T-61.5140 Machine Learning: Advanced Probabilistic Methods, Project Work, S2016

Unto Kuuranne 349583, Clemens Westrup 341057

April 29th 2016

1 Mathematical description for a mixture model with two linear components

Note: We will denote the parameters as $\theta = (\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \sigma_1, \sigma_2, w)$.

1.1 Full posterior likelihood

$$p(\boldsymbol{y}, \boldsymbol{z}, \theta \mid \boldsymbol{x}) = p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}, \theta) p(\theta) \tag{1}$$

1.2 Full log posterior likelihood

$$\begin{aligned}
&\log p(\boldsymbol{y}, \boldsymbol{z}, \theta \mid \boldsymbol{x}) = \log p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}, \theta) + \log p(\theta) \\
&= \sum_{t=1}^{T} \Big\{ z_t \log \left[w \mathcal{N}(y_t \mid \boldsymbol{\phi}_1 \boldsymbol{x}_t, \sigma_1^2) \right] + (1 - z_t) \log \left[(1-w) \mathcal{N}(y_t \mid \boldsymbol{\phi}_2 \boldsymbol{x}_t, \sigma_2^2) \right] \Big\} \\
&\quad + \log \left(\frac{\Gamma(\alpha_w + \beta_w)}{\Gamma(\alpha_w)\Gamma(\beta_w)} \right) + (\alpha_w - 1) \log(w) + (\beta_w - 1) \log(1 - w) \\
&\quad - 0.5P \log(2\pi\sigma_1^2) - 0.5 \log |\boldsymbol{\Sigma}_\phi| - \frac{1}{2\sigma_1^2} (\boldsymbol{\phi}_1 - \boldsymbol{\mu}_\phi)^T \boldsymbol{\Sigma}_\phi (\boldsymbol{\phi}_1 - \boldsymbol{\mu}_\phi) \\
&\quad - 0.5P \log(2\pi\sigma_2^2) - 0.5 \log |\boldsymbol{\Sigma}_\phi| - \frac{1}{2\sigma_2^2} (\boldsymbol{\phi}_2 - \boldsymbol{\mu}_\phi)^T \boldsymbol{\Sigma}_\phi (\boldsymbol{\phi}_2 - \boldsymbol{\mu}_\phi) \\
&\quad + \log \left(\frac{\beta_{\sigma^2}^{\alpha_{\sigma^2}}}{\Gamma(\alpha_{\sigma^2})} \right) - (\alpha_{\sigma^2} + 1)\sigma_1^2 - \frac{\beta_{\sigma^2}}{\sigma_1^2} \\
&\quad + \log \left(\frac{\beta_{\sigma^2}^{\alpha_{\sigma^2}}}{\Gamma(\alpha_{\sigma^2})} \right) - (\alpha_{\sigma^2} + 1)\sigma_2^2 - \frac{\beta_{\sigma^2}}{\sigma_2^2}
\end{aligned} \tag{2}$$

2 Derivation of the EM update equations for the parameters of this model

Responsibilities

From the posterior of the latent variables given the parameters θ we can derive the responsibilities:

$$\begin{aligned}\gamma_t &\equiv p(z_t = 1 \mid \theta_S) = p(z_t = 1 \mid \boldsymbol{\phi}_{1_S}, \boldsymbol{\phi}_{2_S}, \sigma_{1_S}, \sigma_{2_S}, w_S) \\ &= \frac{w_S \mathcal{N}(y_t \mid \boldsymbol{x}_t \boldsymbol{\phi}_{1_S}, \sigma_{1_S}^2)}{w_S \mathcal{N}(y_t \mid \boldsymbol{x}_t \boldsymbol{\phi}_{1_S}, \sigma_{1_S}^2) \;+\; (1 - w_S)\mathcal{N}(y_t \mid \boldsymbol{x}_t \boldsymbol{\phi}_{2_S}, \sigma_{2_S}^2)}\end{aligned} \tag{3}$$

Deriving Q

Now we can derive the expectation of the complete data log-likelihood over the posterior of the latent variables:

$$Q(\cdot) \equiv Q(\boldsymbol{y}, \boldsymbol{x}, \theta, \theta_S) \equiv \boldsymbol{E}_{\boldsymbol{z}|\boldsymbol{y},\boldsymbol{x},\theta_S}[\log\ p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}, \theta)] + \log p(\theta) \tag{4}$$

$$= \sum_{t=1}^{T} \left\{ \gamma_t \log \left[w \mathcal{N}(y_t \mid \boldsymbol{\phi}_1 \boldsymbol{x}_t, \sigma_1^2) \right] + (1 - \gamma_t) \log \left[(1 - w)\mathcal{N}(y_t \mid \boldsymbol{\phi}_2 \boldsymbol{x}_t, \sigma_2^2) \right] \right\} \tag{5}$$

$$+ \log p(\boldsymbol{\phi}_1) + \log p(\boldsymbol{\phi}_2) + \log p(\sigma_1^2) + \log p(\sigma_2^2) + \log p(w) \tag{6}$$

2.1 Differentials for $\boldsymbol{\phi}_j$

$$\frac{\partial}{\partial \boldsymbol{\phi}_1} Q(\cdot) = \sum_{t}^{T} \left\{ \frac{\partial}{\partial \boldsymbol{\phi}_1} \gamma_t \log \left[w \mathcal{N}(y_t \mid \boldsymbol{\phi}_1 \boldsymbol{x}_t, \sigma_1^2) \right] \right\} + \frac{\partial}{\partial \boldsymbol{\phi}_1} \log \mathcal{N}(\boldsymbol{\phi}_1 \mid \boldsymbol{\mu}_\phi, \Sigma_\phi) \tag{7}$$

$$= \sum_{t}^{T} \left\{ \gamma_t \frac{\boldsymbol{x}_t (y_t - \boldsymbol{\phi}_1 \boldsymbol{x}_t)}{\sigma_1^2} \right\} - \frac{\boldsymbol{\phi}_1 - \boldsymbol{\mu}_\phi}{\sigma_1^2 \lambda_\phi} \tag{8}$$

2.2 Updating $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$

$$\boldsymbol{\phi}_1 = \left(\boldsymbol{X}^T (\boldsymbol{\gamma}^T \odot \boldsymbol{X}) + \Sigma_\phi^{-1} \right)^{-1} \left(\boldsymbol{\gamma}(\boldsymbol{y} \odot \boldsymbol{x}) + \boldsymbol{\mu}_\phi \Sigma_\phi^{-1} \right) \tag{9}$$

$$\boldsymbol{\phi}_2 = \left(\boldsymbol{X}^T ((\mathbf{1} - \boldsymbol{\gamma})^T \odot \boldsymbol{X}) + \Sigma_\phi^{-1} \right)^{-1} \left((\mathbf{1} - \boldsymbol{\gamma})(\boldsymbol{y} \odot \boldsymbol{x}) + \boldsymbol{\mu}_\phi \Sigma_\phi^{-1} \right) \tag{10}$$

Where $\odot$ signifies elementwise multiplication.

2.3 Differentials for σ_j^2

$$\frac{\partial}{\partial \sigma_1^2} Q(\cdot) = \frac{\partial}{\partial \sigma_1^2} \sum_{t=1}^{T} \left\{ \gamma_t \log \left[w \mathcal{N}(y_t \mid \boldsymbol{\phi}_1 \boldsymbol{x}_t, \sigma_1^2) \right] \right\} + \frac{\partial}{\partial \sigma_1^2} \log p(\sigma_1^2) + \frac{\partial}{\partial \sigma_1^2} \log p(\boldsymbol{\phi}_1) \tag{11}$$

$$\begin{aligned} &= \frac{1}{2} \sum_{t=1}^{T} \left\{ \gamma_t \frac{(y_t - \boldsymbol{\phi}_1 \boldsymbol{x}_t)^2}{(\sigma_1^2)^2} \right\} - \frac{1}{2} \sum_{t=1}^{T} \left\{ \gamma_t \frac{1}{\sigma_1^2} \right\} + \frac{\beta}{(\sigma_1^2)^2} - \frac{(\alpha + 1)}{\sigma_1^2} \\ &\quad + \frac{1}{2(\sigma_1^2)^2} \left((\boldsymbol{\phi}_1 - \boldsymbol{\mu}_\phi)^T \Sigma_\phi^{-1} (\boldsymbol{\phi}_1 - \boldsymbol{\mu}_\phi) \right) - \frac{1}{2\sigma_1^2} P \end{aligned} \tag{12}$$

2.4 Updating σ_1^2 and σ_2^2

$$\sigma_1^2 = \frac{\sum_{t=1}^{T} \{\gamma_t (y_t - \boldsymbol{\phi}_1 \boldsymbol{x}_t)^2\} + 2\beta + (\boldsymbol{\phi}_1 - \boldsymbol{\mu}_\phi)^T \Sigma_\phi^{-1} (\boldsymbol{\phi}_1 - \boldsymbol{\mu}_\phi)}{\sum_{t=1}^{T} \{\gamma_t\} + 2\alpha + 2 + P} \tag{13}$$

$$\sigma_2^2 = \frac{\sum_{t=1}^{T} \{(1 - \gamma_t)(y_t - \boldsymbol{\phi}_2 \boldsymbol{x}_t)^2\} + 2\beta + (\boldsymbol{\phi}_2 - \boldsymbol{\mu}_\phi)^T \Sigma_\phi^{-1} (\boldsymbol{\phi}_2 - \boldsymbol{\mu}_\phi)}{\sum_{t=1}^{T} \{(1 - \gamma_t)\} + 2\alpha + 2 + P} \tag{14}$$

2.5 Differentials for w

$$\begin{aligned} \frac{\partial}{\partial w} Q(\cdot) = \frac{\partial}{\partial w} \sum_{t=1}^{T} &\left\{ \gamma_t \log \left[w \mathcal{N}(y_t \mid \boldsymbol{\phi}_1 \boldsymbol{x}_t, \sigma_1^2) \right] + (1 - \gamma_t) \log \left[(1 - w) \mathcal{N}(y_t \mid \boldsymbol{\phi}_2 \boldsymbol{x}_t, \sigma_2^2) \right] \right\} \\ &+ \frac{\partial}{\partial w} \log p(w) \end{aligned} \tag{15}$$

$$\frac{\partial}{\partial w} Q(\cdot) = \sum_{t=1}^{T} \left\{ \frac{\gamma_t}{w} - \frac{1 - \gamma_t}{1 - w} \right\} + \frac{\alpha_w - 1}{w} - \frac{\beta_w - 1}{1 - w} \tag{16}$$

2.6 Updating w

$$w = \frac{\sum_{t=1}^{T} \gamma_t + \alpha_w - 1}{T + \alpha_w + \beta_w - 2} \tag{17}$$

$$\tag{18}$$

3 Implementation of the model

```
def reset(self):
    """
        Reset priors and draw parameter estimates from prior.
```

```python
    """
    # priors
    self.alpha_w0       = self.h["alpha_w0"]
    self.beta_w0        = self.h["beta_w0"]

    # Same priors for phi1 and phi2, s2_1, s2_2, don't bother to copy vars twice
    # i.e. alpha_s2_1_0 = alpha_s2_2_0 = alpha_s20
    self.lbd_phi0       = self.h["lbd_phi0"]
    self.alpha_s20      = self.h["alpha_s20"]
    self.beta_s20       = self.h["beta_s20"]
    self.sigma_phi0     = eye(self.pdata) * self.h["lbd_phi0"]
    self.sigma_phi0_inv = eye(self.pdata) / self.h["lbd_phi0"]
    self.mu_phi0        = ones(self.pdata) * self.h["mu_phi0"]

    # Precalculations:
    self.w_gamma_ln_multiplier  = gammaln(self.alpha_w0 + self.beta_w0)
    self.w_gamma_ln_multiplier -= gammaln(self.alpha_w0)
    self.w_gamma_ln_multiplier -= gammaln(self.beta_w0)

    # initial parameter estimates drawn from prior
    self.p              = dict()
    # Weights
    self.p["w"]         = beta(self.alpha_w0, self.beta_w0)
    # Responsibilities
    self.gamma          = binomial(1, self.p["w"], self.ndata)
    # Component 1
    # inverse gamma
    self.p["sigma2_1"] = 1.0 / gamma(self.alpha_s20, 1.0 / self.beta_s20)
    self.p["phi_1"]    = mvnormal(self.mu_phi0, self.p["sigma2_1"] * self.sigma_phi0)
    # Component 2
    # inverse gamma
    self.p["sigma2_2"] = 1.0 / gamma(self.alpha_s20, 1.0 / self.beta_s20)
    self.p["phi_2"]    = mvnormal(self.mu_phi0, self.p["sigma2_2"] * self.sigma_phi0)
```

```python
def draw(self, item):
    """
        Draw a data sample from the current predictive distribution.
        Returns the y-value and z-value
    """
    mean1 = float(item.dot(self.p["phi_1"]))
    std1  = sqrt(self.p["sigma2_1"])
    mean2 = float(item.dot(self.p["phi_2"]))
    std2  = sqrt(self.p["sigma2_2"])

    if np.random.rand() < self.p["w"]:
```

```
            return normal(mean1, std1), 1
        else:
            return normal(mean2, std2), 0
```

```
def logl(self):
    """
        Calculates the full log likelihood for this model.
        Returns the logl (and the values of each term for debugging purposes)
    """

    # Our complete data posterior log likelihood seems to result in incorrect
    # values. Use incomplete data posterior log likelihood instead.
    return self.incompletelogl()

    ll          = zeros(20)
    phi_1_diff = self.p["phi_1"] - self.mu_phi0
    phi_2_diff = self.p["phi_2"] - self.mu_phi0
    phi_1_err  = phi_1_diff.T.dot(phi_1_diff)
    phi_2_err  = phi_2_diff.T.dot(phi_2_diff)
    err_1      = (self.Y - self.X.dot(self.p["phi_1"])) ** 2
    err_2      = (self.Y - self.X.dot(self.p["phi_2"])) ** 2

    gamma = self.gamma

    ### posterior factorizes p(y,z,w,phi,sigma) = p(y,z)p(w)p(phi)p(sigma)
    #                                           = p(y)p(z)p(w)p(phi)p(sigma)

    ### p(y,z)
    ll[0] =      gamma.dot(    self.p["w"]  * norm.logpdf( \
                 self.Y, self.X.dot(self.p["phi_1"]), sqrt(self.p["sigma2_1"])) )
    ll[1] = (1-gamma).dot( (1-self.p["w"]) * norm.logpdf( \
                 self.Y, self.X.dot(self.p["phi_2"]), sqrt(self.p["sigma2_2"])) )

    ### p(z) already in p(y,z)
    #ll[4] = np.sum((gamma * log(self.p["w"])) + \
    #                 ((1 - gamma) * log(1 - self.p["w"])))

    ### p(w)
    ll[5] = self.w_gamma_ln_multiplier
    ll[6] = (self.alpha_w0 - 1) * self.p["w"]
    ll[7] = (self.beta_w0  - 1) * (1 - self.p["w"])

    ### p(phi)
    # phi_1
    ll[8]  = - 0.5 * ( self.pdata * log(2 * pi * self.p["sigma2_1"]) \
```

```python
                               + log(self.lbd_phi0) )
        ll[9]  = - 0.5 * phi_1_err / (self.lbd_phi0 * self.p["sigma2_1"])
        # phi_2
        ll[10] = - 0.5 * ( self.pdata * log(2 * pi * self.p["sigma2_2"]) \
                               + log(self.lbd_phi0) )
        ll[11] = - 0.5 * phi_2_err / (self.lbd_phi0 * self.p["sigma2_2"])

        ### p(sigma2)
        # sigma2_1
        ll[12] = self.alpha_s20 * log(self.beta_s20)
        ll[13] = - gammaln(self.alpha_s20)
        ll[14] = - (self.alpha_s20 + 1.0) * log(self.p["sigma2_1"])
        ll[15] = - self.beta_s20 / self.p["sigma2_1"]
        # sigma2_2
        ll[16] = self.alpha_s20 * log(self.beta_s20)
        ll[17] = - gammaln(self.alpha_s20)
        ll[18] = - (self.alpha_s20 + 1.0) * log(self.p["sigma2_2"])
        ll[19] = - self.beta_s20 / self.p["sigma2_2"]

        return np.sum(ll), ll


    def incompletelogl(self):
        """
            Calculates the incomplete data log likelihood for this model.
            Returns the incomplete logl (and the values of each term for
            debugging purposes)
        """
        ll          = zeros(20)
        phi_1_diff = self.p["phi_1"] - self.mu_phi0
        phi_2_diff = self.p["phi_2"] - self.mu_phi0
        phi_1_err  = phi_1_diff.T.dot(phi_1_diff)
        phi_2_err  = phi_2_diff.T.dot(phi_2_diff)

        ### p(y)
        N1 = norm.pdf(self.Y, self.X.dot(self.p["phi_1"]), sqrt(self.p["sigma2_1"]))
        N2 = norm.pdf(self.Y, self.X.dot(self.p["phi_2"]), sqrt(self.p["sigma2_2"]))
        ll[0] = np.sum( np.log( self.p["w"]*N1 + (1-self.p["w"])*N2 ) )

        ### p(w)
        ll[1] = self.w_gamma_ln_multiplier
        ll[2] = (self.alpha_w0 - 1) * self.p["w"]
        ll[3] = (self.beta_w0  - 1) * (1 - self.p["w"])

        ### p(phi)
```

```python
    # phi_1
    ll[4]  = - 0.5 * ( self.pdata * log(2 * pi * self.p["sigma2_1"]) \
                       + log(self.lbd_phi0) )
    ll[5]  = - 0.5 * phi_1_err / (self.lbd_phi0 * self.p["sigma2_1"])
    # phi_2
    ll[6] = - 0.5 * ( self.pdata * log(2 * pi * self.p["sigma2_2"]) \
                      + log(self.lbd_phi0) )
    ll[7] = - 0.5 * phi_2_err / (self.lbd_phi0 * self.p["sigma2_2"])

    ### p(sigma2)
    # sigma2_1
    ll[8] = self.alpha_s20 * log(self.beta_s20)
    ll[9] = - gammaln(self.alpha_s20)
    ll[10] = - (self.alpha_s20 + 1.0) * log(self.p["sigma2_1"])
    ll[11] = - self.beta_s20 / self.p["sigma2_1"]
    # sigma2_2
    ll[12] = self.alpha_s20 * log(self.beta_s20)
    ll[13] = - gammaln(self.alpha_s20)
    ll[14] = - (self.alpha_s20 + 1.0) * log(self.p["sigma2_2"])
    ll[15] = - self.beta_s20 / self.p["sigma2_2"]

    return np.sum(ll), ll
```

```python
def EM_iter(self):
    """
        Executes a single round of EM updates for this model.

        Has checks to make sure that updates increase logl and
        that parameter values stay in sensible limits.
    """

    # ==================== E-STEP ====================

    # norm.pdf works on a vector, returning probability for each separately
    propto_gamma1 =      self.p["w"]  * norm.pdf( \
                  self.Y, self.X.dot(self.p["phi_1"]), sqrt(self.p["sigma2_1"]))
    propto_gamma2 = (1 - self.p["w"]) * norm.pdf( \
                  self.Y, self.X.dot(self.p["phi_2"]), sqrt(self.p["sigma2_2"]))

    self.gamma = propto_gamma1 / (propto_gamma1 + propto_gamma2) # responsibilities

    # ==================== M-STEP ====================

    # ========== Component weights w ==========
    num = 2*np.sum(self.gamma) + self.alpha_w0 - 1
```

```
        den = 2*self.ndata + self.alpha_w0 + self.beta_w0 - 2
        self.p["w"] = num / den

        self.assert_logl_increased("w")


        # ========== Variances sigma2 ==========
        # phi_1 and phi_2 still have the previous value, i.e. from step s, and
        # we are calculating sigma for step s+1

        # sigma2_1
        phie = np.sum((self.p["phi_1"] - self.mu_phi0) ** 2)  / self.lbd_phi0
        phiX = self.p["phi_1"].dot(self.X.T)
        target_err = (self.Y - phiX)**2
        err = self.gamma.dot(target_err)
        num = 2*self.beta_s20 + err + phie
        den = 2*self.alpha_s20 + 2.0 + np.sum(self.gamma) + self.pdata
        self.p["sigma2_1"] = num / den
        if self.p["sigma2_1"] < 0.0:
            raise ValueError("sigma2_1 < 0.0")

        # sigma2_2
        phie = np.sum((self.p["phi_2"] - self.mu_phi0) ** 2)  / self.lbd_phi0
        phiX = self.p["phi_2"].dot(self.X.T)
        target_err = (self.Y - phiX)**2
        err = (1-self.gamma).dot(target_err)
        num = 2*self.beta_s20 + err + phie
        den = 2*self.alpha_s20 + 2.0 + np.sum(1-self.gamma) + self.pdata
        self.p["sigma2_2"] = num / den
        if self.p["sigma2_2"] < 0.0:
            raise ValueError("sigma2_2 < 0.0")


        # ========== Variables phi ==========

        # phi_1
        sum_gammayx = self.gamma.T.dot( (Y * self.X.T).T )
        resp_matrix = eye(self.ndata) * self.gamma
        sum_gammaxx = self.X.T.dot(resp_matrix.dot(self.X))
        sigma_mu        = self.sigma_phi0_inv.dot(self.mu_phi0)
        sigma_phi_inv   = self.sigma_phi0_inv + sum_gammaxx
        self.p["phi_1"] = solve(sigma_phi_inv, sigma_mu + sum_gammayx)

        # phi_2
        sum_gammayx = (1-self.gamma).T.dot(  (Y * self.X.T).T  )
```

```
    resp_matrix = eye(self.ndata) * (1-self.gamma)
    sum_gammaxx = self.X.T.dot(resp_matrix.dot(self.X))
    sigma_mu        = self.sigma_phi0_inv.dot(self.mu_phi0)
    sigma_phi_inv   = self.sigma_phi0_inv + sum_gammaxx
    self.p["phi_2"] = solve(sigma_phi_inv, sigma_mu + sum_gammayx)

    self.assert_logl_increased("phi and sigma updates")
```

4 Tests with the mixture model

P	T	$\mathcal{L}_1$ train	$\mathcal{L}_{100}$ train	$\mathcal{L}_\delta$ train	$\mathcal{L}_1$ val	$\mathcal{L}_{100}$ val	$\mathcal{L}_\delta$ val
16	128	-239.79	-150.54	-89.25	-118.51	-77.28	-41.23
16	64	-47.09	-47.09	-0.00	-60.07	-60.07	0.00
16	32	-28.29	-22.46	-5.84	-48.46	-48.81	0.35
16	16	-3.55	-3.55	-0.00	-105.22	-105.22	0.00
16	8	15.53	15.53	-0.00	12.41	12.41	0.00
8	128	-116.32	-116.32	-0.00	-45.82	-45.82	0.00
8	64	-27.83	-27.83	-0.00	-17.98	-17.98	-0.00
8	32	-41.25	-26.56	-14.69	-37.56	-25.33	-12.23
8	16	-5.51	-5.01	-0.51	-6.24	-6.14	-0.10
8	8	-4.47	-0.37	-4.09	-9.77	-14.14	4.36
2	128	-132.82	-132.82	-0.00	-42.92	-42.92	-0.00
2	8	2.19	2.19	-0.00	3.76	3.76	-0.00
1	128	-117.97	-117.97	-0.00	-49.94	-49.94	-0.00
1	64	-39.05	-39.05	-0.00	-5.51	-5.51	0.00
1	32	-15.64	-15.64	-0.00	0.45	0.45	0.00
1	16	-4.65	-4.65	-0.00	3.68	3.68	-0.00
1	8	6.03	6.03	-0.00	-1.35	-1.35	-0.00

Figure 1: Comparison of likelihood using the mixture model a single initialization compared to a 100 initializations and with varying dimensionality and number of data points.

Generally more dimensions and more training data make the model harder to fit, decreasing the likelihood. This can be seen from the results and especially the dimensionality of the data has a strong effect on the decrease of the likelihood since we are trying to fit a model with a 2-dimensional basis (due to its two linear components) into a feature space with increasing dimensionality. Obviously in 2 dimensions the mixture model does not model does not yield any improvement as it is impossible to fit better than a linear model in that case and thus reduces to the same linear model with both components converging towards the same parameters.

EM is a local optimization procedure that can converge to local minima. Hence running it multiple times and choosing the model with the best training likelihood can be beneficial as it increases the probability for a better fit that is closer to the global optimum. Thus, multiple initializations tend to lead to an increase in likelihood of the training data and this can observed in many cases in the results shown in table 4 as well.

On the other hand we can also observe the effect of over-fitting due to the multiple initializations. While in several cases the test validation likelihood of a single initialization is lower than the test validation likelihood using multiple initializations as expected, there are also cases where the likelihood with multiple initializations is actually lower than using just a single initialization (e.g. in the case of $P = 8$ and $T = 8$).

5 Comparison the mixture model with the simple linear model

NOTE: These values for the Mixture Model have been calculated with the full log posterior likelihood, which might have problems in the code. Regardless of debugging it for a long time, we did not manage to locate the problem. Thus these results might not be comparable.

5.1 Data from the simple linear model

P	T	$\mathcal{L}$ LM train	$\mathcal{L}$ MM train	$\mathcal{L}$ LM validation	$\mathcal{L}$ MM validation
10	100	-92.81	-82.74	-31.51	-25.79
10	10	-10.87	-2.51	-13.81	-4.10
2	100	-73.44	-69.05	-17.29	-2.97
2	10	-2.37	4.07	0.10	7.03

Figure 2: Likelihood calculations for data drawn from the simple linear model

Already with the data from the simple linear model the mixture model seems to find a better fit.

5.2 Data from the mixture model

P	T	$\mathcal{L}$ LM train	$\mathcal{L}$ MM train	$\mathcal{L}$ LM validation	$\mathcal{L}$ MM validation
10	100	-144.07	-97.19	-62.76	-78.94
10	10	-2.37	5.15	-46.00	0.69
2	100	-63.90	-56.44	-23.18	-9.53
2	10	-4.79	2.52	-1.99	5.70

Figure 3: Likelihood calculations for data drawn from the mixture model

It seems that in with high dimensions with a lot of data points, the mixture model can overfit. With less data or less dimensions the mixture model gives better results.