^n and $\mathbf{x} = \frac{\mathbf{x}_i - \mathbf{x}_j}{h}$, the Epanechnikov kernel is

$$K(\mathbf{x}) = \frac{1}{V_s(n)} (1 - \|\mathbf{x}\|^2) I(\|\mathbf{x}\|^2 < 1)$$

where $V_s(n)$ is the volume of a unit-radius $(n - 1)$ -sphere. Note that a $(n - 1)$ -sphere lives in a n -dimensional space, and hence a 1-sphere corresponds to a circle.

Also, recall that the LOOCV likelihood minimizes

$$CV_i(h) = -\frac{1}{N} \sum_{i=1}^N \log \hat{f}_h^{-i}(x_i).$$

3 Regression, 10 points

The Nadaraya-Watson estimator is a nonparametric regression method that naturally extends from KDE. Nadaraya-Watson regression (NWR) can be used for highly nonlinear regression problems for which linear regression may not be suitable.

We have provided a training and test astronomy dataset with 4 features to be used for prediction. These files are at `datasets/redshift/refined_astroset_train.ds` and `datasets/redshift/refined_astroset_test.ds`. Each of the features is derived from 4 spectral features representing the power at a particular band of the electromagnetic spectrum. Of particular interest to astronomers is to predict the redshift of objects in the sky at which a telescope is pointed. The redshift values are in `datasets/redshift/alldata_zs_train` and `datasets/redshift/alldata_zs_test`. Your task is to estimate the redshift via Nadaraya-Watson regression and linear regression by regressing on the 4 features in the `refined_astroset` data files.

a) Implement the Nadaraya-Watson estimator using the Epanechnikov kernel. This should only require a minor tweak to your KDE code. Name the executable `nwr`.

Your program should accept the parameters `train_data_filename`, `test_data_filename`, `train_labels_filename`, `test_labels_filename`, and a bandwidth parameter `h`. It should output to one file:

`nwr_predictions.txt` - your target predictions for the test points

Choose the optimal bandwidth on the training set by minimizing the L_2 LOOCV objective

$$CV_{L_2}(h) = \frac{1}{n} \sum_{i=1}^N (y_i - \hat{r}_h^{-i}(x_i))^2.$$

Without too much effort, it should be clear how you can get around the problem of estimating separate \hat{r}_h^{-i} for each i , arriving at a $O(N)$ improvement in computation time.

b) For the problem of predicting redshift, compare the L_2 error from NWR to the L_2 error achieved by linear regression (you need not implement linear regression unless you cannot find an implementation), for both the training and test set.

c) Choose two predictor variables, and plot your test set predictions of redshift against these two variables together. So you should have one predictor variable along the x axis, one along the y axis, and the corresponding redshift prediction along the z axis. MATLAB would be one easy way to produce such a plot.

d) *Extra Credit* - Use a statistical test to see whether NWR predicts redshift significantly better than linear regression at the $\alpha = 0.1$ significance level.

4 Classification, 10 points

For the last stellar part of this problem set, you will identify quasars in the sky. The first 4 features are similar to the features from the redshift dataset. The last feature is binary and indicates whether or not the object is a quasar. A training and test set for the predictor variables are provided in the files `datasets/quasars/qstraincolors.txt` and `datasets/quasars/qstestcolors.txt` respectively, and the training labels and test labels are provided in the files `datasets/quasars/qstrainlabels.txt` and `datasets/quasars/qstestlabels.txt`.

All of the classification programs should accept the file parameters accepted by your NWR implementation, so your program should accept the parameters `train_data_filename`, `test_data_filename`, `train_labels_filename`, and `test_labels_filename`. All error reported for the following classifiers should be zero-one loss on the test set after training the classifiers on the training set.

a) Implement the naive Bayes classifier (see page 359 of All of Statistics for the key equations for this classifier). Name the executable `naivebayes`.

Your program should output to one file:

`naivebayes_predictions.txt` - your class predictions for the test points

What is the test error?

b) Implement a 1-nearest-neighbor classifier. Name the executable `1nn`.

Your program should output to one file:

`1nn_predictions.txt` - your class predictions for test points

What is the test error?

c) Implement the nonparametric Bayes classifier (NBC). Since NBC is quite similar to KDE, you could largely recycle your KDE code here. Maintain a separate bandwidth for each class. Name the executable `nonparametricbayes`.

Your program should accept the two additional parameters `h1` and `h2`. It should output to one file:

`nonparametricbayes_predictions.txt` - your class predictions for the test points

What are the test errors for multiple decent choices of the pair of class-bandwidths (h_1, h_2) ?

d) Implement a mixture of Gaussians classifier, representing the points of each class with a mixture of k Gaussians (you may use the same k for each class). You should be able to use your MoG estimation code from earlier to estimate each class-conditional MoG model. Name the executable `mog_classifier`.

Your program should accept the additional parameter `k`, and it should output to one file:

`mog_predictions.txt` - your class predictions for the test points

For 3 sensible choices of k , what test errors do you get?

e) Download SVM^{light} from <http://svmlight.joachims.org> and train a support vector machine (SVM) with a Gaussian kernel on the optical character recognition data set in `datasets/ocr`. We have removed all but 2 classes, to make the digit classification problem for a 2-class SVM. To train the classifier:

```
svm_learn -t 2 -g 0.01 -c 10 datasets/ocr/optdigits_train.txt
```

and to test the SVM on the test data

```
svm_classify datasets/ocr/optdigits_test.txt svm_model
```

Try to maximize the accuracy (reported by `svm_classify`) on the test set, by trying different parameter settings of the kernel bandwidth σ (via parameter `-g`) and the regularization parameter C (via parameter `-c`). We recommend that you use the provided parameter settings $[\sigma, C] = [0.01, 10]$ as a starting point. What is the best setting of (σ, C) that you found, and what is the corresponding accuracy on the test set for that setting?

f) **Extra Credit** - Use a statistical test to see whether the nonparametric Bayes classifier performs significantly better than the nearest-neighbor classifier at the $\alpha = 0.1$ significance level.

5 Extra Credit: Improve Sequential Minimal Optimization, 0-100 points

[Note: The grading for the extra credit is fairly subjective. The max of 100 points roughly equates to a publishable paper. You are free to the extra credit assignment up until the end of the semester.]

Hack the working set selection heuristic (how pairs of points are chosen) for Sequential Minimal Optimization to see if you can make it faster. If you don't know what SMO is, you can find out all about it [here](#). Your code must provide the same answer as SMO, and you will be awarded more points for an algorithm that is constructively creative or is faster than SMO.