Computer Science 315
Assignment 4

2014

Deadline: Friday, 25 April 2014

Overview

In this assignment you are given unlabelled data, i.e. the observations are not provided with class labels. The objective is to assign a class label to each observation. For this you will first implement the k-means algorithm. In this implementation you should experiment with different initializations provided by the Python implementation. Note in particular the problems that can arise from bad initializations, multiple local minima, and empty clusters.

Strictly speaking the Gaussian Mixture Model is not a clustering algorithm, but can be used as such. Implement the GMM, using the skeleton code as provided. Use the k-means algorithm and the binary split to initialize the Expectation Maximization (EM) algorithm.

Write a short report that explains your findings. Restrict your report to your contributions, explain through images, tables and/or graphs where possible. Report anything interesting that you have observed, and tell me your conclusions. Code is not part of a report, but you may include any interesting fragments that you may want to bring to my attention.

Bad style and sloppy work will be penalized!

Investigation

k-Means

One important application of the k-means algorithm is so-called Vector Quatization (VQ). You are going to apply VQ to color images. First note that any pixel is represented by a triplet $[r, g, b]$ where each of them can typically assume one of 256 values, i.e. each pixel can be presented by 256^3 different values. This is a lot! You want to reduce the number of possibilities through VQ.

Take all the pixels of a picture(s) of your choice as data values and do k-means clustering using an increasing number of clusters (start with 2 clusters). Now reconstruct your image by assigning each pixel value the value of its cluster mean. Display the result as an image for each value of k. What happens as you increase k?

Are some images better suited for this type of treatment than others?

Experiment a little and report your findings.

Put this investigation in a Python module called `vq.py`. Your module must be set up so that running the module outputs the results and any comments you have on the results.

GMM

1. Load the data in the file `data/speech.dat`. This is a pickled[1] dictionary, containing a training and a test set for a diphthong classification task. Each set is a dictionary, with a key for each diphthong. The value corresponding to each key is a list of encoded speech fragments represented as nx16 arrays.[2]

2. Normalize the training and test data by subtracting the mean of all the 16-dimensional training points.

3. We shall assume that the rows of the nx16 arrays are i.i.d. samples from an underlying distribution, which is the same for all samples from a given diphthong. Thus, we can use all the rows from all the speech fragments for a diphthong in the training set to construct a model of the 16-dimensional points appearing in a speech signal representing that diphthong. Do this using a GMM to model each diphthong. Since the GMM's initialization use k-means with binary split) can lead to local optima, fit each diphthong's model a number of times, and keep the one with the best negative log-likelihood.

4. Given these models of each diphthong, and information on the prior probabilities of each diphthong appearing, one can calculate the most likely class for a given observation (i.e. collection of 16-dimensional points). This is done by appropriately adjusting the negative log-likelihood of each model for the given observation to take the prior proportion information into account.

5. Perform the classification approach described above for k from 1 to 6. You can experiment with GMMs where both full– and diagonal covariance matrices are used. (This should take a while to run, and may sometimes give anomalous results, and thus should not be executed in your submission when `speech.py` is run. However, the code to do this should be in the submission.)

6. Plot typical training and test error rates calculated in the last step against k on a line plot, and try to explain what you see.

7. Develop a Python module called `speech.py` for this application.

[1] Python uses a module called `pickle` to save Python objects to a file. Such saved objects are said to be pickled. If you are not familiar with reading and writing Python objects to file, it is recommended you consult the documentation of Python's `pickle` module.

[2] The data points are a small selection of *cepstra* of speech samples from a standard database for speech recognition problems, the Texas Instruments/Massachusetts Institute of Technology (TIMIT) database. To find out more about this representation, see the Wikipedia page for "cepstrum".

8. Now do the same with the signature data set. Again build an GMM for each signature using the training set. Test your models using the test sets. Develop a Python module called `sign.py` for this application.

Experiment a little and report your findings.

Your modules must be set up so that running the module outputs the results and any comments you have on the results.

Evaluation

1. Implementation: 10 marks
2. Investigation: 30 marks. (k-means — 15; GMMs — 15)
3. Report: 60 marks