CS461 HW5

John Bailon

December 16, 2024

1 EM Algorithm

1.1 Log-Likelihood

A regular Gaussian distribution is represented below

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

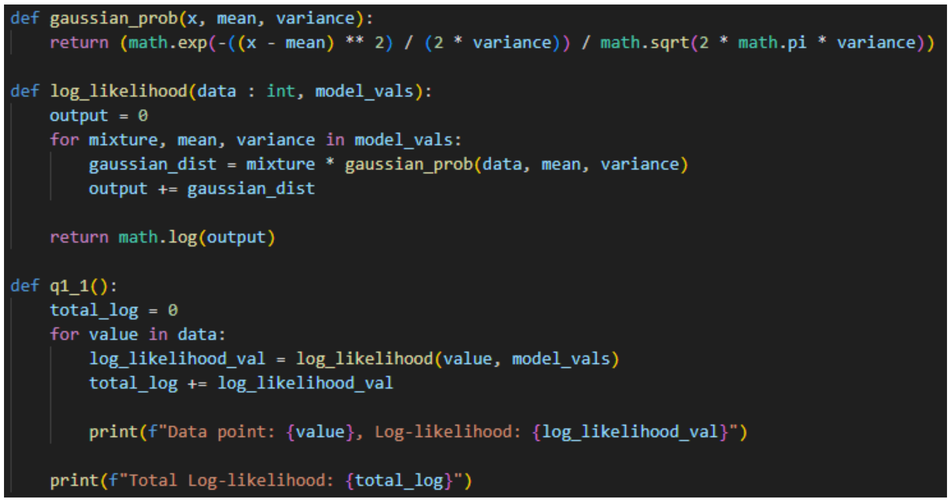
For a Gaussian mixture model, we introduce weight coefficients for each model and their respective mean and variance.

$$f_X(x) = \pi_0 \cdot \mathcal{N}(\mu_0, \sigma_0^2) + \pi_1 \cdot \mathcal{N}(\mu_1, \sigma_1^2)$$

To find the log-likelihood for each point, we will first sum the Gaussian probabilities of each model and then take the log.

$$logL = log(\sum_k \pi_k \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} exp(-\frac{(x-\mu_k)^2}{2\sigma_k^2}))$$

Using the code below, I got the following values:
d1: -2.0939
d2: -1.4851
d3: -1.4851
d4: -2.0939
Total log-likelihood: -7.1582



```python
def gaussian_prob(x, mean, variance):
    return (math.exp(-((x - mean) ** 2) / (2 * variance)) / math.sqrt(2 * math.pi * variance))

def log_likelihood(data : int, model_vals):
    output = 0
    for mixture, mean, variance in model_vals:
        gaussian_dist = mixture * gaussian_prob(data, mean, variance)
        output += gaussian_dist

    return math.log(output)

def q1_1():
    total_log = 0
    for value in data:
        log_likelihood_val = log_likelihood(value, model_vals)
        total_log += log_likelihood_val

        print(f"Data point: {value}, Log-likelihood: {log_likelihood_val}")

    print(f"Total Log-likelihood: {total_log}")
```

Figure 1: 1.1 Code for Log-Likelihood

1.2 E-Step

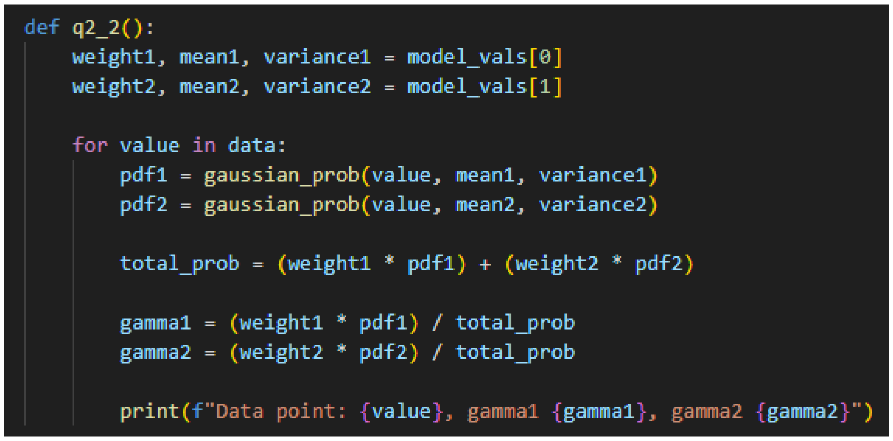
For the E-step, we have to calculate the responsibility that each group K assumes for a given data point. The responsibility of group k for point n is given by the following.

$$\gamma_n k = \frac{\pi_k \cdot \mathcal{N}(x_i|\mu_k, \sigma_k^2)}{\sum_k \pi_k \cdot \mathcal{N}(x_i|\mu_k, \sigma_k^2)}$$

For each data point, we will calculate the Gaussian probability, multiply it by its weight, and divide by the sum of both models.

Using the code below, I got the following values:

Data	γ_0	γ_1
d1	0.0180	0.9820
d2	0.1192	0.8808
d3	0.8808	0.1192
d4	0.9820	0.0180



```python
def q2_2():
    weight1, mean1, variance1 = model_vals[0]
    weight2, mean2, variance2 = model_vals[1]

    for value in data:
        pdf1 = gaussian_prob(value, mean1, variance1)
        pdf2 = gaussian_prob(value, mean2, variance2)

        total_prob = (weight1 * pdf1) + (weight2 * pdf2)

        gamma1 = (weight1 * pdf1) / total_prob
        gamma2 = (weight2 * pdf2) / total_prob

        print(f"Data point: {value}, gamma1 {gamma1}, gamma2 {gamma2}")
```

Figure 2: 1.2 Code for E-Step

1.3 M-Step

Finally, we will update the parameters, weight coeff, mean, and variance for each model.

The weight is updated using:

$$\pi_k(t+1) = \frac{\sum_n \gamma_{nk}(\theta_t)}{N}$$

$$\pi_0(t+1) = \pi_1(t+1) = \frac{0.0180 + 0.1192 + 0.8808 + 0.9820}{4} = \frac{1}{2}$$

The mean is updated using:

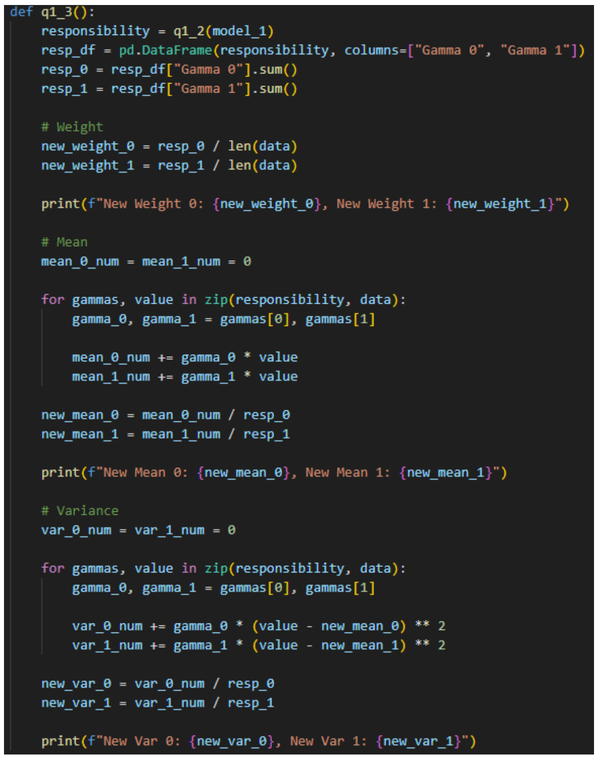
$$\mu_k(t+1) = \frac{\sum_N \gamma_{nk} * x_n}{\sum_N \gamma_{nk}}$$

The variance is updated using:

$$\sigma_k^2(t+1) = \frac{\sum_N \gamma_{nk} * (x_n - \mu_k)^2}{\sum_N \gamma_{nk}}$$

Using the code below, I got the following values:

Parameter	Model 0	Model 1
π	0.5	0.5
μ	-1.3448	1.3448
σ^2	0.6914	0.6914



```python
def q1_3():
    responsibility = q1_2(model_1)
    resp_df = pd.DataFrame(responsibility, columns=["Gamma 0", "Gamma 1"])
    resp_0 = resp_df["Gamma 0"].sum()
    resp_1 = resp_df["Gamma 1"].sum()

    # Weight
    new_weight_0 = resp_0 / len(data)
    new_weight_1 = resp_1 / len(data)

    print(f"New Weight 0: {new_weight_0}, New Weight 1: {new_weight_1}")

    # Mean
    mean_0_num = mean_1_num = 0

    for gammas, value in zip(responsibility, data):
        gamma_0, gamma_1 = gammas[0], gammas[1]

        mean_0_num += gamma_0 * value
        mean_1_num += gamma_1 * value

    new_mean_0 = mean_0_num / resp_0
    new_mean_1 = mean_1_num / resp_1

    print(f"New Mean 0: {new_mean_0}, New Mean 1: {new_mean_1}")

    # Variance
    var_0_num = var_1_num = 0

    for gammas, value in zip(responsibility, data):
        gamma_0, gamma_1 = gammas[0], gammas[1]

        var_0_num += gamma_0 * (value - new_mean_0) ** 2
        var_1_num += gamma_1 * (value - new_mean_1) ** 2

    new_var_0 = var_0_num / resp_0
    new_var_1 = var_1_num / resp_1

    print(f"New Var 0: {new_var_0}, New Var 1: {new_var_1}")
```

Figure 3: 1.3 Code for M-Step

1.4 Updated Log-likelihood

Using the values found in 1.3 and the code from 1.1, I recomputed the log-likelihood below. The total log-likelihood has increased.

d1: -1.7376
d2: -1.4933
d3: -1.4933
d4: -1.7376
Total log-likelihood: -6.4619

2 Exact v. Approximate inference

2.1 Variable Elimination

Using the structure of the provided network and conditional probability and marginalization, we have:

$$P(Cloudy|Sprinkler = T, WetGrass = T) = \alpha \sum_{rain} P(Cloudy, Sprinkler, Rain, WetGrass)$$

We will use normalization later to find the posterior probability. Next, we substitute the joint probability for its Bayesian network representation

$$= \alpha \sum_{rain} P(Cloudy) * P(Sprinkler|Cloudy) * P(Rain|Cloudy) * P(WetGrass|Sprinkler, rain)$$

$$= \alpha * P(Cloudy) * P(Sprinkler|Cloudy) \sum_{rain} P(Rain|Cloudy) * P(WetGrass|Sprinkler, rain)$$

For Cloudy = T

$$= \alpha * 0.5 * 0.1 * ((0.8 * 0.99) + (0.2 * 0.9))$$
$$= 0.0486\alpha$$

For Cloudy = F

$$= \alpha * 0.5 * 0.5 * ((0.2 * 0.99) + (0.8 * 0.9))$$
$$= 0.2295\alpha$$

Finally, normalize

$$P(Cloudy|Sprinkler = T, WetGrass = T) = \frac{0.0486}{0.0486 + 0.2295} = 17.48\%$$

2.2 Gibbs Sampling

For Gibbs sampling, we will assign all variables in the network to arbitrary values, fixing the evidence variables, sprinkler, and wet grass, to true. Next, we will generate samples for both cloudy and rainy by calculating their probability conditional on their Markov blankets. The general form is

$$P(Target|MarkovBlanket) = \alpha P(Target|Parent) * P(Children|CoparentandTarget)$$

For Cloudy:

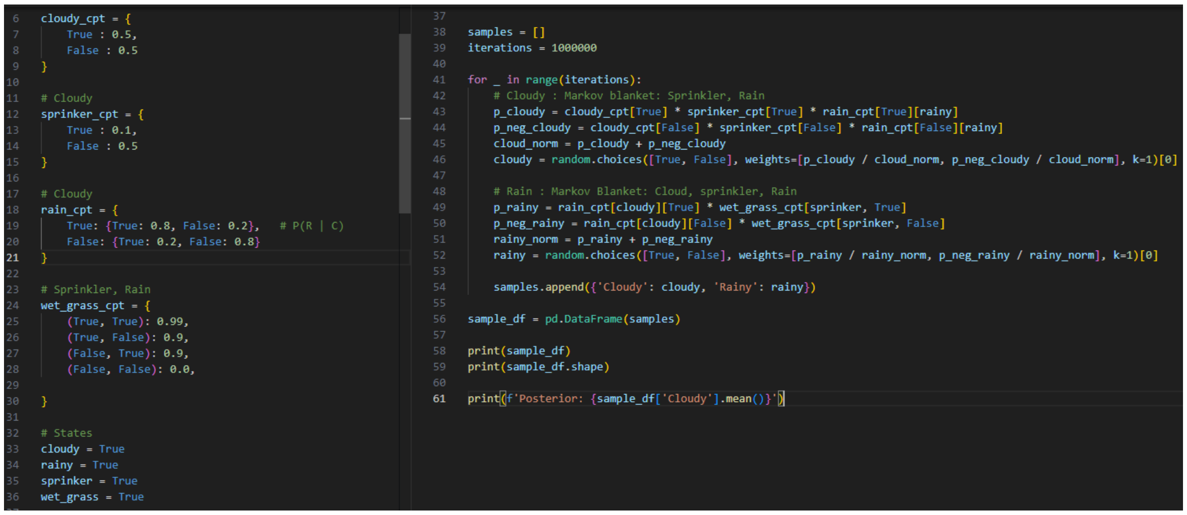
$$P(Cloudy|Sprinkler, Rainy) = \alpha P(Cloudy) * P(Sprinkler|Cloudy) * P(Rainy|Cloudy)$$

For Rainy:

$$P(Rainy|Cloudy, wetgrass, Sprinkler) = \alpha P(Rainy|Cloudy) * P(WetGrass|Sprinkler, Rainy)$$

We will assign cloudy and rainy based on the computed probability and repeat. The mean of the generated sample will approximate the prior.

Below is the code I used to solve this problem. After 1,000,000 iterations, the prior is estimated to be 17.53%.



```python
cloudy_cpt = {
    True : 0.5,
    False : 0.5
}

# Cloudy
sprinker_cpt = {
    True : 0.1,
    False : 0.5
}

# Cloudy
rain_cpt = {
    True: {True: 0.8, False: 0.2},   # P(R | C)
    False: {True: 0.2, False: 0.8}
}

# Sprinkler, Rain
wet_grass_cpt = {
    (True, True): 0.99,
    (True, False): 0.9,
    (False, True): 0.9,
    (False, False): 0.0,

}

# States
cloudy = True
rainy = True
sprinker = True
wet_grass = True
```

```python

samples = []
iterations = 1000000

for _ in range(iterations):
    # Cloudy : Markov blanket: Sprinkler, Rain
    p_cloudy = cloudy_cpt[True] * sprinker_cpt[True] * rain_cpt[True][rainy]
    p_neg_cloudy = cloudy_cpt[False] * sprinker_cpt[False] * rain_cpt[False][rainy]
    cloud_norm = p_cloudy + p_neg_cloudy
    cloudy = random.choices([True, False], weights=[p_cloudy / cloud_norm, p_neg_cloudy / cloud_norm], k=1)[0]

    # Rain : Markov Blanket: Cloud, sprinkler, Rain
    p_rainy = rain_cpt[cloudy][True] * wet_grass_cpt[sprinker, True]
    p_neg_rainy = rain_cpt[cloudy][False] * wet_grass_cpt[sprinker, False]
    rainy_norm = p_rainy + p_neg_rainy
    rainy = random.choices([True, False], weights=[p_rainy / rainy_norm, p_neg_rainy / rainy_norm], k=1)[0]

    samples.append({'Cloudy': cloudy, 'Rainy': rainy})

sample_df = pd.DataFrame(samples)

print(sample_df)
print(sample_df.shape)

print(f'Posterior: {sample_df['Cloudy'].mean()}')
```

Figure 4: 2.2 Gibbs Sampling Implementation

3 VAE Lower Bound (ELBO)

3.1 Derive the Inequality

First we start with the marginal log-likelihood:

$$\log P_\theta(x_i) = log \sum_z P_\theta(x_i, Z)$$

Next, we add an approximate posterior by multiplying and dividing,

$$= log \sum_z q_\phi(Z|x_i) * \frac{P_\theta(x_i, Z)}{q_\phi(Z|x_i)}$$

$$= log E_{q_\phi(Z|x_i)}[\frac{P_\theta(x_i, Z)}{q_\phi(Z|x_i)}]$$

Next, we use Jensen's inequality, which states:

$$E[\log X] \leq \log E[X]$$

We can rewrite the above as an inequality,

$$\geq E_{q_\phi(Z|x_i)}[log \frac{P_\theta(x_i, Z)}{q_\phi(Z|x_i)}]$$

Next, manipulate the equation using log properties,

$$\geq E_{q_\phi(Z|x_i)}[log(P_\theta(x_i, Z)) - log(q_\phi(Z|x_i))]$$

Expand the joint probability, $P_\theta(x_i, Z)$ and once again use log properties

$$\geq E_{q_\phi(Z|x_i)}[log(P_\theta(x_i|Z)) + log(P_\theta(Z)) - log(q_\phi(Z|x_i))]$$

Expand the terms within the expected value, and factor out -1 from the last two terms

$$\geq E_{q_\phi(Z|x_i)}[log(P_\theta(x_i|Z))] - E_{q_\phi(Z|x_i)}[log(q_\phi(Z|x_i)) - log(P_\theta(Z))]$$

Here we can substitute the second term for the KL divergence and reorder to complete the proof.

$$= -D_{KL} q_\phi(z|x_i)||p_\theta(z) + E_{q_\phi(Z|x_i)}[log(P_\theta(x_i|Z))])$$

4 RBM Movie Recommendation System

4.1 Bipolar Coding

The RBM in Figure 2 represents a recommendation system that encodes the preference for movies. A binary coding is not sufficient for this system as it needs to encode both like and dislike of a movie. Thus, a bipolar coding is more appropriate as like and dislike can be represented by positive and negative values, and unwatched movies can be represented with zeroes.

4.2 Modify RBM Conditional Probability

The derived conditional probability does not work as the sigmoid function ranges from 0 to 1. Since we are using a bipolar coding, we need our function to range from -1 to 1.

We follow the same derivation as in class. However, instead of normalizing with h = 0, we must normalize with h = -1.

$$P(h = 1|m1, m2, m3 = \alpha \cdot exp(m^t W[:, 1] + C_1)$$

$$P(h = -1|m1, m2, m3 = \alpha \cdot exp(-m^t W[:, 1] + C_1)$$

Normalizing these functions we can use hyperbolic identities to derive

$$P(h = 1|m1, m2, m3 = \frac{1 + \tanh exp(m^t W[:, 1] + C_1)}{2}$$

$$P(m = 1|m1, m2, m3 = \frac{1 + \tanh exp(W[1, :]h + b_1)}{2}$$

4.3 Predict Movie Preference