7. Implementing EM for matrix completion

Extension Note: Project 4 due date has been extended by 1 more day to **August 22 23:59UTC** .

We need to update our EM algorithm a bit to deal with the fact that the observations are no longer complete vectors. We use Bayes' rule to find an updated expression for the posterior probability $p(j|u) = P(y=j|x_{C_u}^{(u)})$:

$$p(j \mid u) = \frac{p(u|j) \cdot p(j)}{p(u)} = \frac{p(u|j) \cdot p(j)}{\sum_{j=1}^{K} p(u|j) \cdot p(j)} = \frac{\pi_j N(x_{C_u}^{(u)}; \mu_{C_u}^{(j)}, \sigma_j^2 I_{C_u \times C_u})}{\sum_{j=1}^{K} \pi_j N(x_{C_u}^{(u)}; \mu_{C_u}^{(j)}, \sigma_j^2 I_{C_u \times C_u})}$$

This is the soft assignment of cluster u to data point j.

To minimize numerical instability, you will be re-implementing the E-step in the log-domain, so you should calculate the values for the log of the posterior probability, $\ell(j,u) = \log(p(j|u))$ (though the actual output of your E-step should include the non-log posterior).

Let $f(u,i) = \log(\pi_i) + \log\left(N(x_{C_u}^{(u)}; \mu_{C_u}^{(i)}, \sigma_i^2 I_{C_u \times C_u})\right)$. Then, in terms of f, the log posterior is:

$$\begin{aligned}
\ell(j|u) \quad &= \log(p(j \mid u)) = \log\left(\frac{\pi_j N(x_{C_u}^{(u)}; \mu_{C_u}^{(j)}, \sigma_j^2 I_{C_u \times C_u})}{\sum_{j=1}^{K} \pi_j N(x_{C_u}^{(u)}; \mu_{C_u}^{(j)}, \sigma_j^2 I_{C_u \times C_u})}\right) \\
&= \log\left(\pi_j N(x_{C_u}^{(u)}; \mu_{C_u}^{(j)}, \sigma_j^2 I_{C_u \times C_u})\right) - \log\left(\sum_{j=1}^{K} \pi_j N(x_{C_u}^{(u)}; \mu_{C_u}^{(j)}, \sigma_j^2 I_{C_u \times C_u})\right) \\
&= \log(\pi_i) + \log\left(N(x_{C_u}^{(u)}; \mu_{C_u}^{(i)}, \sigma_i^2 I_{C_u \times C_u})\right) - \log\left(\sum_{j=1}^{K} \exp\left(\log\left(\pi_j N(x_{C_u}^{(u)}; \mu_{C_u}^{(j)}, \sigma_j^2 I_{C_u \times C_u})\right)\right)\right) \\
&= f(u,j) - \log\left(\sum_{j=1}^{K} \exp(f(u,j))\right)
\end{aligned}$$

Once we have evaluated $p(j|u)$ in the E-step, we can proceed to the M-step. We wish to find the parameters π, μ, and σ that maximize $\ell(X;\theta)$, the expected complete log-likelihood:

$$\ell(X;\theta) = \sum_{u=1}^{n} \log\left(\sum_{j=1}^{K} \pi_j N(x_{C_u}^{(u)} | \mu_{C_u}^{(j)}, \sigma_j^2 I_{|C_u| \times |C_u|})\right),$$

To maximize $\ell(X;\theta)$, we keep $p(j|u)$ (the soft-assignments) fixed, and maximize over the model parameters. Some of the parameters can be updated exactly as before with complete example vectors. For example,

$$\hat{\pi}_j = \frac{\sum_{u=1}^{n} p(j|u)}{n}$$

But we must be more careful in updating $\mu^{(j)}$ and σ_j^2. This is because the parameters appear differently in the likelihood depending on how incomplete the observation is. Notice that some coordinates of $\mu^{(j)}$ do not impact observation $x_{C_u}^{(u)}$ at all. But we can proceed to separately update each coordinate of $\mu^{(j)}$.

We will take the derivative with respect to the to the lth movie coordinate for cluster k.

First, note that, by decomposing the multivariate spherical Gaussians into univariate spherical Gaussians as before, we can write, if $k \in C_u$:

$$\begin{aligned}
\frac{\partial}{\partial \mu_l^{(k)}} N\left(x_{C_u}^{(u)} | \mu_{C_u}^{(k)}, \sigma_k^2 I_{|C_u| \times |C_u|}\right) &= N\left(\ldots\right) \frac{\frac{\partial}{\partial \mu_l^{(k)}}\left(\frac{1}{\sqrt{2\pi}\sigma l,(k)} \exp\left(-\frac{1}{2\sigma_{l,(k)}^2}\left(x_l^{(u)} - \mu_l^{(k)}\right)^2\right)\right)}{\left(\frac{1}{\sqrt{2\pi}\sigma l,(k)} \exp\left(-\frac{1}{2\sigma_{l,(k)}^2}\left(x_l^{(u)} - \mu_l^{(k)}\right)^2\right)\right)} \\
&= N\left(\ldots\right) \frac{x_l^{(u)} - \mu_l^{(k)}}{\sigma_{l,(k)}^2}
\end{aligned}$$

where $N\left(\ldots\right) = N\left(x_{C_u}^{(u)} | \mu_{C_u}^{(k)}, \sigma_k^2 I_{|C_u| \times |C_u|}\right)$.

If $k \notin C_u$, that derivative is 0. To cover both cases, we can write:

$$\frac{\partial}{\partial \mu_l^{(k)}} N\left(x_{C_u}^{(u)} | \mu_{C_u}^{(k)}, \sigma_k^2 I_{|C_u| \times |C_u|}\right) = N\left(x_{C_u}^{(u)} | \mu_{C_u}^{(k)}, \sigma_k^2 I_{|C_u| \times |C_u|}\right) \delta\left(l, C_u\right) \frac{x_l^{(u)} - \mu_l^{(k)}}{\sigma_{l,(k)}^2}$$

where $\delta\left(i, C_u\right)$ is an indicator function: 1 if $i \in C_u$ and zero otherwise.

Therefore,

$$\begin{aligned}
\frac{\partial \ell\left(X; \theta\right)}{\partial \mu_l^{(k)}} &= \sum_{u=1}^{n} \frac{\frac{\partial}{\partial \mu_l^{(k)}} \sum_{j=1}^{K} \pi_j N\left(x_{C_u}^{(u)} | \mu_{C_u}^{(j)}, \sigma_j^2 I_{|C_u| \times |C_u|}\right)}{\sum_{j=1}^{K} \pi_j N\left(x_{C_u}^{(u)} | \mu_{C_u}^{(j)}, \sigma_j^2 I_{|C_u| \times |C_u|}\right)} \\
&= \sum_{u=1}^{n} \frac{\pi_k \frac{\partial}{\partial \mu_l^{(k)}} N\left(x_{C_u}^{(u)} | \mu_{C_u}^{(k)}, \sigma_k^2 I_{|C_u| \times |C_u|}\right)}{\sum_{j=1}^{K} \pi_j N\left(x_{C_u}^{(u)} | \mu_{C_u}^{(j)}, \sigma_j^2 I_{|C_u| \times |C_u|}\right)} \\
&= \sum_{u=1}^{n} \frac{\pi_k N\left(x_{C_u}^{(u)} | \mu_{C_u}^{(k)}, \sigma_k^2 I_{|C_u| \times |C_u|}\right)}{\sum_{j=1}^{K} \pi_j N\left(x_{C_u}^{(u)} | \mu_{C_u}^{(j)}, \sigma_j^2 I_{|C_u| \times |C_u|}\right)} \delta\left(l, C_u\right) \frac{x_l^{(u)} - \mu_l^{(k)}}{\sigma_{l,(k)}^2} \\
0 &= \sum_{u=1}^{n} p\left(k|u\right) \delta\left(l, C_u\right) \frac{x_l^{(u)} - \mu_l^{(k)}}{\sigma_{l,(k)}^2} \\
\hat{\mu}_l^{(k)} &= \frac{\sum_{u=1}^{n} \delta\left(l, C_u\right) p\left(k \mid u\right) x_l^{(u)}}{\sum_{u=1}^{n} \delta\left(l, C_u\right) p\left(k \mid u\right)} \\
\hat{\mu}_l^{(j)} &= \frac{\sum_{u=1}^{n} \delta\left(l, C_u\right) p\left(j \mid u\right) x_l^{(u)}}{\sum_{u=1}^{n} \delta\left(l, C_u\right) p\left(j \mid u\right)}
\end{aligned}$$

We do **not** compute the mean update in the log domain; we use $p\left(j|u\right)$ instead of $\ell\left(j, u\right)$. When you set $\mu_i^{(j)}$ and σ_j^2 in the implementation, it will be easier, and not lead to numerical underflow issues, to use $p\left(j|u\right)$ instead of the logarithm $\ell\left(j, u\right)$.

Finally, the update equation for the variance is not too different from before:

$$\hat{\sigma}_j^2 = \frac{1}{\sum_{u=1}^{n} |C_u| p(j|u)} \sum_{u=1}^{n} p(j|u) \left\| x_{C_u}^{(u)} - \hat{\mu}_{C_u}^{(j)} \right\|^2$$

Implementation guidelines:

- You may find LogSumExp useful. But remember that your M-step should return the new $P = \hat{\pi}$, not the log of $\hat{\pi}$.

- The following will not affect the update equation above, but will affect your implementation: since we are dealing with incomplete data, we might have a case where most of the points in cluster j are missing the i-th coordinate. If we are not careful, the value of this coordinate in the mean will be determined by a small number of points, which leads to erratic results. Instead, we should only update the mean when $\sum_{u=1}^{n} p(j|u) \delta(i, C_u) \geq 1$ Since $p(j|u)$ is a soft probability assignment, this corresponds to the case when at least one full point supports the mean.

- To also avoid the variances of clusters going to zero due to a small number of points being assigned to them, in the M-step you will need to implement a minimum variance for your clusters. We recommend a value of 0.25, though you are free to experiment with it if you wish. Note that this issue, as well as the thresholded mean update in the point above, are better dealt with through regularization; however, to keep things simple, we do not do regularization here.

- To debug your EM implementation, you may use the data files `test_incomplete.txt` and `test_complete.txt`. Compare your results to ours from `test_solutions.txt`.
  Correction note (Aug 13) : The file `test_solutions.txt` has been updated on Aug 13. Please make sure to use the version in the latest netflix.tar.gz.

Implementing E-step (2)

1.0/1.0 point (graded)

In `em.py`, fill in the `estep` function so that it works with partially observed vectors where missing values are indicated with zeros, and perform the computations in the log domain to help with numerical stability.

Available Functions: You have access to the NumPy python library as `np`, to the `GaussianMixture` class and to typing annotation `typing.Tuple` as `Tuple`. You also have access to `scipy.special.logsumexp` as `logsumexp`

Hint: For this function, you will want to use `log(mixture.p[j] + 1e-16)` instead of `log(mixture.p[j])` to avoid numerical underflow

```

    """
    mu, var, weight = mixture
    n, _ = X.shape
    k, _ = mu.shape
    log_prob_mat = np.zeros([n, k])
    Xm = np.ma.array(X, mask = (X == 0))
    for i in range(k):
        log_prob = np.log(weight[i] + 1e-16) + np.log(pdf_gaussian(Xm, mu[i], var[i]))
        log_prob_mat[:,i] = log_prob
    log_prob_all = logsumexp(log_prob_mat, axis = 1)
    log_post = log_prob_mat - np.tile(log_prob_all.reshape(n, 1), (1, k))
    post = np.exp(log_post)
    log_likelihood = np.sum(log_prob_all)
    return post, log_likelihood
```

Press ESC then TAB or click outside of the code editor to exit

Correct

```
def estep(X: np.ndarray, mixture: GaussianMixture) -> Tuple[np.ndarray, float]:
    """E-step: Softly assigns each datapoint to a gaussian component

    Args:
        X: (n, d) array holding the data, with incomplete entries (set to 0)
        mixture: the current gaussian mixture

    Returns:
        np.ndarray: (n, K) array holding the soft counts
            for all components for all examples
        float: log-likelihood of the assignment

    """
    n, _ = X.shape
    K, _ = mixture.mu.shape
    post = np.zeros((n, K))

    ll = 0
    for i in range(n):                              可以直接做mask，效果一样
        mask = (X[i, :] != 0)
        for j in range(K):
            log_likelihood = log_gaussian(X[i, mask], mixture.mu[j, mask],
                                          mixture.var[j])
            post[i, j] = np.log(mixture.p[j] + 1e-16) + log_likelihood
        total = logsumexp(post[i, :])
        post[i, :] = post[i, :] - total
        ll += total

    return np.exp(post), ll


def log_gaussian(x: np.ndarray, mean: np.ndarray, var: float) -> float:
    """Computes the log probablity of vector x under a normal distribution

    Args:
        x: (d, ) array holding the vector's coordinates
        mean: (d, ) mean of the gaussian
        var: variance of the gaussian

    Returns:
        float: the log probability
    """
    d = len(x)
    log_prob = -d / 2.0 * np.log(2 * np.pi * var)
    log_prob -= 0.5 * ((x - mean)**2).sum() / var
    return log_prob
```

Test results

See full output

CORRECT

See full output

Submit   You have used 5 of 20 attempts

Answers are displayed within the problem

Implementing M-step (2)

1.0/1.0 point (graded)

In `em.py`, fill in the `mstep` function so that it works with partially observed vectors where missing values are indicated with zeros, and perform the computations in the log domain to help with numerical stability.

Available Functions: You have access to the NumPy python library as `np`, to the `GaussianMixture` class and to typing annotation `typing.Tuple` as `Tuple`. You also have access to `scipy.misc.logsumexp` as `logsumexp`

Correction Note (Aug 8): The boilerplate code for this function was changed on August 8th. Make sure you have the latest version of netflix.tar.gz, or correct the file `em.py` as follows:

```diff
- def mstep(X: np.ndarray, post: np.ndarray,
+ def mstep(X: np.ndarray, post: np.ndarray, mixture: GaussianMixture,
           min_variance: float = .25) -> GaussianMixture:
     """M-step: Updates the gaussian mixture by maximizing the log-likelihood
     of the weighted dataset
@@ -34,6 +34,7 @@ def mstep(X: np.ndarray, post: np.ndarray,
         X: (n, d) array holding the data, with incomplete entries (set to 0)
         post: (n, K) array holding the soft counts
             for all components for all examples
+        mixture: the current gaussian mixture
         min_variance: the minimum variance for each gaussian
```

```python
def mstep(X: np.ndarray, post: np.ndarray, mixture: GaussianMixture,
          min_variance: float = .25) -> GaussianMixture:
    """M-step: Updates the gaussian mixture by maximizing the log-likelihood
    of the weighted dataset

    Args:
        X: (n, d) array holding the data, with incomplete entries (set to 0)
        post: (n, K) array holding the soft counts
            for all components for all examples
        mixture: the current gaussian mixture
        min_variance: the minimum variance for each gaussian

    Returns:
        GaussianMixture: the new gaussian mixture
    """
    old_mu       = mixture
```

Press ESC then TAB or click outside of the code editor to exit

Correct

```
def mstep(X: np.ndarray, post: np.ndarray, mixture: GaussianMixture,
          min_variance: float = .25) -> GaussianMixture:
    """M-step: Updates the gaussian mixture by maximizing the log-likelihood
    of the weighted dataset

    Args:
        X: (n, d) array holding the data, with incomplete entries (set to 0)
        post: (n, K) array holding the soft counts
            for all components for all examples
        mixture: the current gaussian mixture
        min_variance: the minimum variance for each gaussian

    Returns:
        GaussianMixture: the new gaussian mixture
    """
    n, d = X.shape
    _, K = post.shape

    n_hat = post.sum(axis=0)
    p = n_hat / n

    mu = mixture.mu.copy()
    var = np.zeros(K)

    for j in range(K):
        sse, weight = 0, 0
        for l in range(d):
            mask = (X[:, l] != 0)
            n_sum = post[mask, j].sum()
            if (n_sum >= 1):
                # Updating mean
                mu[j, l] = (X[mask, l] @ post[mask, j]) / n_sum
            # Computing variance
            sse += ((mu[j, l] - X[mask, l])**2) @ post[mask, j]
            weight += n_sum
        var[j] = sse / weight
        if var[j] < min_variance:
            var[j] = min_variance

    return GaussianMixture(mu, var, p)
```

Test results

See full output

CORRECT

See full output

Submit

You have used 3 of 20 attempts

Answers are displayed within the problem

Implementing run

1.0/1.0 point (graded)

In `em.py`, fill in the `run` function so that it runs the EM algorithm. As before, the convergence criteria that you should use is that the improvement in the log-likelihood is less than or equal to 10^{-6} multiplied by the absolute value of the new log-likelihood.

Available Functions: You have access to the NumPy python library as `np`, to the `GaussianMixture` class and to typing annotation `typing.Tuple` as `Tuple`. You also have access to the `estep` and `mstep` functions you have just implemented

Correction note (Aug 8): Since the `mstep` function in previous problem has been defined differently since Aug 8, you will need to modify `run` function accordingly. Note that the grader will accept as correct a `run` function that works with either the earlier or current version of `mstep`.

```
            for all components for all examples
        float: log-likelihood of the current assignment
    """
    old_log_likelihood = None
    while 1:
        post, new_log_likelihood = estep(X, mixture)
        mixture = mstep(X, post, mixture)
        if old_log_likelihood is not None:
            if (new_log_likelihood - old_log_likelihood) < 1e-6 * abs(new_log_likelihood):
                break
        old_log_likelihood = new_log_likelihood
    return mixture, post, new_log_likelihood
```

Press ESC then TAB or click outside of the code editor to exit

Correct

```
def run(X: np.ndarray, mixture: GaussianMixture,
        post: np.ndarray) -> Tuple[GaussianMixture, np.ndarray, float]:
    """Runs the mixture model

    Args:
        X: (n, d) array holding the data
        post: (n, K) array holding the soft counts
            for all components for all examples

    Returns:
        GaussianMixture: the new gaussian mixture
        np.ndarray: (n, K) array holding the soft counts
            for all components for all examples
        float: log-likelihood of the current assignment
    """

    prev_ll = None
    ll = None
    while (prev_ll is None or ll - prev_ll > 1e-6 * np.abs(ll)):
        prev_ll = ll
        post, ll = estep(X, mixture)
        mixture = mstep(X, post, mixture)

    return mixture, post, ll
```

Test results

See full output

CORRECT

See full output

Submit You have used 2 of 20 attempts

Answers are displayed within the problem

Discussion

Show Discussion

Topic: Unit 4 Unsupervised Learning (2 weeks) :Project 4: Collaborative Filtering via Gaussian Mixtures / 7. Implementing EM for matrix completion