3. Expectation–maximization algorithm

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

Recall the Gaussian mixture model presented in class:

$$P(x|\theta) = \sum_{j=1}^{K} \pi_j N(x; \mu^{(j)}, \sigma_j^2 I),$$

where θ denotes all the parameters in the mixture (means $\mu^{(j)}$, mixing proportions π_j, and variances σ_j^2). The goal of the EM algorithm is to estimate these unknown parameters by maximizing the log-likelihood of the observed data $x^{(1)}, \ldots, x^{(n)}$. Starting with some initial guess of the unknown parameters, the algorithm iterates between E- and M-steps. The E-Step softly assigns each data point $x^{(i)}$ to mixture components. The M-step takes these soft-assignments as given and finds a new setting of the parameters by maximizing the log-likelihood of the weighted dataset (expected complete log-likelihood).

Implement the EM algorithm for the Gaussian mixture model desribed above. To this end, complete the functions `estep`, `mstep` and `run` in `naive_em.py`. In our notation,

- `X`: an (n, d) Numpy array of n data points, each with d features
- `K`: number of mixture components
- `mu`: (K, d) Numpy array where the j^{th} row is the mean vector $\mu^{(j)}$
- `p`: $(K,)$ Numpy array of mixing proportions π_j, $j = 1, \ldots, K$
- `var`: $(K,)$ Numpy array of variances σ_j^2, $j = 1, \ldots, K$

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. In slightly more algebraic notation:
$\text{new log-likelihood} - \text{old log-likelihood} \leq 10^{-6} \cdot |\text{new log-likelihood}|$

Your code will output updated versions of a `GaussianMixture` (with means `mu`, variances `var` and mixing proportions `p`) as well as an (n, K) Numpy array `post`, where $\text{post}[i, j]$ is the posterior probability $p(j|x^{(i)})$, and LL which is the log-likelihood of the weighted dataset.

Here are a few points to check to make sure that your implementation is indeed correct:

1. Make sure that all your functions return objects with the right dimension.

2. EM should monotonically increase the log-likelihood of the data. Initialize and run the EM algorithm on the toy dataset as you did earlier with K-means. You should check that the LL values that the algorithm returns after each run are indeed always monotonically increasing (non-decreasing).

3. Using $K = 3$ and a seed of 0, on the toy dataset, you should get a log likelihood of -1388.0818.

4. As a runtime guideline, in your testing on the toy dataset, calls of `run` using the values of K that we are testing should run in on the order of seconds (i.e. if each call isn't fairly quick, that may be an indication that something is wrong).

5. Try plotting the solutions obtained with your EM implementation. Do they make sense?

Implementing E-step

1.0/1.0 point (graded)

Write a function `estep` that performs the E-step of the EM algorithm

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

```
    """
    mu, var, weight = mixture
    n, _ = X.shape
    k, _ = mu.shape
    prob_mat = np.zeros([n, k])
    prob_all = np.zeros(n)
    log_likelihood = 0
    for i in range(k):
        prob = weight[i] * pdf_2dgaussian(X, mu[i], var[i])
        prob_mat[:,i] = prob
        prob_all += prob
        log_likelihood += prob
    soft_counts = prob_mat / np.tile(prob_all.reshape(n, 1), (1, k))
    log_likelihood = np.sum(np.log(log_likelihood))
    return soft_counts, 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
        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):
        for j in range(K):
            likelihood = gaussian(X[i], mixture.mu[j], mixture.var[j])
            post[i, j] = mixture.p[j] * likelihood
        total = post[i, :].sum()
        post[i, :] = post[i, :] / total
        ll += np.log(total)

    return post, ll


def gaussian(x: np.ndarray, mean: np.ndarray, var: float) -> float:
    """Computes the 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 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 np.exp(log_prob)
```

Test results

See full output

CORRECT

See full output

Solution:

The E-step update is:

$$p(j \mid t) = \frac{p_j N\left(x; \mu^{(j)}, \sigma_j^2 I\right)}{\sum_{j=1}^{K} p_j N\left(x; \mu^{(j)}, \sigma_j^2 I\right)}$$

The log-likelihood computation is:

$$\sum_{t=1}^{n} \log\left(\sum_{j=1}^{K} p_j N\left(x^{(t)}; \mu^{(j)}, \sigma_j^2 I\right)\right)$$

Submit You have used 2 of 20 attempts

Answers are displayed within the problem

Implementing M-step

1.0/1.0 point (graded)
Write a function `mstep` that performs the M-step of the EM algorithm

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

```
            for all components for all examples

    Returns:
        GaussianMixture: the new gaussian mixture
    """
    n_data, dim = X.shape
    n_data, k = post.shape
    n_clusters = np.einsum("ij -> j", post)
    weight = n_clusters / n_data
    mu =  post.T @ X / n_clusters.reshape(k, 1)
    var = np.zeros(k)
    for i in range(k):
        var[i] = np.sum(post[:,i].T @ (X - mu[i])**2 / (n_clusters[i] * dim))
    return GaussianMixture(mu, var, weight)

```

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

Correct

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

    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
    """
    n, d = X.shape
    _, K = post.shape

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

    mu = np.zeros((K, d))
    var = np.zeros(K)

    for j in range(K):
        # Computing mean
        mu[j, :] = (X * post[:, j, None]).sum(axis=0) / n_hat[j]
        # Computing variance
        sse = ((mu[j] - X)**2).sum(axis=1) @ post[:, j]
        var[j] = sse / (d * n_hat[j])

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

Test results

CORRECT

See full output

See full output

Solution:

The M-step update is:

$$
\begin{aligned}
\hat{n}_j &= \sum_{t=1}^{n} p(j \mid t) \\
\hat{p}_j &= \frac{\hat{n}_j}{n} \\
\hat{\mu^{(j)}} &= \frac{1}{\hat{n}_j} \sum_{t=1}^{n} p(j \mid t)\, x^{(t)} \\
\hat{\sigma}_j^2 &= \frac{1}{d\hat{n}_j} \sum_{t=1}^{n} p(j \mid t)\, \|x^{(t)} - \hat{\mu}^{(j)}\|^2
\end{aligned}
$$

Submit You have used 3 of 20 attempts

Answers are displayed within the problem

Implementing run

1.0/1.0 point (graded)

Write a function `run` that runs the EM algorithm. The convergence criterion you should use is described above.

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

```
    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
```

```
    """
    old_log_likelihood = None
    while 1:
        post, new_log_likelihood = estep(X, mixture)
        mixture = mstep(X, post)
        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 8 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 / 3. Expectation–maximization algorithm