

CS 294 - Practical Machine Learning - Homework 6

This is the last homework and will be due on bSpace on Friday, May 9. Please pick one of the three problems to do. Email Charles Sutton for questions about the first problem, Kurt Miller for the second, and Alex Shyr for the third.

Problem 1: Anomaly Detection using PCA In this part of the assignment, you are provided with the Abilene data set of network traffic (contained in the file `abilene.tar`). The data consists of four 1008×41 matrices which record the amount of traffic in the Abilene network in 4 consecutive weeks. Each row of a matrix (in the files `WeekX.dat`) records the amount of traffic that go through the 41 links of the network in 10-minute intervals during a period of one week time. In addition, there are files (`WeekXlabel.dat`) that contain the label of the network traffic at each interval (1 if normal, -1 if abnormal). As such, a data point X_t represents the traffic at the t -th interval.

We shall use the data for the first week to learn a simple model for the “normal” behavior of network traffic, and then test for anomalies on the data set of the second week, updating the model using the data of week 2, and test on week 3, and so on. The basic assumption of our approach is that normally the data lie in a low-dimensional subspace detectable by the PCA method. Thus, the abnormal traffic can be detected by looking at the deviation of each data point from that subspace.

To review from the lecture, the detection procedure works in three steps:

Step 1 (PCA): Compute the top k eigenvectors of the covariance matrix using PCA. Call the resulting eigenvectors v_1, \dots, v_k and eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_k$. Construct the matrix V of top eigenvectors $V = (v_1 v_2 \dots v_k)$, so that each column of V is one eigenvector.

Step 2 (Computing the test score): For each test data point X_t represented as a row vector, compute the distance from X_t to the subspace spanned by the top k eigenvectors as $S(X_t) = DD^T$, where $D = X_t(I - VV^T)$.

Step 3 (Thresholding): Let α be a desired false alarm rate. Raise an alarm whenever $S(X_t) > Q_\alpha$. The threshold Q_α is computed as follows:

$$Q_\alpha = \phi_1 \left[\frac{c_\alpha \sqrt{2\phi_2 h_0^2}}{\phi_1} + 1 + \frac{\phi_2 h_0 (h_0 - 1)}{\phi_1^2} \right]^{\frac{1}{h_0}}$$

where c_α is the $(1 - \alpha)$ -percentile of the standard normal distribution (i.e., $P(N(0; 1) > c_\alpha) = \alpha$) and $h_0 = 1 - \frac{2\phi_1\phi_3}{3\phi_2^2}$, and $\phi_p = \sum_{j=k+1}^{41} \lambda_j^p$ for $p = 1, 2, 3$. If you were wondering, this threshold is derived from theoretically analyzing the distribution of $S(X_t)$ for normal traffic.

Complete the following.

1. Implement the procedure described above on the Abilene data. Choose the number k of principal components to represent the subspace of normal traffic data. Explain your choice.
2. For different choices of the false alarm rate α , compute the threshold Q_α accordingly, and test the detection procedure on the test set. Plot the false alarm rate and compare the actually measured false alarm with the desired α . Also plot the ROC curve.
3. The choice of threshold above is very specific to the statistic derived from the PCA method. One can also use the following choice of threshold, which is more generally applicable: Based on the training data set, e.g., the data from the first week, estimate the mean μ and standard deviation σ of the statistic $S(X)$. Then on the second week, given a desired false alarm rate α , raise an alarm whenever $S(X) > \mu + \sigma c_\alpha$. Repeat the same set of experiment as in part (2). Does this perform better than the threshold Q_α from the last part?
4. Periodically updating the model of normal behavior can help reduce both the false alarm and misdetection rate. An *offline* learning approach is one in which the model is trained only once and then run on all the test data. For example, you might perform PCA only on the week 1 data, and the resulting eigenvectors are used for anomaly detection in weeks 2, 3, and 4. In an *online* approach, on the other hand, the model is retrained on new data as it arrives. For example, here you would run PCA on the week 1 data, test on week 2, then run PCA on weeks 1 and 2 together, test on week 3, and so on. Compare the performance of online and offline learning on this data.

Problem 2: clustering with Dirichlet processes You will find a full implementation of the CRP Gibbs sampler for the DP in `dp_gibbs_sampler.r` along with several very simple data sets. The point of this assignment is to have you understand the code and get a feel for working with the DP. Remember that using a Gibbs sampler is only one method of doing inference in the DP. We will show how in some cases this is a good method, but we will also expose some of its weaknesses.

1. Your first task is to copy `dp_gibbs_sampler.r` into `finite_gibbs_sampler.r` and modify the code to work in the parametric (fixed, finite numbers K of clusters) setting. This should require very few lines of change. In order to do this, you will need to look over the code and understand the basics of what is going on. Do not worry about efficiency in your implementation (if we were worried about efficiency, we would not be doing this in R).

We will use the same prior on the cluster-specific parameters, but now we will need to specify an exact number of clusters before running the sampler. Fortunately, for this assignment, we are only running this on toy data sets, so you can verify how many clusters there are (if there is any doubt, the number in the file name indicates how

many clusters were used to generate the data). Try running your finite sampler with both correct and incorrect numbers of clusters to see how it performs. Several of the data sets are nearly identical with the only differences being the scale or the variances of the points in the clusters.

One big difference in the code will be that now you will need to decide how to initialize your clusters. Remember to initialize the cluster assignments z as well as the cluster specific means and covariances. For the demo in class, a completely random initialization of z was used and then the mean and covariance for each cluster were seeded with the mean and covariance of the points assigned to that cluster. In practice, this is a very poor method to initialize the sampler. A better approach would be to seed the assignments using k-means or a GMM.

2. Now run the DP sampler on the same data sets. Note how the DP often (though not always) converges to the correct number of clusters. Pick a few of the data sets and plot the number of clusters per sample over time for 5-10 different runs of the sampler (due to the randomization, no two runs will be identical). A common practice is to ignore clusters that have fewer than some small number of data points. If you do this, make sure you mention what threshold you used. This is done because due to the random sampling, small clusters frequently appear, but they often disappear as quickly as they appeared.
3. The DP sampler will consistently perform poorly on several of the data sets. Which ones and why might it be performing poorly on these? It will help to look at several runs of the sampler on these data sets when coming up with your answer. Describe at a high level what difficulties the sampler is having in converging to the correct clusterings. This is a problem with our method of inference in the DP, not a problem with the specification of the DP.
4. Compare the performance of the DP and the finite sampler. This can be a very high level comparison.

Problem 3: SVM with Active Learning The active learning concepts discussed in class are pretty high-level. We will implement the "uncertainty sampling" concept here. Let's take the spam dataset from Assignment 1 (or some dataset of your interest), and attempt classification using SVM.

Complete the following:

1. Randomly set aside a subset of the data for testing; use the rest for training.
2. Select increasing sizes of the training set and train a SVM from each training set. Plot the test error of each SVM versus the training size.
3. Start off with a small training set, and train the corresponding SVM. Using the current SVM, pick the next training data that is closest to the separating hyperplane (i.e. the

data point with the highest uncertainty). To speed up the training process, you can select several data points in batch. Add the selected data to the current training set, and retrain a SVM. Repeat the process until the entire training set is used. Plot the test error of each SVM versus the training size.

4. Compare the plots. Do they make sense?
5. **Bonus** Instead of Uncertainty Sampling, we can also use "Query by Committee". Pick a set of classification techniques (e.g. nearest-neighbor, decision tree, naive bayes) and repeat the previous experiment with the following criterion for choosing the next data point: select the point that has the most disagreements among the classifiers. Does this approach perform better than Uncertainty Sampling?