Programming Assignment 3

Gaussian mixture model and EM

1 EM Algorithm

1.1 singular matrix

When we encounter singular covariance matrix where Gaussian probability density function is not calculable, we use `allow_singular` flag for `normal_distribution.pdf` which, omits the check for singularity.

1.2 Tuning number of components K

As we can see from figure 1 and 2 that the marginal benefit dramatically decreases as the roughly the "true" number of distribution combination. Certainly we can use this trick to more complex dataset, but it won't be as obvious, or perhaps there is no "true" number.

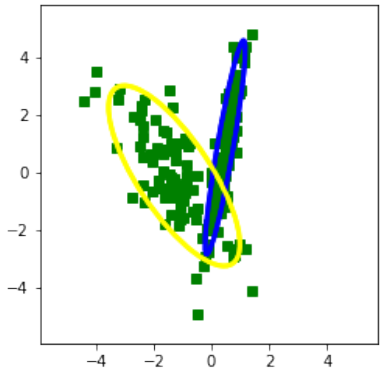

Figure 1: dataset 2, where 2 Gaussians is observed intuitively

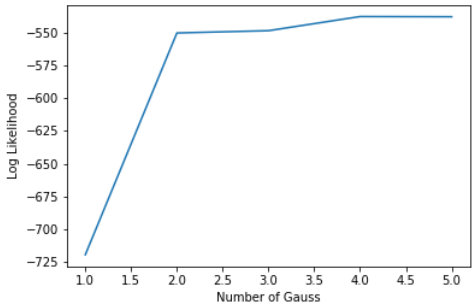

Figure 2: Tuning over K

Here is an example (Figure 3) dataset whose true combination number is not observable intuitively or analytically. In this case, to tune over K, we are therefore blind in terms of the "True Gaussian" trick mentioned above. Since increasing K does not dramatically increase log likelihood for any K (consult Figure 4), although moronically increases, the best guess would arguably be $K = 1$.

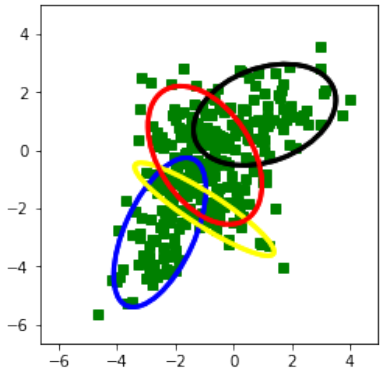

Figure 3: dataset 3, where number of Gaussian is not directly observable

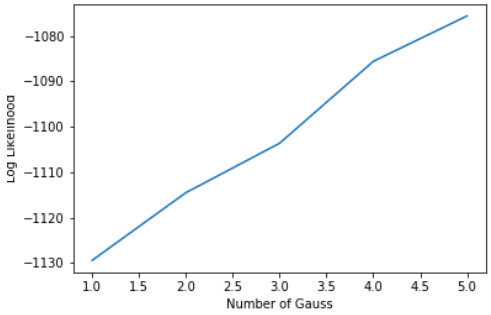

Figure 4: Tuning over K on dataset 3

Corresponding `small` dataset have similar behavior, although varies a little on different runs, but overall behavior is all the same, multiple restarts solves this.

1.3 Tuning μ and Σ

As we mentioned in the last section that `small` dataset is more prone to bad initial μ and Σ since their optima tends to vary among restarts.

We now compare two ways of how μ was initialized. By default, we sample K random points from X as our μ. Here another method is shown: where $\mu_{kd} = random(MIN_{X_d}, MAX_{X_d})$.

This second method helps avoid local optima for small datasets. Take `small_2` for example (since we sort-of know that it has two Gaussians), as shown in Figure 5 that the second method converged to the global optima at $K = 2$ whereas the default method (random sample) does not. Certainly, is issue is not observed for large datasets, but it is worth noticing for small or unknown datasets.

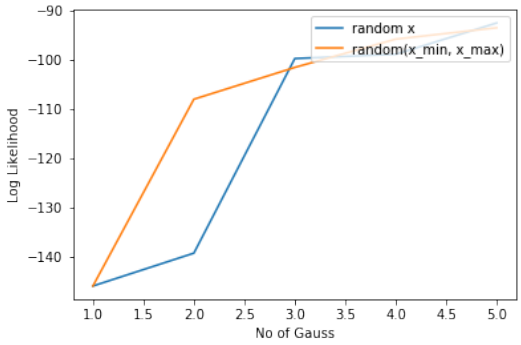

Figure 5: two different ways of initializing μ

I would have expected that sampling method would take shorter iterations to converge. Surprisingly, we do not see significant change in number of iteration different between above mentioned two methods.

No observable different occurred during tuning of Σ.

1.4 Convergence Criterion

As shown in `iteration` method, we directly calculate the equality of $l(\theta)$ and $l(\theta')$. If we were using a truly continuous system, this will never halt; however, we know that Python (or any program for that matter) has floating number representation limit. Therefore, we deem the algorithm converge when we hit floating number limit in Python.

2 Variations

2.1 Diagonal Matrix

Instead updating the full covariance matrix, we assume only diagonal values are non-zero. Theoretically, this enables us to save space; but for simplicity and consistency of code, we just use zero instead. This assumption indicates that our features do not co-vary, in other words, independent.

Formally, we denote the diagonal vector (which is the variance vector) as σ^2, hence making our Gaussian probability density function has the following form:

$$pdf(x|\mu,\sigma) = \frac{1}{2\pi^{d/2}\prod_{i=1}^{d}\sigma_d^2} exp(-\frac{1}{2}\sum_{j=1}^{d}\frac{(x_d-\mu_d)^2}{\sigma_d^2})$$

Hence for every M step, we update the variance vector as follows:

$$\sigma_{ik}'^2 = \frac{\sum_n r(z_{kn})(x_{in}-\mu_{ik}')^2}{\sum_n r(z_{kn})}$$

where $i \in \{1,...,d\}$, r is the responsibility and z is the latent variable, for which we assume that we know

the value in this step (calculated in the previous step). Notice that σ_k^2 is now a d-dimensional vector instead of a $d \times d$ matrix.

Also worth mention is that diagonal matrix representation is more efficient to train.

2.2 K-Means as Initializer for μ

Although we can augment our GMM to imitate K-Means, we wrote an K-Means from scratch so that it would be more efficient, we will discuss running time later in next section.

Since we are preparing μ for Gaussian model, we will use euclidean distance as the distance metric for K-Means subroutine.

2.3 Diagonal Matrix GMM vs. general GMM

Certainly Diagonal Matrix is not as descriptive as full-fledged co-variance matrix, which means, in turn, makes it less likely to over-fit, as shown in Figure 6; notice the obvious over-fitting occurred in the upper-right corner for covariance matrix model.

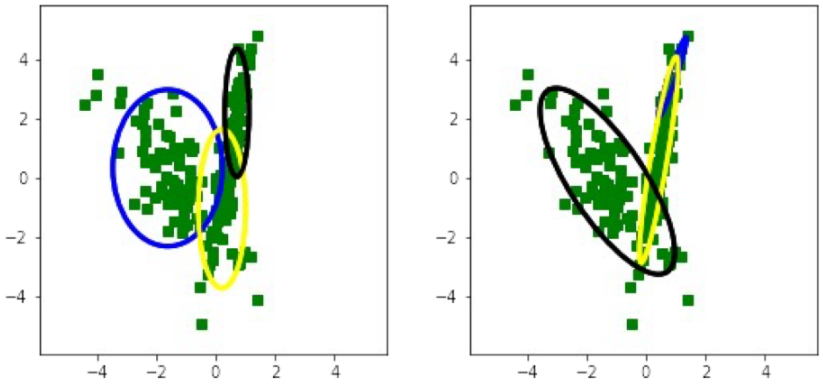

Figure 6: dataset 2, $K = 3$, Diagonal Matrix (left), Covariance Matrix (right)

2.4 K-Means initializer GMM vs. general GMM

Comparing to initialization of μ at random, the most salient effect of using a K-Means initializer is in improvement of running time. As shown in Figure X, K-Means has significantly lower running time on all datasets while achieving almost identical end results (not shown).

3 Model Selection

3.1 How EM was setup

For efficiency reasons, we do not repeatedly rerun the model with random parameters, we just run the model once with the random parameter. In practice, we see that there is no real observable differences, which suggests that our model has really high probability of converging to something close to global optima (if not on global optima) on the given datasets.

3.2 Testing out models on small datasets

As we can see in Figure 7 that although we made strong assumption regarding covariance being 0 in the diagonal case, the log likelihood do not degrade much for all three datasets. As expected, not being as descriptive, diagonal models has lower log likelihood. However, we do not conclude that they performs less well since we do not know whether covariance models are over-fitting since we do not have validation data.

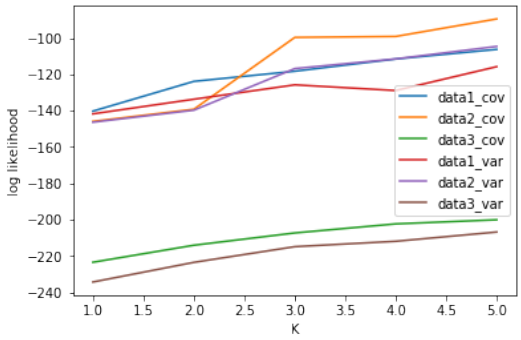

Figure 7: Small datasets

3.3 Large datasets

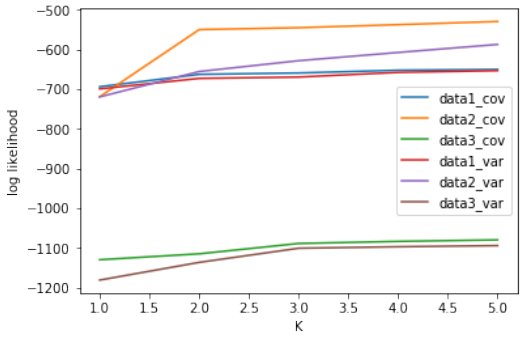

Figure 8: Large datasets

Obviously large datasets are not as prone to local optima, but more prune to over-fitting.

3.4 Cross Validation

Here is an obvious trend of over-fitting illustrated by Figure 9 where we use 3-fold cross validation and observe the effect of increasing K. Notice that for almost all of the datasets and models (whether covariance or variance), as we increase K beyond 4, the validation performance decreases. The sole exception is dataset 2, which we have been talked about, is an obvious $K = 2$ covariance dataset, it would be naturally for our covariance model drop significantly after 3.

Needless to say that the performance is measured by log likelihood on the validation set, not on the training set using the trained parameters.

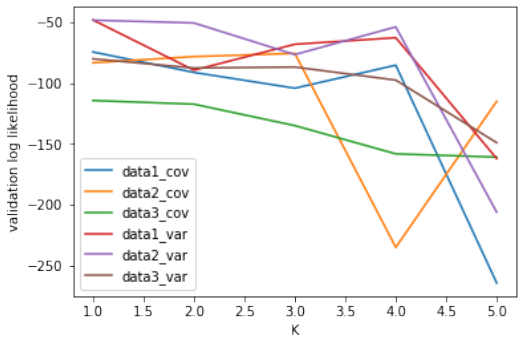

Figure 9: 3-Fold Validation

If we compare covariance model with the variance version, from Figure 10 we see that variance model performs strictly better than covariance model although marginal decrease increases as K gets bigger due to over-fitting.

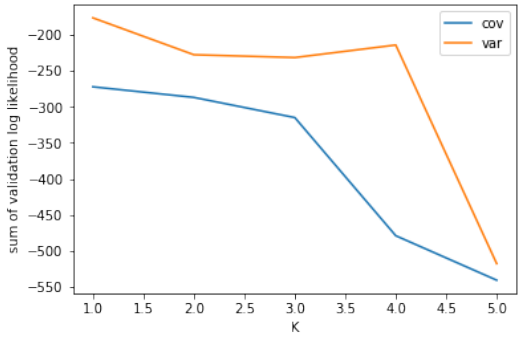

Figure 10: average log likelihood for 3-fold cross validate

3.5 Number of folds

As number of folds increases, we have more data to work with, hence our model is able to fit the data better as shown in Figure 11. Of course here we are shown with a $K = 3$ case where over-fitting is not much of a problem.

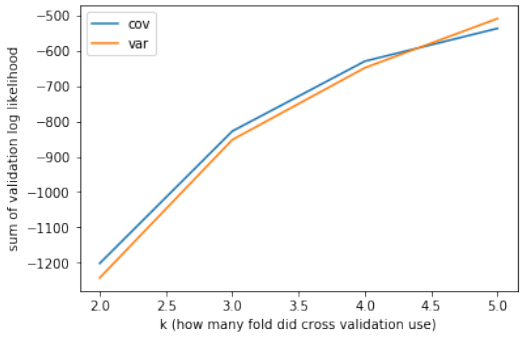

Figure 11: k-fold

4 Prediction

4.1 Mystery 1

From log likelihood heat map (Figure 12) that mystery 1 could be a $K = 1$ covariance matrix model dataset or $K = 2$ variance vector model dataset.

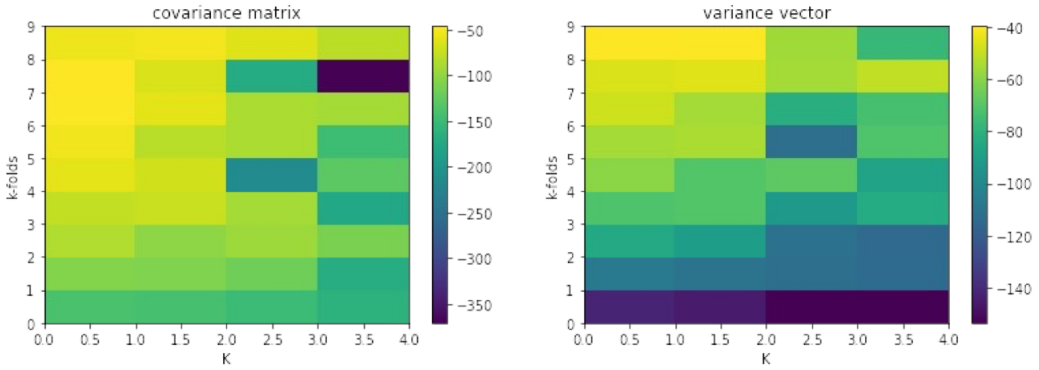

Figure 12: k-fold

4.2 Mystery 2

From log likelihood heat map (Figure 12) that mystery 2 could be a $K = 2$ covariance matrix model dataset or $K = 1$ variance vector model dataset.

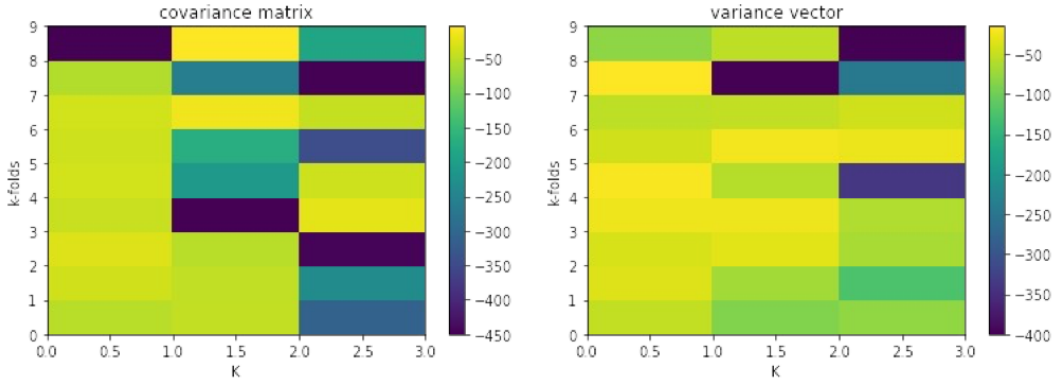

Figure 13: k-fold