10-701/15-781, Machine Learning: Homework 5

Eric Xing, Tom Mitchell, Aarti Singh
Carnegie Mellon University
Updated on March 24, 2010

- The assignment is due at 10:30am (beginning of class) on **Mon, April 26, 2010**.
- Separate you answers into three parts, one for each TA, and put them into 3 piles at the table in front of the class. Don't forget to put both your name and a TA's name on each part.

1 SVMs [Amr, 30 + 10 extra credits points]

1.1 Kernels and Feature Maps

Given the following dataset in 1-d space (Figure 1), which consists of 3 positive data points $\{-1, 0, 1\}$ and 3 negative data points $\{-3, -2, 2\}$.

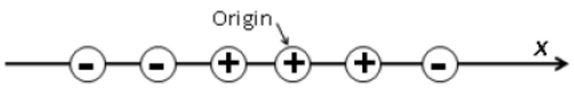


Figure 1: Dataset

(1) **(3 pts)** Find a feature map($\{\mathbf{R}^1 \rightarrow \mathbf{R}^2\}$), which will map the data in the ordinal 1-d *input space* (x) to a 2-d *feature space* (y_1, y_2) so that the data becomes linearly-separable. Plot the dataset after mapping in 2-d space.

(2) **(5 pts)** Write down the decision boundary $w_2y_2 + w_1y_1 + w_0$ given by hard-margin linear SVM in the feature space. Draw this decision boundary on your plot and mark the corresponding support vector(s).

(3) **(5 pts)** What is the equivalent decision boundary in the original input space? Draw this decision boundary over the points in Figure 1.

(4) **(3 pts)** For the feature map you choose, what is the corresponding kernel $K(x_1, x_2)$?

(5) **(4 pts)** What is the maximum number of points in the input space that can be shattered by an SVM classifier using the kernel in (4)? Explain in one or two sentences.

1.2 SVM and other Classifiers

(1) The idea of mapping the data from the input space to another feature-space in which the data becomes linearly separable was used earlier in the class by another classifier.

  (a) **(2 pts)** What is the name of this classifier?

  (b) **(3 pts)** List one difference between the way this idea was used by SVM and this classifier. Discuss the implication of this difference on both classifiers (with regard to training and/or testing, etc.).

(2) The final decision rule of an SVM classifer using kernel K is given by:

$$y * (z) = \text{sign}\Big(\sum_{i \in \text{SV}} \alpha_i y_i K(x_i, z) + b \Big) \quad (1)$$

In this case K can be interpreted as a similarity metric.

(a) **(2 pts)** We have studied one classifier whose decision rule looks similar to (1), what is the name of this classifier?

(b) **(3 pts)** List one difference between the classifier in (a) and SVM with regard to their decision rules and discuss the implication of this difference on both classifiers (again with regard to training and/or testing, etc).

1.3 EXTRA CREDIT: Dimension of Transformed Feature Space

One popular choice for the kernel, K, in SVM is the dth degree polynomial:

$$K(\mathbf{x}, \mathbf{z}) = (1 + \langle \mathbf{x}, \mathbf{z} \rangle)^d$$

The 2nd degree polynomial kernel corresponds to the inner product of a transformed feature space. If $\mathbf{x}$ and $\mathbf{z}$ are both 2-dimensional vectors in the input space , then the dimension of the transformed feature space is 6:

$$\begin{aligned} k(\mathbf{x}, \mathbf{z}) &= (1 + \langle \mathbf{x}, \mathbf{z} \rangle)^2 \\ &= (1 + x_1 z_1 + x_2 z_2)^2 \\ &= 1 + 2x_1 z_1 + 2x_2 z_2 + x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= \langle [1 \ \sqrt{2}x_1 \ \sqrt{2}x_2 \ \ x_1^2 \ \sqrt{2}x_1 x_2 \ \ x_2^2], [1 \ \sqrt{2}z_1 \ \sqrt{2}z_2 \ \ z_1^2 \ \sqrt{2}z_1 z_2 \ \ z_2^2] \rangle \\ &= \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle \end{aligned}$$

(1) **(5 pts)** Let's generalize this example. Assume $\mathbf{x}$ and $\mathbf{z}$ are p-dimensional, what is the dimension of the transformed feature space for the dth degree polynomial kernel? **Note**: you don't have to explicitly list the form of each dimension as in the example above, I am just looking for the dimensionality. (**Hint:** This is a combination with replacement problem — brush up on your combinatorics background)

(2) **(5 pts)** According to (1), calculate the dimension of the transformed feature space for a 15-dimensional $\mathbf{x}$ using the 3rd degree polynomial kernel. Also for this specific setting, compute the ratio between the number of multiplications required to evaluate the kernel in the input space vs. evaluating the dot-product in the feature space.

2 PCA Tutorial [25 pts, Field Cady]

Principal Component Analysis (PCA) is one of the most popular dimensionality reduction techniques, partly because it is very simple to understand and implement.

Geometrically, if your data lies "mostly" on a certain hyperplane, PCA just projects your data onto this hyperplane. If your data does indeed lie mostly on a hyperplane, this technique preserves most of the variation between your points; in applications where a single data point contains more numbers than you have data points, your data will lie *exactly* on a hyperplane. If your data is not very hyperplanar, there may still be good dimensionality reduction available, but the techniques are more complicated, and much less common.

Computationally, the principal components of a dataset end up being just the eigenvectors of the covariance matrix of your data. The associated eigenvalues measures how much the data varies along a direction. If most of the variation is in only a few eigenvalues, than your data lies almost completely on the hyperplane spanned by their eigenvectors. Because covariance matrices are positive semi-definite, all eigenvalues are non-negative.

This problem walks you through a simple application of PCA to an interesting dataset. I'm providing patchy sample code for a bare-bones analysis - feel free to make it better. I will give extra credit, at my discretion, for cool improvements. No need to submit your actual code for this homework.

2.1 Loading the data

The data we use are pictures of a hand holding a rice bowl and rotating it. They are available at http://vasc.ri.cmu.edu/idb/images/motion/hand/. Though the images are very large and intricate, there is only one underlying degree of freedom; the orientation of the hand.

Download the images and, since there's a lot of data, take a sparse subsample of them to use as your data.

```
disp('loading data');
n=481;  % number of images
subsample=8;    %use every 8th pixel in a row/column
fx = 1:subsample:512;
fy = 1:subsample:480;
px=length(fx);  %new image sizes
py=length(fy);
data=zeros(py,px,n);  % cube to store all your data
for i = 1:n
    img = imread(sprintf('hand.seq%d.png',i));
    data(:,:,i)=img(fy,fx);
end
```

In order to take eigenvectors we need to make our individual data points into vectors, rather than 2d matrices. We also need to subtract out the mean.

```
disp('vectorizing and centering data');
X = zeros(py*px,n);  % each column will be a data point
for i = 1:n
    X(:,i) = reshape(data(:,:,i),py*px,1);
end
averageVec = mean(X,2);  % each element is the mean of a row of X
for i = 1:n
    X(:,i) = X(:,i)-averageVec;
end
```

2.2 Basic PCA [9 pts]

Now construct the covariance matrix of the data.

```
disp('finding eigenvalues and eigenvectors');
Cov = X * X';
[evecs,D] =  eig(Cov);
evals = diag(D);    % make it a vector
```

Our data has only one underlying degree of freedom, but how flat is the data? Plot the eigenvalues of the covariance matrix and discuss your conclusions.

```
plot(evals);
```

If appropriate, provide a plot of a subset of the eigenvalues to make your results more clear.

2.3 Basic application [8 pts]

We said before that our data has one inherent degree of freedom; and since the hand in the pictures turns almsot all the way around, you would think the data should actually form a closed loop, writhing around in a very high dimensional space. Using PCA to reduce the data to 2 dimensions corresponds to projecting this loop onto the plane that makes it look "flattest". Do this, and comment on your results.

```
disp('doing dimensionality reduction');
principalComps = evecs';
reducedX = principalComps * X;
scatter(reducedX(3840,:),reducedX(3839,:));
```

2.4 Reconstructing Your Data [8 pts]

We can use the principal component to visualize not just the shape of our dataset, but the ways in which the individual points vary. Recall that we only processed a subsample of each image in the original dataset; display the first of these subsampled images. Display the average subsampled image and the principal eigenImage. Now reconstruct the first subsampled image from the average image and the first three eigenimages. How well did our reconstruction work?

```
image1 = reshape(averageVec+X(:,1),py,px);
averageImage = reshape(averageVec,py,px);
eigenImage3840 = reshape(evecs(:,3840),py,px);
eigenImage3839 = reshape(evecs(:,3839),py,px);
eigenImage3838 = reshape(evecs(:,3838),py,px);
reconstructed = averageImage + ...
                reducedX(3840,1)*eigenImage3840 + ...
                reducedX(3839,1)*eigenImage3839 + ...
                reducedX(3838,1)*eigenImage3838;
imagesc(image1);
imagesc(eigenImage3840);
imagesc(reconstructed);
```

2.5 Extra Credit [8 pts]

Do some type of additional analysis on the data using PCA. Present your results as a figure with a brief description of what I'm looking at.

3 AdaBoost [Ni, 30 pt]

Given N examples (x_i, y_i), where y_i is the label and $y_i = +1$ or $y_i = -1$. Let $\mathrm{I}(\cdot)$ be the indicator function, which is 1 if the condition in () is true and 0 otherwise. In this exercise, we use the following version for AdaBoost algorithm:

1. Initialize $w_i^1 = 1/N$ $(i = 1, ..., N)$
2. For $t = 1, ..., T$,
   a. Learn a weak classifier $h_t(x)$ by minimizing the weighed error function J_t, where $J_t = \sum_{i=1}^N w_i^t I(h_t(x_i) \neq y_i)$;
   b. Compute the error rate for the learnt weak classifier $h_t(x)$: $\epsilon_t = \sum_{i=1}^N w_i^t \mathrm{I}(h_t(x_i) \neq y_i)$;
   c. Compute the weight for $h_t(x)$: $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$;
   d. Update the weight for each example: $w_i^{t+1} = \frac{w_i^t \exp\{-\alpha_t y_i h_t(x_i)\}}{Z_t}$, where Z_t is the normalization factor for w_i^{t+1}: $Z_t = \sum_{i=1}^N w_i^t \exp\{-\alpha_t y_i h_t(x_i)\}$.
3. Output the final classifier: $H(x) = sign(f_T(x))$, where $f_T(x)$ is a linear combination of the weak classifiers, i.e., $f_t(x) = \sum_{m=1}^t \alpha_m h_m(x)$.

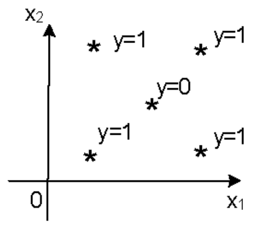
3.1 Sequential Optimization [5 pts]

In class, we learnt that AdaBoost tries to minimize the negative exponential loss: $E = \sum_{i=1}^N \exp\{-y_i f_T(x_i)\}$ sequentially. That is to say, at the t^{th} $(1 \leq t \leq T)$iteration, we want to choose appropriate weight α_t and the corresponding weak classifier $h_t(x)$ so that the overall loss E (accumulated up to t^{th} iteration) is minimized. It was proved that this strategy leads to the update rule: in the t^{th} iteration, $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$.

Now, if we change the objective function to square loss $E = \sum_{i=1}^N (y_i - f_T(x_i))^2$ and we still want to optimize it sequentially. What is the new update rule for α_t?

3.2 [5 pts]

As shown in the figure below, we have five training samples with label 0.0 or 1.0.



Now let's assume that the base functions $h_t(x)$ are linear classifiers. Will the boosting algorithm (with square loss) always get zero training error after sufficient iterations? What is the minimum number of iterations before it can reach zero training error?

3.3 About Margin[5 pts]

Draw the objective functions of SVM, logistic regression, and Adaboost together. Assume we have a single training sample and a single feature.

hints: for AdaBoost assume that $h_t(x)$ is given, and the parameter is α_t.

3.4 [5 pts]

What does “margin” mean? Do logistic regression and Adaboost have margins?

3.5 Overfitting [5 pts]

There is an interesting applet written by Yoav Freund (`http://cseweb.ucsd.edu/~yfreund/adaboost/`). With it, you can create your own data set and train AdaBoost models.

Please design a dataset showing that AdaBoost does overfit. You can print a screen shot which includes both data points and error curves.

3.6 [5 pts]

Can you think of a strategy to prevent Boosting from overfitting?