Course › Unit 4 Unsupervised Learning (2 weeks) › Project 4: Collaborative Filtering via Gaussian Mixtures › 3. Expectation–maximization algorithm

Audit Access Expires May 11, 2020

You lose all access to this course, including your progress, on May 11, 2020.

3. Expectation–maximization algorithm

Data Generation Models

0:00 / 0:00 1.25x

Video

Download video file

Transcripts

Download SubRip (.srt) file

Download Text (.txt) file

Recap of the EM algorithm

0:00 / 0:00 1.25x

Video
Download video file

Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

Gaussian Mixtures Models for Matrix Completion

0:00 / 0:00 1.25x

Video
Download video file

Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

Gaussian Mixtures Models for Matrix Completion Continued

0:00 / 0:00 1.25x

Video
Download video file

Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

EM algorithm for Matrix Completion

0:00 / 0:00 1.25x

Video
Download video file

Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

Recall the Gaussian mixture model presented in class:

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

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

0.0/1.0 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`

```
def estep(X: np.ndarray, mixture: GaussianMixture) -> Tuple[np.ndarray,
    """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
    """
    mu, var, p = mixture.mu, mixture.var, mixture.p
    n, d = X.shape  # 250, 2
    K = p.shape[0]
```

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

Incorrect

```
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

INCORRECT

See full output

Solution:

The E-step update is:

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

The log-likelihood computation is:

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

Submit You have used 1 of 25 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`

```
def mstep(X: np.ndarray, post: np.ndarray) -> GaussianMixture:
    """M-step: Updates the gaussian mixture by maximizing the log-likel
    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  # (250, 2)
    K = post.shape[1]  # 3
    denom = 1 / np.sum(post, axis=0)
```

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

See full output

CORRECT

See full output

Solution:

The M-step update is:

$$\hat{n}_j = \sum_{t=1}^{n} p(j \mid t)$$

$$\hat{p}_j = \frac{\hat{n}_j}{n}$$

$$\hat{\mu^{(j)}} = \frac{1}{\hat{n}_j} \sum_{t=1}^{n} p(j \mid t)\, x^{(t)}$$

$$\hat{\sigma}_j^2 = \frac{1}{d\hat{n}_j} \sum_{t=1}^{n} p(j \mid t) \left\| x^{(t)} - \hat{\mu}^{(j)} \right\|^2$$

Submit

You have used 2 of 25 attempts

Answers are displayed within the problem

Implementing run

0.0/1.0 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

```
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
    """
```

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

Unanswered

```
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
```

Submit

You have used 0 of 25 attempts

Answers are displayed within the problem

Discussion

Hide Discussion

Topic: Unit 4 Unsupervised Learning (2 weeks) :Project 4: Collaborative Filtering via Gaussian Mixtures / 3. Expectation–maximization algorithm

Add a Post

Show all posts | by recent activity

My test script for section 3: Expectation–maximization algorithm
Hello If you want to test your `estep` and `mstep` functions, I've created a [test script][1]. It ... 7
Pinned Community TA

probability vs. likelihood?
I wonder what is the actual difference between probability and likelihood. The later one is wi... 6

run function problem @STAFF
my run function is v similar to k-means but with log_likelyhood instead of cost still getting thi... 3

Can anyone say what's wrong with my code? (And Staff, can you extend the deadlines? :D) 5

[STAFF] E-step code please
Now that it is past the deadline, can someone please share the E-step code. I have my imple... 2

Error in M-step Solution?
Found one possible minor error in the solution - The posted solution to M-Step is: def mstep(... 1

The solutions are not fully vectored! :(Can anyone share the fully vectorised version of the EM?
Hi, I was really anxious to see the solution today cause I did not manage to fully vectorised. N... 1

[STAFF] Unable to resolve why E-step solution is incorrect. please help
I've implemented the computation correct as per the suggestions in comments in **compute... 6

[staff] Run step: E and M are correct, Run incorrect; log-likelihoods on next page correct
I'm checking against 10e-6. It's such a simple loop. It feels like I'm just trying to flip on a light s... 1

Staff: Please extend deadline
It would be a great support for the learners if you could extend the deadline by a day at least.... 4

Are people getting credit for the E step only be reverse-engineering a bug in the grader?
Is it not the case that, in the denominator of the exponential expression for the multivariate ... 2

FYI, do not use "dtype = np.float16" in this project