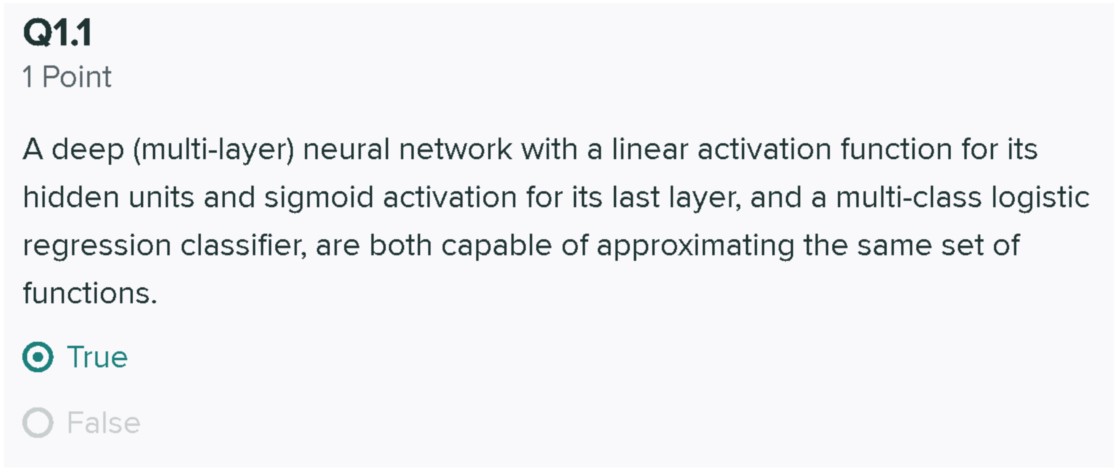
Commonly Wrong Problems

[Note: We talk through each option and why they're correct/wrong, so watch the recording if you're confused about a specific option not explained (in detail) on the doc]

- **QnA #3 Q1.1**



Q1.1

1 Point

A deep (multi-layer) neural network with a linear activation function for its hidden units and sigmoid activation for its last layer, and a multi-class logistic regression classifier, are both capable of approximating the same set of functions.

[✓ True]

False

Neural Network vs Logistic Regression

Sigmoid unit: $$o(\mathbf{x}) = \sigma(w_0 + \sum_i w_i x_i)$$

Logistic function as a Graph

1-Hidden layer, 1 output NN: $$o(\mathbf{x}) = \sigma\left(w_0 + \sum_h w_h \underbrace{\sigma(w_0^h + \sum_i w_i^h x_i)}_{}\right)$$

$$\text{Output, } o(\mathbf{x}) = \sigma(w_0 + \sum_i w_i X_i) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))}$$

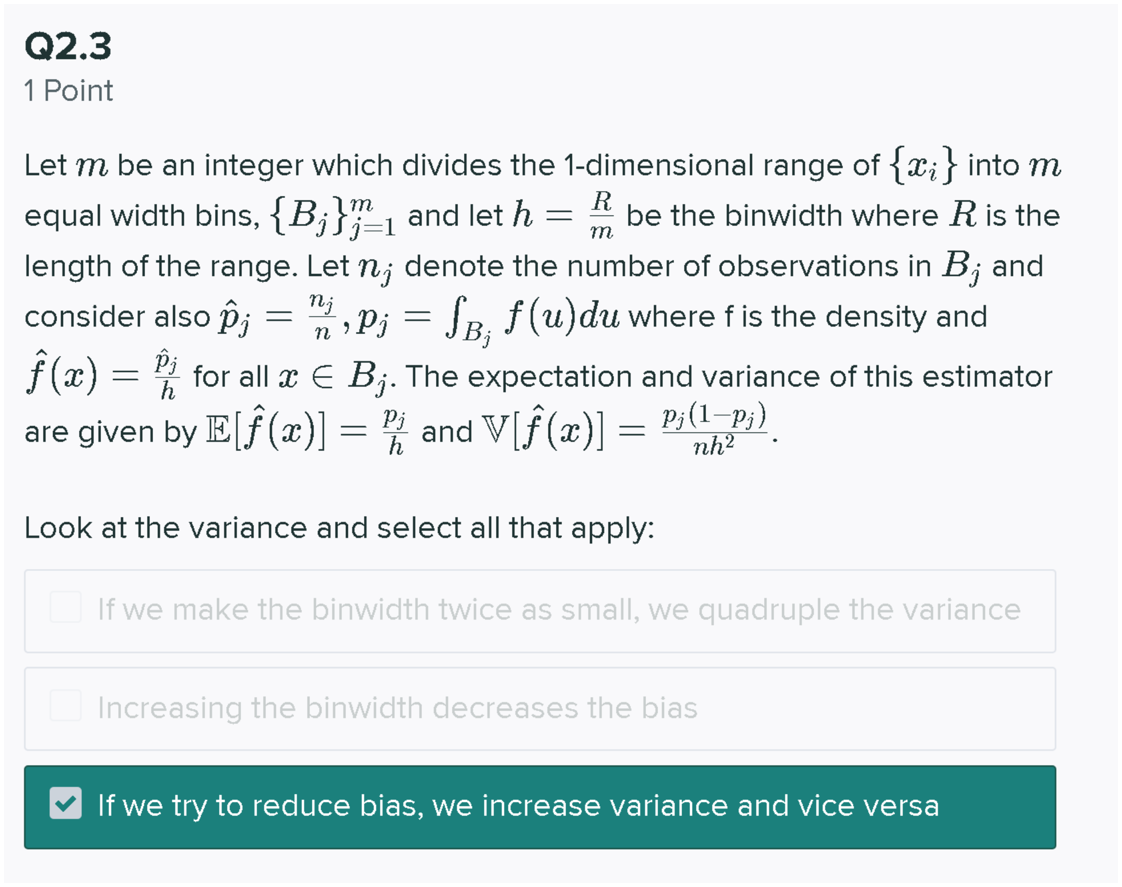
Point 1: Multi-layer

- Nested linear combinations can always be re-expressed as a single linear combination -> multi-layer = single layer

Point 2: Multi-class

- Each output label probability in the neural network is independent of the other outputs -> equivalent to applying logistic regression for each class

- **QnA #3 2.3 + 2.4**



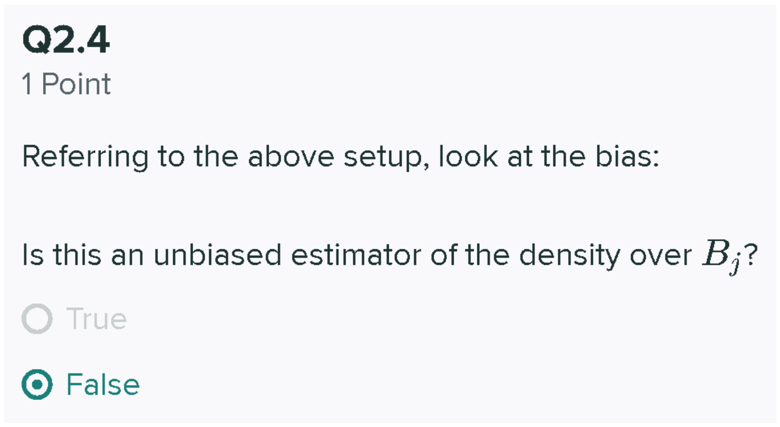
Q2.3

1 Point

Let m be an integer which divides the 1-dimensional range of $\{x_i\}$ into m equal width bins, $\{B_j\}_{j=1}^{m}$ and let $h = \frac{R}{m}$ be the binwidth where R is the length of the range. Let n_j denote the number of observations in B_j and consider also $\hat{p}_j = \frac{n_j}{n}, p_j = \int_{B_j} f(u)du$ where f is the density and $\hat{f}(x) = \frac{\hat{p}_j}{h}$ for all $x \in B_j$. The expectation and variance of this estimator are given by $\mathbb{E}[\hat{f}(x)] = \frac{p_j}{h}$ and $\mathbb{V}[\hat{f}(x)] = \frac{p_j(1-p_j)}{nh^2}$.

Look at the variance and select all that apply:

- [ ] If we make the binwidth twice as small, we quadruple the variance
- [ ] Increasing the binwidth decreases the bias
- [✓] If we try to reduce bias, we increase variance and vice versa



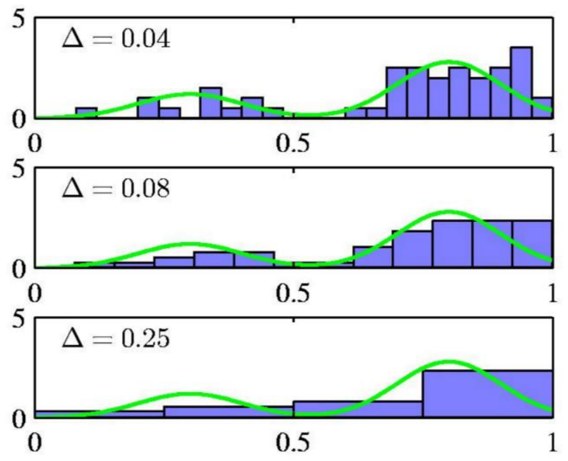
Q2.4

1 Point

Referring to the above setup, look at the bias:

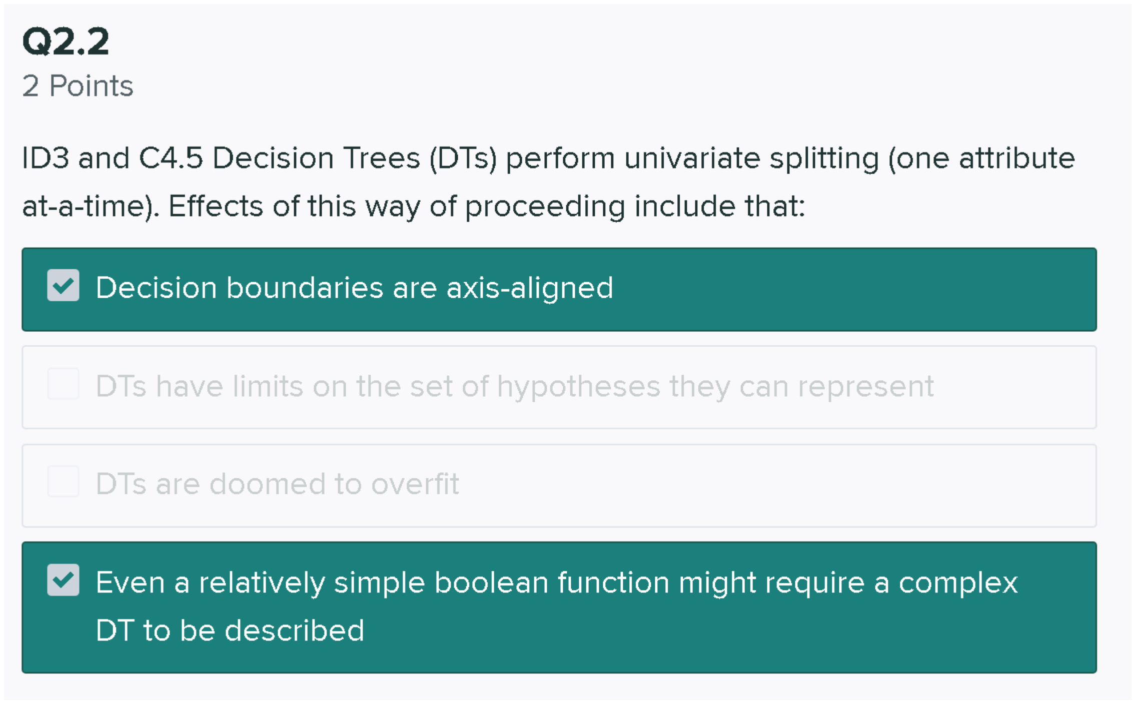
Is this an unbiased estimator of the density over B_j?

True [✓ False]



- Bias: Difference between expected value and true value

- **QnA #4 Q2.2 + Q2.3**

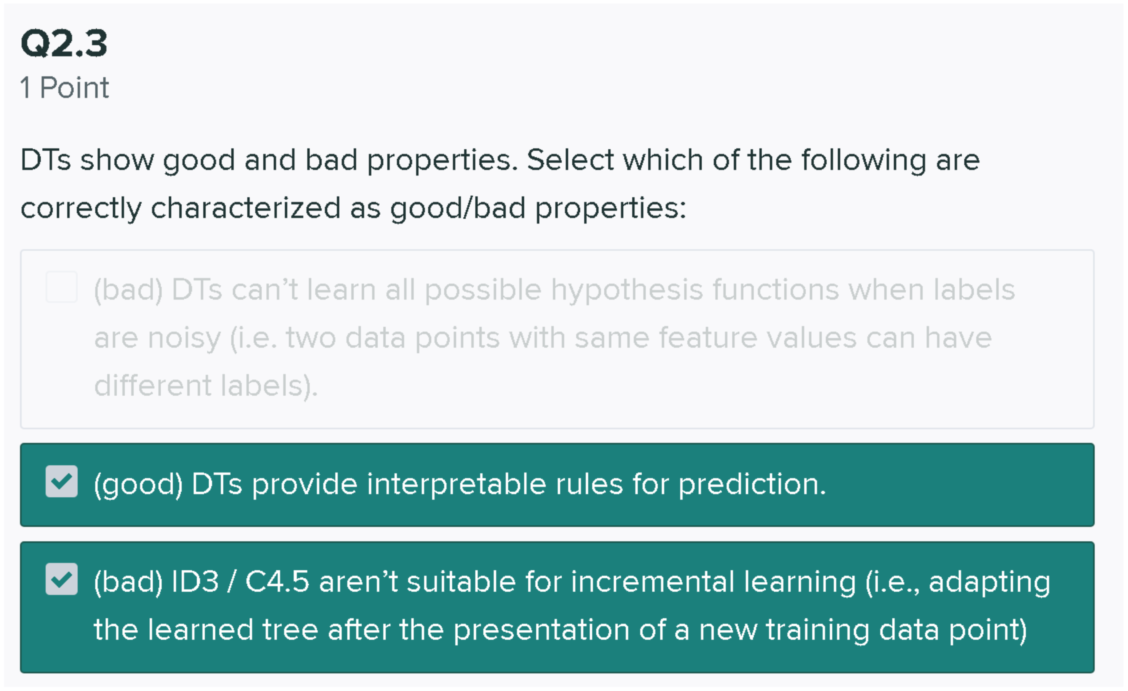


Q2.2
2 Points

ID3 and C4.5 Decision Trees (DTs) perform univariate splitting (one attribute at-a-time). Effects of this way of proceeding include that:

- [✓] Decision boundaries are axis-aligned
- [ ] DTs have limits on the set of hypotheses they can represent
- [ ] DTs are doomed to overfit
- [✓] Even a relatively simple boolean function might require a complex DT to be described

- No limit: Can express any function of the input features (worst-case: one path for each training point)



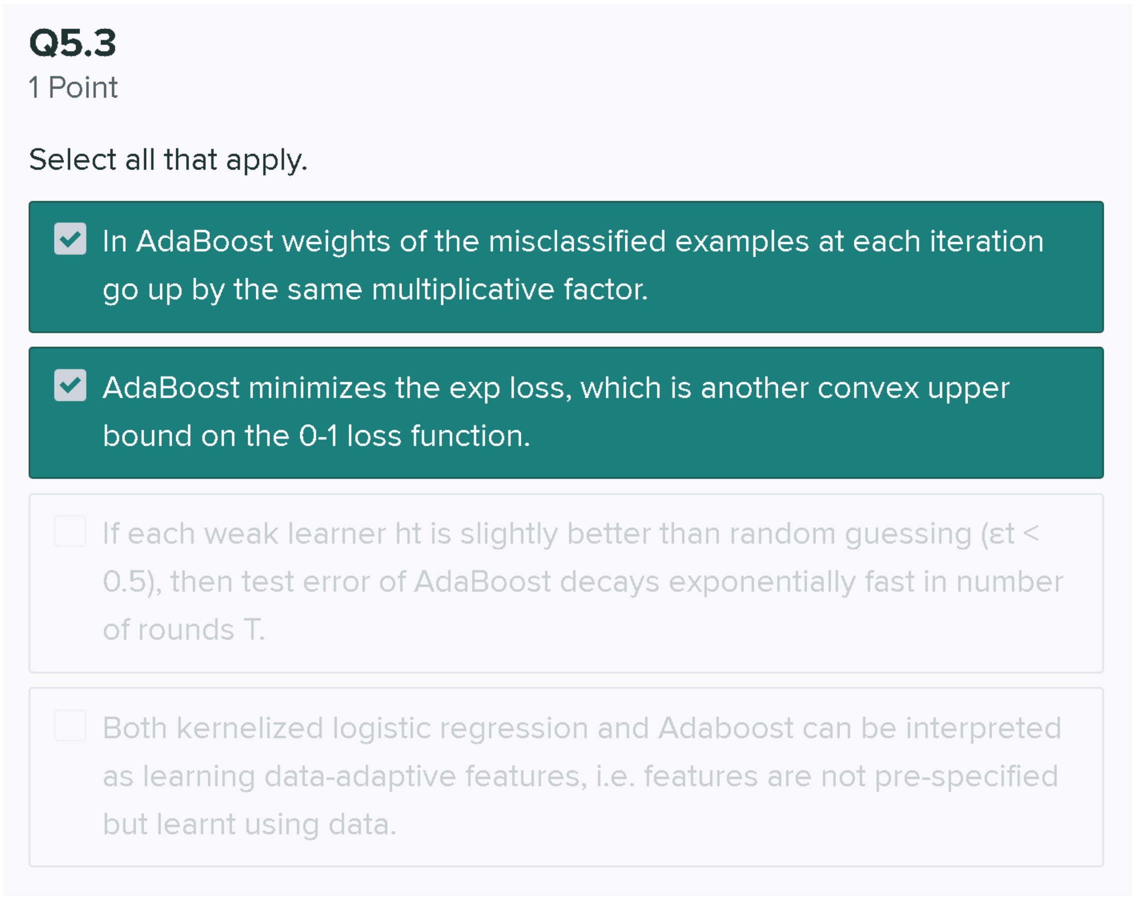
Q2.3
1 Point

DTs show good and bad properties. Select which of the following are correctly characterized as good/bad properties:

- [ ] (bad) DTs can't learn all possible hypothesis functions when labels are noisy (i.e. two data points with same feature values can have different labels).
- [✓] (good) DTs provide interpretable rules for prediction.
- [✓] (bad) ID3 / C4.5 aren't suitable for incremental learning (i.e., adapting the learned tree after the presentation of a new training data point)

- DT doesn't need perfect classification (it just needs to pick the best decision with noisy labels)

- **QnA #4 Q2.2 + Q2.3**



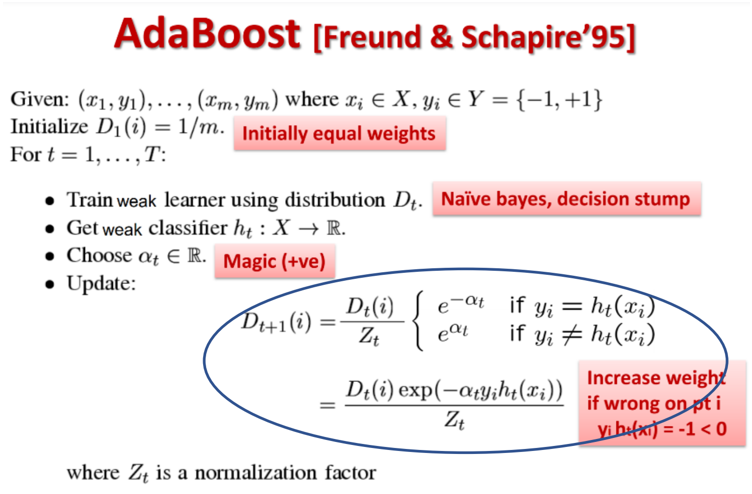
Q5.3

1 Point

Select all that apply.

- [✓] In AdaBoost weights of the misclassified examples at each iteration go up by the same multiplicative factor.
- [✓] AdaBoost minimizes the exp loss, which is another convex upper bound on the 0-1 loss function.
- [ ] If each weak learner ht is slightly better than random guessing (εt < 0.5), then test error of AdaBoost decays exponentially fast in number of rounds T.
- [ ] Both kernelized logistic regression and Adaboost can be interpreted as learning data-adaptive features, i.e. features are not pre-specified but learnt using data.

train, not test error



AdaBoost [Freund & Schapire'95]

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$. **Initially equal weights**

For $t = 1, \dots, T$:

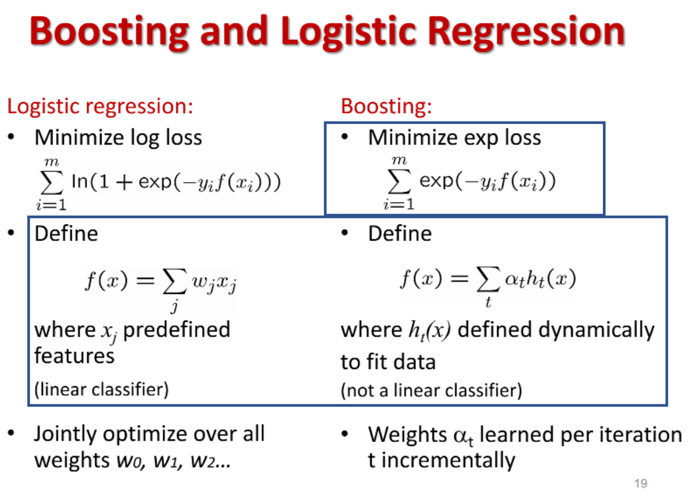
- Train weak learner using distribution D_t. **Naïve bayes, decision stump**
- Get weak classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$. **Magic (+ve)**
- Update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$= \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Increase weight if wrong on pt i $y_i h_t(x_i) = -1 < 0$

where Z_t is a normalization factor



Boosting and Logistic Regression

Logistic regression:	Boosting:
• Minimize log loss $\sum_{i=1}^{m} \ln(1 + \exp(-y_i f(x_i)))$	• Minimize exp loss $\sum_{i=1}^{m} \exp(-y_i f(x_i))$
• Define $f(x) = \sum_j w_j x_j$ where x_j predefined features (linear classifier)	• Define $f(x) = \sum_t \alpha_t h_t(x)$ where $h_t(x)$ defined dynamically to fit data (not a linear classifier)
• Jointly optimize over all weights w_0, w_1, w_2...	• Weights α_t learned per iteration t incrementally

19

Expectation Maximization (EM) with Gaussian Mixture Models (GMM)

Let z be a multinomial random latent variable with components $z_1, z_2, ..., z_k$, where each component takes on 0 or 1 *i.e.* P$(z_j = 1)$ is the probability that a point comes from gaussian distribution j.

Let $\lambda = \mu_1, \mu_2, ..., \mu_k, \Sigma_1, ..., \Sigma_k, \pi_1, ..., \pi_k$ where $\pi_j =$ P$(z_j$=1).

The log likelihood $\ell(\lambda|x_1, x_2, ..., x_m) = \sum_{i=1}^{m} logP(x_i|\lambda) = \sum_{i=1}^{m} log \sum_{j=1}^{k} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)$.

(Note: These are the more standard notations used for EM. It differs slightly from class notation, with the correspondence as: $z = y$, $\mu = p$)

(a) E-step: Calculate the posterior probability $P(z_j = 1|x_i, \lambda)$ $\forall i, j$.

$P(z_j = 1|x_i, \lambda)$

$= \frac{p(z_j=1|\pi_j)p(x_i|z_j=1,\mu_j,\Sigma_j)}{p(x_i|\lambda)}$ [Bayes Rule]

$= \frac{\pi_j \mathcal{N}(x_i|\mu_j,\Sigma_j)}{\sum_{l=1}^{k} \pi_l \mathcal{N}(x_i|\mu_l,\Sigma_l)}$ [Marginalization for denominator]

(Note: In lecture, Aarti removed the denominator and represented the proportional probability with the numerator)

(b) M-step: Apply MLE and update the parameters π_j, μ_j, Σ_j $\forall j$.

For example, we solve for μ_j by taking the derivative of log likelihood w.r.t μ_j and setting it to 0.

$\frac{\partial \ell}{\partial \mu_j} = \frac{\partial \ell}{\partial \mu_j} \sum_{i=1}^{m} log \sum_{l=1}^{k} \pi_l \mathcal{N}(x_i|\mu_l, \Sigma_l)$ [Log likelihood function]

$= \sum_{i=1}^{m} \frac{1}{\sum_{l=1}^{k} \pi_l \mathcal{N}(x_i|\mu_l,\Sigma_l)} \frac{\partial \ell}{\partial \mu_j} \sum_{l=1}^{k} \pi_l \mathcal{N}(x_i|\mu_l, \Sigma_l)$ [Differentiation rule: $\frac{\partial}{\partial x} ln(u(x)) = \frac{1}{u(x)} * u'(x)$]

$= \sum_{i=1}^{m} \frac{1}{\sum_{l=1}^{k} \pi_l \mathcal{N}(x_i|\mu_l,\Sigma_l)} \frac{\partial \ell}{\partial \mu_j} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)$ [Eliminating terms with no u_j]

$= \sum_{i=1}^{m} \frac{\mathcal{N}(x_i|\mu_j,\Sigma_j)\pi_j}{\sum_{l=1}^{k} \pi_l \mathcal{N}(x_i|\mu_l,\Sigma_l)} \frac{\partial \ell}{\partial \mu_j} \frac{(x_i-\mu_j)^2}{2\Sigma_j}$ [Exponential rule: $\frac{\partial}{\partial x} e^{u(x)} = e^{u(x)} * u'(x)$]

$= \sum_{i=1}^{m} P(z_j = 1|x_i, \lambda) \frac{\partial \ell}{\partial \mu_j} \frac{(x_i-\mu_j)^2}{2\Sigma_j}$ [Substitute from E-step]

$= \sum_{i=1}^{m} P(z_j = 1|x_i, \lambda)\Sigma_j^{-1}(x_i - \mu_j)$ [Derivative of log gaussian density function]

Setting this to 0, you get: $\mu_j = \frac{\sum_{i=1}^{m} P(z_j=1|x_i,\lambda)x_i}{\sum_{i=1}^{m} P(z_j=1|x_i,\lambda)}$

Similar calculation produces Σ_j and π_j.