University of Applied Sciences RheinMain
Prof. Dr. Adrian Ulges

10.11.2016

Machine Learning

Course Work 3

to complete by: 17.11.2016

Please execute this course work (as well as any following) in teams of two.

Exercise 3.1 (Expectation Maximization)
Implement Expectation Maximization for Gaussian Mixture Models as introduced in the lecture. Implement the method yourself, do not use a ready-made implementation.

- Load `EM.zip` from the course homepage and run `EM.py`. This should give you a plot of some 2D test data and then raise a `NotImplementedError`.
- Implement the `train()`-method in the class `ExpectationMaximization`. Use diagonal covariance matrices (see `np.diag()`), i.e. set the covariances to 0 but learn each cluster's variances. Keep a uniform prior $P(c)$ for the clusters (i.e., do not learn the prior). Initialize the cluster's means with random samples from the input data (see `random.sample()`), and the variances with the whole dataset's variance.
- Avoid loops. Use Numpy's array operations instead. This makes your code more readable and (a lot) more efficient. As an example: Here is some bad (left) and good (right) code that standardizes the columns of an $n \times m$ data matrix X.

```
def standardize_bad(X):
    n,m = X.shape
    for col in range(m):
        # compute mean
        mean = 0.
        for row in range(n):
            mean += X[row,col]
        mean = mean / n
        # compute std. deviation
        var = 0.
        for row in range(n):
            var += (X[row,col]-mean)**2
        std = sqrt(var / n)
        # standardize
        for row in range(n):
            X[row,col] = (X[row,col]-mean)/std
    return X
```

```
def standardize_good(X):
    means = X.mean(axis=0)
    stds  = X.std(axis=0)
    return (X-means)/stds
```

- Use `scipy.stats.multivariate_normal` to compute the multivariate normal density.
- Test your code with the small 3-cluster dataset from `generate_test_data()`. Run `plot()` after each iteration and illustrate the progress of your clustering.
- Simply run your EM for 100 iterations (think about a better stopping criterion, though).

Exercise 3.2 (Multiple Restarts)
Implement a method `quality(model, X)` that measures how well your model fits your data X. Restart your clustering multiple times (each time initializing with different random means) and keep only the best result.

Exercise 3.3 (Keyframe Extraction)
Given a video sequence, we want to extract a few keyframes that represent the video 'as well as possible'. For example, have a look at the "Game of Thrones" scene in `got.mp4`: We would clearly expect 3 keyframes here, one showing Sansa, one showing Jon, one showing the total. Obviously, we could solve the problem by clustering the video's frames and picking one representative frame per cluster. Implement a class `KeyframeExtractor` following this idea:

- Read an input video's frames using OpenCV (`cv2.VideoCapture`). For speedup reasons, only use every q'th frame (make q a free parameter).
- Represent each frame by its $(8 \times 8 \times 8)$ color histogram (see `cv2.calcHist()`).
- Collect all color histograms and reduce them to P dimensions by PCA (using `sklearn`'s PCA class). Use $P = 2$ (you can also run tests with $P > 2$ and check if you get better results).
- Run EM clustering on the P-dimensional 'frame features'. Visualize the features and your learned EM model (you can simly call `plot()`).
- For each cluster k, write out the keyframe $\mathbf{x}$ that matches the cluster best (the one that maximizizes $p(\mathbf{x}|\mu_k, \Sigma_k)$).
- Test your results for the videos `got.mp4` (with 3 keyframes), `99.mp4` (with 6 keyframes), and `lwt.mp4` (with 5 keyframes). Do you obtain reasonable results?

Exercise 3.4 (Sources)
Check-in an up-to-date version of your project by Wed, 16th, 16:00.

Exercise 3.5 (Report)
Put together a presentation of 2-3 slides, showing your keyframes, visualizations, status, and any open questions.