

CS195-5: Introduction to Machine Learning
Fall 2006

Problem Set #4 : Grape

Out: October 27, 2006

Due: November 10, 2006, 11am EDT

Instructions

In order to identify problem sets to the automated submission system we will assign code names to each problem set. This one will be called “grape”.

How and what to submit? Please submit your solutions using the automated system in the following way.

1. Create a directory (say, `ps4`) in which you will work on your solutions. All files mentioned below are assumed to reside in that directory.
2. Typeset the written parts of your solution in \LaTeX . The final results should be a document in PDF (preferable) or in PostScript. please name this document `grapesolution.pdf` or `grapesolution.ps`, according to the format. **Important: please start the solution of every problem with `\newpage`.**
3. Create Matlab code (extension `.m`) and data (`.mat`) files needed.
4. Create a file named `README`, and include in it a short description of all Matlab files you are submitting.
5. **From the directory `ps4`**, run
`/course/cs195-5/bin/cs195-5handin grape`

You can resubmit your work as many times as needed, and each subsequent version will override the previous one.

Late submissions: there will be a penalty of 25 points for any solution submitted past the deadline until 11am on Monday, November 13. No submissions will be accepted past then.

What is the required level of detail? When asked to derive something, please clearly state the assumptions, if any, and strive for balance: justify any non-obvious steps, but try to avoid superfluous explanations. When asked to plot something, please include the figure as well as the code used to plot it; if multiple entities appear on a plot, make sure that they are clearly distinguishable (by color or style of lines and markers). When asked to provide a brief explanation or description, try to make your answers concise, but do not omit anything you believe is important.

1 Support Vector Machines

In this problem we will train Support Vector Machines with radial basis function (Gaussian) kernels. We have provided Matlab code with implementation (with a few missing pieces you will need to fill in). The code is available on the course website and also on the course file system, `/course/cs195-5/matlab/svm/`.

The following Matlab functions are provided; use `help` and read the documentation in the files to understand the details of the implementation.

`trainSVM.m` train an SVM given a labeled data set (represented as Matlab `struct`, a regularization parameter C , the kernel function (as a function name or a Matlab function handle) and a single parameter value to be passed to the kernel.

`testSVM.m` test a trained SVM on a given data set, and also produce a plot illustrating the results (assumes 2D data).

`plotSVM.m` plot the support vectors and decision boundary of an SVM (assumes 2D data).

`svmDiscriminant.m` compute the value of the discriminant function associated with a trained SVM on a given set of examples. Returns a real number, the sign of which can be used to classify those examples.

Recall that to construct the SVM we need to solve the *dual problem*

$$\operatorname{argmax}_{\alpha} \left\{ \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \right\} \quad (1)$$

$$\text{subject to } \sum_{i=1}^N \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C \text{ for all } i = 1, \dots, N. \quad (2)$$

The solution for this dual quadratic program is found in `trainSVM.m` using Matlab's own `quadprog` routine, which requires a particular, slightly different formulation of the quadratic program. Specifically, the call `alpha = quadprog(H,f,A,b,Aeq,beq,l,u,x0);` will find the solution to the following problem:

$$\operatorname{argmin}_{\alpha} \frac{1}{2} \alpha^T \mathbf{H} \alpha + \mathbf{f}^T \alpha, \quad (3)$$

$$\text{such that } \mathbf{A} \alpha \leq \mathbf{b}, \quad (4)$$

$$\mathbf{A} \text{eq} \alpha = \mathbf{b} \text{eq}, \quad (5)$$

$$\text{and } \mathbf{l} \leq \alpha \leq \mathbf{u}. \quad (6)$$

The equalities and inequalities between vectors are interpreted element-wise, e.g., $\mathbf{l} \leq \alpha \leq \mathbf{u}$ means that for every i , $l_i \leq \alpha_i \leq u_i$. Passing an empty array `[]` for any of the arguments will tell `quadprog` to ignore the corresponding constraint. For instance, you can see that in `trainSVM.m` we ignore the inequality constraints given by `A` and `b`. The last argument `x0` specifies the recommended starting guess for the optimizer.

Problem 1 [15 points]

Write the following Matlab code:

- `Krbf.m` with a function which takes as arguments two sets of examples X_N and Z_M (as column vectors) and the RBF (Gaussian) kernel width σ , and returns the kernel matrix. That is, if $X_N = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ and $Z_M = [\mathbf{z}_1, \dots, \mathbf{z}_M]$, the call `K=Krbf(X,Z,d)` returns an $N \times M$ matrix `K` such that

$$K_{ij} = \exp \left(-\frac{1}{\sigma^2} \|\mathbf{x}_i - \mathbf{z}_j\|^2 \right).$$

- Fill in the two missing pieces in `trainSVM.m`: the calculation of `H` and the calculation of `w0`.

Try to avoid loops and express as much of the computation as possible in vector-matrix notation (which Matlab is most efficient at).

End of problem 1

Advice: In order to complete `trainSVM.m` you will need to figure out how to convert the formulation in (1) to the form of (3), and to extend the solution of Problem 4 in PS Apricot to the kernel case. Note that the calculations necessary to find w_0 should be carried out in the feature space.

For the next problem you will need the data provided in `dataSVM.mat`. The file contains three structures representing data sets: `dataTrain`, `dataTest` and `dataTest2`, all obtained from the same distribution.

Problem 2 [15 points]

Train an SVM on `dataTrain` with the RBF kernel for the following values of σ : 0.1, 1, 2, 5, 10 and 50 (use $C=10$). Use `testSVM.m` to evaluate the resulting SVM on the training and two test sets, and plot the errors as a function of σ , as well as the decision boundaries obtained with each of the six SVMs. Briefly explain the behavior apparent in these results.

End of problem 2

2 Non-parametric methods: locally weighted regression

In this section we will look at a non-parametric method for regression: locally weighted regression (LWR). This method proceeds as following:

1. Let $X_N = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ be the training data, with corresponding observed values of the target function y_1, \dots, y_N .
2. For the given test input \mathbf{x}_0 , we find k nearest neighbors of \mathbf{x}_0 in X_N . Without loss of generality, let these be $\mathbf{x}_1, \dots, \mathbf{x}_k$.
3. Compute *normalized weights* w_1, \dots, w_k using a kernel function K :

$$w_l = \frac{K(\mathbf{x}_0, \mathbf{x}_l)}{\sum_{m=1}^k K(\mathbf{x}_0, \mathbf{x}_m)}. \quad (7)$$

4. Augment inputs as necessary for the desired parametric model (e.g., add the constant unit feature for linear model, add quadratic terms for a quadratic model etc.). Fit a parametric regression model to the neighbors weighted by w_l , yielding estimated $\hat{\mathbf{w}}$.
5. Predict $\hat{y}_0 = \hat{\mathbf{w}}^T \mathbf{x}_0$.

In the case of a simple linear regression, the fitting step can be done by the familiar least squares method, except that the data (both inputs and outputs) are weighted. Namely, if \mathbf{X} would be the unweighted design matrix for the relevant inputs, LWR replaces it with \mathbf{X}' where the i -th row of \mathbf{X} has been multiplied by w_i . Similarly, if \mathbf{y} is the vector of corresponding unweighted outputs, let \mathbf{y}' be the vector $[w_1 y_1, \dots, w_k y_k]^T$. Then, LWR fit of \mathbf{w} is

$$\hat{\mathbf{w}} = (\mathbf{X}'^T \mathbf{X}')^{-1} \mathbf{X}'^T \mathbf{y}'.$$

Problem 3 [5 points]

Suppose our feature space has dimension d , and $d \geq k$. Let \mathbf{X}' be the (weighted) $k \times d$ design matrix for the nearest neighbors. The matrix $\mathbf{X}'^T \mathbf{X}'$ is not invertible; thus, we can't get meaningful unique least-squares solution by applying the standard technique using pseudoinverse. How do you suggest solving this?

End of problem 3

Advice: Think of this as an overfitting problem, where we don't have enough data to fit even the linear coefficients; how have we approached this kind of problems in the class?..

We have provided a function `knnLinear.m` which implements simple k -NN regression. That is a more naive local regression scheme, which fits a model to the neighbors without weighting them. We will use this as a baseline, and its code will serve as a starting point for implementing LWR.

Problem 4 [20 points]

Write a function `lwrLinear.m` that implements locally-weighted regression in 1D. This function will fit a linear model to the weighted neighbors of a text point \mathbf{x}_0 ; the weights will be assigned using a Gaussian kernel of fixed width, $\sigma = 0.2$. The function signature (i.e., the input and output arguments) should be identical to that of `knnLinear` except that `lwrLinear` will take

σ as one additional argument. Document the modifications and additions relative to `knnLinear`.

Using the training and test data in file `lwrData.mat` (names of the variables are self-explanatory), test LWR and k -NN on the test data for the following values of k : 1, 5, 30, 100. For each k , plot the observed values of y for the test data and the fit obtained with LWR and k -NN regression models, and report the mean squared error of prediction for both models. Briefly summarize the trends you see in these results. In particular, please address the following questions: how does the behavior of the fit change with k for each of the models? What can you say about the advantages and disadvantages of each model?

End of problem 4

3 The EM algorithm

In this part of the problem set we will derive and implement the EM algorithm for a discrete space, in which the observations are d -dimensional binary vectors (containing either 0 or 1). We will model distributions in this space as mixture of multivariate Bernoulli distributions. That is,

$$p(\mathbf{x} | \theta, \mathbf{p}) = \sum_{l=1}^k p_l p(\mathbf{x} | \theta_l), \quad (8)$$

where $\theta = [\theta_1, \dots, \theta_l]$ are the parameters of the k components, and the mixing probabilities $\mathbf{p} = [p_1, \dots, p_k]^T$ are subject to $\sum_l p_l = 1$.

The l -th component of the mixture is parametrized by a d -dimensional vector $\theta_l = [\theta_{l1}, \dots, \theta_{ld}]^T$. The value of θ_{lj} is the probability of 1 in the j -th coordinate in a vector drawn from this distribution. Since the dimensions are assumed to be independent, given θ_l , the conditional distribution of \mathbf{x} under this component is

$$p(\mathbf{x} | \theta_l) = \prod_{j=1}^d p(x_j | \theta_{lj}) = \prod_{j=1}^d \theta_{lj}^{x_j} (1 - \theta_{lj})^{1-x_j}. \quad (9)$$

The hidden variables here are, just in the Gaussian mixture case, the identities of the component that generated each observation. We will denote the hidden variable associated with \mathbf{x}_i by z_i . The EM with the model proceeds

as follows. The E-step consists of computing the posterior of $p(z_i | \mathbf{x}_i; \theta, \mathbf{p})$. In the M-step, we need to update the estimates of θ and \mathbf{p} . These updates can be derived in a way similar to the derivation for the Gaussian mixture. They are¹:

$$\theta_c^{(t+1)} = \frac{1}{N_c^{(t)}} \sum_{i=1}^N \mathbf{x}_i p(z_i = c | \mathbf{x}_i; \theta^{(t)}, \mathbf{p}^{(t)}), \quad (10)$$

$$p_c^{(t+1)} = \frac{N_c^{(t)}}{N}. \quad (11)$$

where N_c^t is the weighted “mass” of the c -th component,

$$N_c^{(t)} = \sum_{i=1}^N p(z_i = c | \mathbf{x}_i; \theta^{(t)}, \mathbf{p}^{(t)}),$$

and the notation $\theta_c^{(t)}, p_c^{(t)}$ stands for the values of the relevant parameters in the end of iteration t .

This model could be applicable, for instance, in modeling distributions of documents using binary features like the ones we looked at in Problem Set Nectarine. Here, however, we will apply it to another domain—images of handwritten digits. We will use the data in `digitDataBinary`; it contains images of digits 1 through 4, that have been cropped to 20×20 pixels and pixel values normalized to be 0 or 1 (by thresholding the original pixels). The variable names in that file should be self-explanatory.

We have provided a skeleton for an implementation of EM with this model; you will need to complete this code. It consists of the following functions; you will find all of them useful. The relevant files are available on the Web and also on the course file system,

`/course/cs195-5/matlab/emDiscrete`

`runem.m` the function that runs the EM iterations.

`estep.m`, `mstep.m` the E-step and the M-step implementation.

`conditional.m` computes $p(\mathbf{x}_i | \theta_l)$ for each i, l .

`logLikelihood.m` computes the conditional probabilities and the log-likelihood of the data under a given values of mixture parameters.

¹See PRML Section 9.3.3 for the derivation.

Problem 5 [25 points]

Write the exact expression for the posterior probability $p(z_i = c | \mathbf{x}_i; \theta, \mathbf{p})$ in terms of \mathbf{x}_i and the elements of θ and \mathbf{p} . Implement it in `estep.m`. Also, fill in the missing code for the component parameter updates in `mstep.m`.

Run your code on `digitTrain` with $k = 12$ components, using the call

```
[theta,p,logL]=runem(digitTrain,12,1);
```

This will set stopping criterion to min. change of 1 in log-likelihood. Once the EM has converged, we can visualize the parameters θ :

```
figure;colormap gray;
for k=1:12
    subplot(3,4,k); imagesc(reshape(theta(:,k),20,20));axis image;
end
```

Turn in the plot obtained by the code above, and your interpretation of the components; what properties of the data do they capture?

End of problem 5

Advice: Running EM will take a while, even with an efficient implementation; typically, it should converge after about 100 iterations. In practice, since the model found by EM depends on random initialization (first guess of θ), we would run EM a number of times, and take the results of the run that lead to the highest log-likelihood. You are welcome to do that, but it is OK to just run EM once and report the results of that single run.

In addition to exploring the data in an unsupervised fashion (done in the problem above) we can use the mixture model for classification. While above we arbitrarily decided to use twelve components in the mixture, a more principled way is to select the model based on an information criterion that balances the likelihood and model complexity. Here we will use the Bayesian Information Criterion (BIC). BIC for a model parametrized by θ on a data set X_N

$$BIC(\theta, X_N) = \log p(X_N; \theta) - \frac{\pi(\theta)}{2} \log N,$$

where $\pi(\theta)$ is the number of free parameters in the model. For instance, mixture of two unconstrained Gaussians in d -dimensional space has $d(d + 1) + 2d + 1$ parameters: $d(d + 1)/2$ for each covariance matrix, d for each

mean, and 1 for the priors (only one and not two since $p(1) = 1 - p(2)$). This criterion can be seen as a form of regularization; we will select the model with the highest value of BIC, instead of the model with the highest likelihood.

Problem 6 [20 points]

Write a Matlab function that performs BIC model selection with Bernoulli mixture EM. Using this function, apply the EM algorithm separately to the training data for each digit class from 1 to 4. Let $\theta_c = \{\theta_{c1}, \dots, \theta_{ck}\}$ and $\mathbf{p}_c = \{p_{c1}, \dots, p_{ck}\}$ be the estimated mixture parameters for class c , where k is the number of mixture components selected based on BIC. Turn in the plot of θ s (similar to the plot in the previous problem) for each class-specific mixture.

Now write down the expression of the Bayes optimal classifier, under the assumption that the estimated mixture model is accurate (and assuming equal prior 1/4 for each digit class), in terms of a test input \mathbf{x} and θ_c, \mathbf{p}_c . Implement this classifier in Matlab. In addition, construct a “baseline” classifier that models each class-conditional as a single Bernoulli distribution (estimate its parameters using the closed-form ML solution for Bernoulli). Report the test error obtained with these two classifiers on `digitTest`, and briefly explain the results.

End of problem 6

Advice: You should be able to largely reuse the code written for the previous problem.