14.5 条件付き混合モデル

平成 28 年 9 月 11 日

概 要

PRML の「14.5 条件付き混合モデル」についての実装と考察

目 次

1　問題設定

回帰問題, 分類問題ともに条件付き混合モデルを考える.

2　14.5.1 線形回帰モデルの混合

2.1　アルゴリズム

それぞれ重みパラメータ $\mathbf{w}_k$ で支配される K 個の線形回帰モデルを考える. このすべての構成要素は共通の雑音の分散を用いることとする. 混合分布は

$$p(t|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k N(t|\mathbf{w}_k^T \boldsymbol{\phi}, \beta^{-1}) \quad (14.34)$$

と書ける. ここで, $\boldsymbol{\theta}$ はモデル内のすべての適応パラメータの集合である.
観測値である $\{\boldsymbol{\phi}_n, t_n\}$ のデータ集合が与えられたとき, 対数尤度関数は

$$\ln p(\mathbf{t}|\boldsymbol{\theta}) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k N(t_n|\mathbf{w}_k^T \boldsymbol{\phi}_n, \beta^{-1}) \right) \quad (14.35)$$

そして, 各データ点が混合中のどの混合要素から生成されたかを表す隠れ変数 $z_{nk} \in \{0, 1\}$ を導入する. これより完全データ対数尤度関数は

$$\ln p(\mathbf{t}, Z|\boldsymbol{\theta}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \ln \{\pi_k N(t_n|\mathbf{w}_k^T \boldsymbol{\phi}_n, \beta^{-1})\} \quad (14.36)$$

これを EM アルゴリズムを用いて最大化する.
まず E ステップでは負担率 γ_{nk} を計算する.

$$\gamma_{nk} = E[z_{nk}] = p(k|\boldsymbol{\phi}_n, \boldsymbol{\theta}^{old}) = \frac{\pi_k N(t_n|\mathbf{w}_k^T \boldsymbol{\phi}_n, \beta^{-1})}{\sum_j \pi_j N(t_n|\boldsymbol{w}_j^T \boldsymbol{\phi}_n, \beta^{-1})} \quad (14.37)$$

そして, M ステップでは完全データ尤度関数の期待値を最大化する.

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) = E_Z[\ln p(\mathbf{t}, Z|\boldsymbol{\theta})] = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} \{\ln \pi_k + \ln N(t_n|\mathbf{w}_k^T \boldsymbol{\phi}_n, \beta^{-1})\}$$

これを, π_k についてラグランジュの未定乗数法を用いることで最大化すると

$$\pi = \frac{1}{N} \sum_{n=1}^{N} \gamma_{nk} \quad (14.38)$$

を得る. また, $\mathbf{w}_K$, β について最大化すると

$$\mathbf{w}_k = (\Phi^T R_k \Phi)^{-1} \Phi R_k \boldsymbol{t} \quad (14.42), \quad \beta^{-1} = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} (t_n - \mathbf{w}_k^T \boldsymbol{\phi}_n)^2 \quad (14.44)$$

となる.

線形回帰モデルの混合

1. (E ステップ) 負担率を計算する.
2. (M ステップ) 混合比, 各モデルのパラメータを計算する.
3. 1,2 を繰り返す.
4. 各モデルに混合比をかけたものを足し合わせることで, 混合モデルを作る.

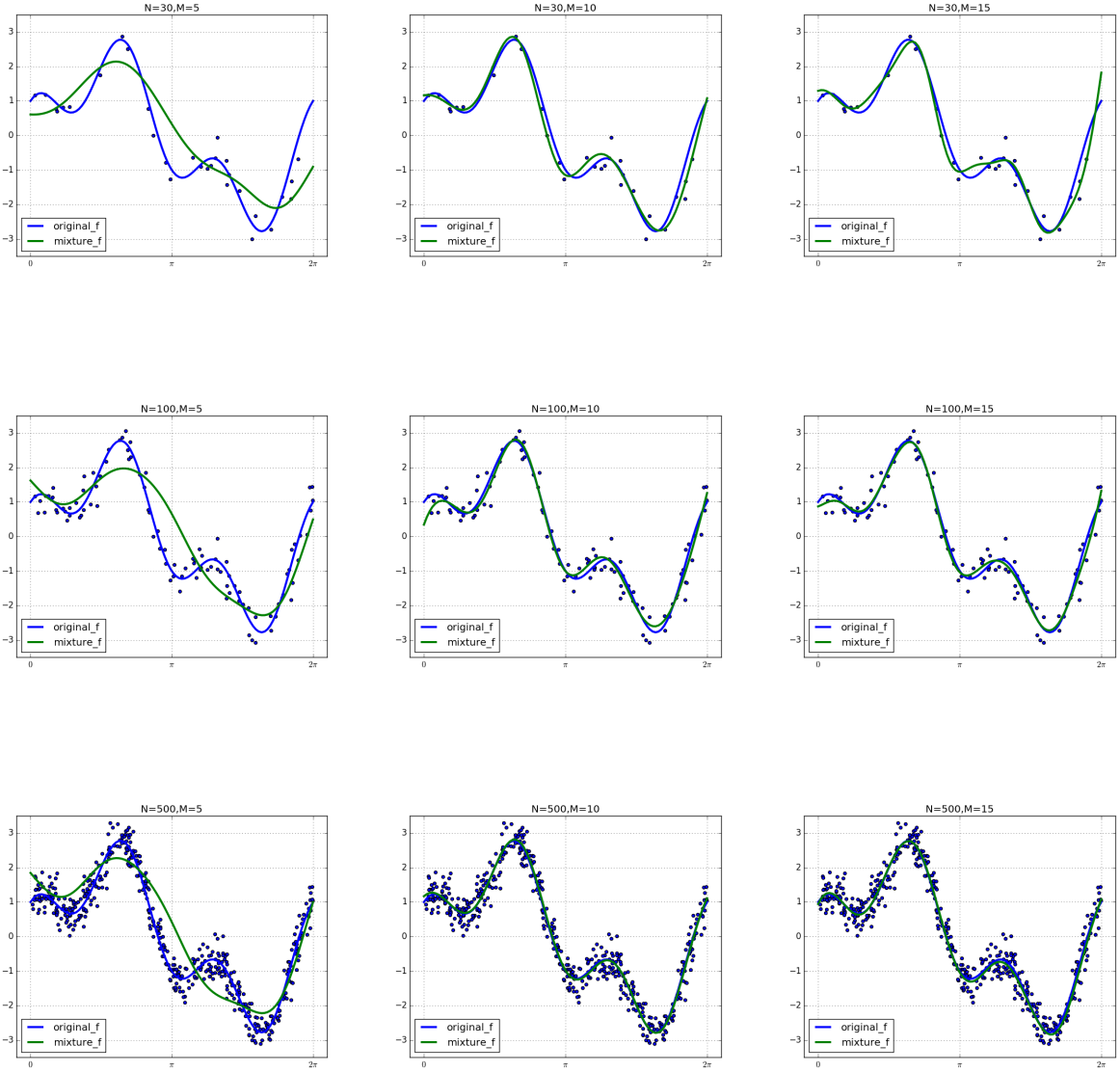
2.2 コード

混合線形回帰のコード (mixture_regression.py).

```
K=3
P=np.zeros((K,N,M))
for k in range(K):
        for n in range(N):
                for m in range(M):
                        P[k,n,m]=func(x[n],m,mu[m],s,k)
W=np.zeros((K,M))
R=np.zeros((K,N,N))
PI=np.ones(K)/K
gamma=np.zeros((N,K))
beta=1.0/(0.3)**2

diff=1
loop=0
while diff>10**-6 and loop<32:
        for n in range(N):
                tmp=np.zeros(K)
                for k in range(K):
                        tmp[k]=PI[k]*gauss(t[n],dot(W[k,:],P[k,n,:]),beta)
                for k in range(K):
                        gamma[n,k]=tmp[k]/np.sum(tmp)

        tmp_PI=PI
        for k in range(K):
                PI[k]=np.sum(gamma[:,k])/N

        for k in range(K):
                for n in range(N):
                        R[k,n,n]=gamma[n,k]

        tmp_W=W
        for k in range(K):
                tmp1=dot(dot(P[k,:,:].T,R[k,:,:]),P[k,:,:])
                tmp1=inv(tmp1)
                tmp2=dot(dot(P[k,:,:].T,R[k,:,:]),t)
                W[k,:]=dot(tmp1,tmp2)

        tmp_beta=beta
        tmp=0
        for n in range(N):
                for k in range(K):
                        tmp+=gamma[n,k]*(t[n]-dot(W[k,:],P[k,n,:]))**2
        beta=N/tmp
        loop+=1
        diff=norm(PI-tmp_PI)+norm(W-tmp_W)+abs(beta-tmp_beta)

def model_f(x):
        tmp=0
        for k in range(K):
                for m in range(M):
                        tmp+=PI[k]*W[k,m]*func(x,m,mu[m],s,k)
        return tmp
```

2.3 結果

ガウス, シグモイド, tanh 基底を用いた線形基底関数モデルの混合モデルを考えた.



誤差は

M \ N	30	100	500
5	0.82	0.89	0.88
10	0.35	0.29	0.28
15	0.35	0.28	0.27

表 1: ガウス基底

M \ N	30	100	500
5	0.87	0.79	0.78
10	0.34	0.28	0.27
15	0.34	0.28	0.27

表 2: シグモイド基底

M \ N	30	100	500
5	0.87	0.82	0.80
10	0.35	0.29	0.27
15	0.36	0.29	0.27

表 3: tanh 基底

M \ N	30	100	500
5	0.75	0.73	0.70
10	0.35	0.31	0.27
15	0.36	0.29	0.27

表 4: 混合線形回帰モデル

となった.

2.4 まとめ

線形回帰モデルの混合では, より滑らかなモデルができた.
また, 注目すべきは, 個々のモデルが弱いときに誤差減少に効果があったことである. 個々のモデルが弱いとき, モデルに依存してしまいがちなのをうまくまとめた感じがする. 加えて, 個々のモデルが強いときにも, 効果は薄いものの一定の効果があるように感じる.
簡単なモデルを数多く訓練しそれを混合させるという方法で生きてくるだろう.

3 14.5.2 ロジスティックモデルの混合

3.1 アルゴリズム

それぞれ重みパラメータ $\mathbf{w}_k$ で支配される K 個のロジスティック回帰モデルを考える. 混合分布は

$$p(t|\boldsymbol{\phi}, \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k y_k^t (1 - y_k)^{1-t} \quad (14.45)$$

と書ける. ここで, $\boldsymbol{\phi}, \boldsymbol{\theta}$ は特徴ベクトル, モデル内のすべての適応パラメータの集合である. また, $y_k = \sigma(\mathbf{w}_k^T \boldsymbol{\phi})$ である.
観測値である $\{\boldsymbol{\phi}_n, t_n\}$ のデータ集合が与えられたとき, 尤度関数は

$$p(\mathbf{t}|\boldsymbol{\theta}) = \prod_{n=1}^{N} \left(\sum_{k=1}^{K} \pi_k y_{nk}^{t_n} (1 - y_{nk})^{1-t_n} \right) \quad (14.46)$$

そして, 各データ点が混合中のどの混合要素から生成されたかを表す隠れ変数 $z_{nk} \in \{0, 1\}$ を導入する. これより完全データ尤度関数は

$$\ln p(\mathbf{t}, Z|\boldsymbol{\theta}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \left\{ \pi_k y_{nk}^{t_n} (1 - y_{nk})^{1-t_n} \right\}^{z_{nk}} \quad (14.47)$$

これを EM アルゴリズムを用いて最大化する.
まず E ステップでは負担率 γ_{nk} を計算する.

$$\gamma_{nk} = E[z_{nk}] = p(k|\boldsymbol{\phi}_n, \boldsymbol{\theta}^{old}) = \frac{\pi_k y_{nk}^{t_n} (1 - y_{nk})^{1-t_n}}{\sum_j \pi_j y_{nj}^{t_n} (1 - y_{nj})^{1-t_n}} \quad (14.48)$$

そして, M ステップでは完全データ尤度関数の期待値を最大化する.

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) = E_Z[\ln p(\mathbf{t}, Z|\boldsymbol{\theta})] = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} \{\ln \pi_k + t_n \ln y_{nk} + (1 - t_n) \ln (1 - y_{nk})\} \quad (14.49)$$

これを, π_k についてラグランジュの未定乗数法を用いることで最大化すると

$$\pi = \frac{1}{N}\sum_{n=1}^{N}\gamma_{nk} \quad (14.50)$$

を得る. また, $\mathbf{w}_K$ について最大化は閉形式で求まらず, 反復再重み付け最小二乗アルゴリズムを用いてこれを求める. ここでは,

$$\nabla_k Q = \sum_{n=1}^{N}\gamma_{nk}(t_n - y_{nk})\boldsymbol{\phi}_n \quad (14.51)$$

$$H_k = -\nabla_k\nabla_k Q = \sum_{n=1}^{N}\gamma_{nk}y_{nk}(1-y_{nk})\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T \quad (14.52)$$

を用いて,

$$\mathbf{w}_k^{(new)} = \mathbf{w}_k^{(old)} + H_k^{-1}\nabla_k Q$$

で収束まで更新する.

混合ロジスティック回帰

1. (E ステップ) 負担率を計算する.
2. (M ステップ) 混合比を計算する.
3. (M ステップ) 反復再重み付け最小二乗アルゴリズムを用いて各モデルのパラメータを計算する.
4. 1,2,3 を繰り返す.
5. 各モデルに混合比をかけたものを足し合わせることで, 混合モデルを作る.

3.2 コード

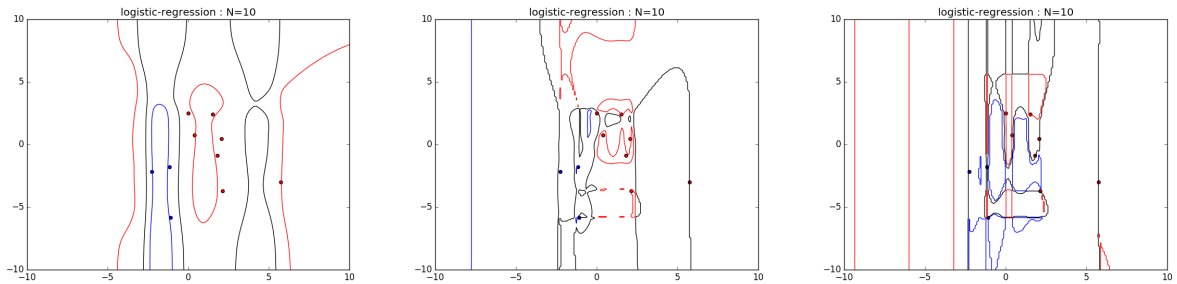
変分ロジスティック回帰のコード (variational_logistic_regression.py).

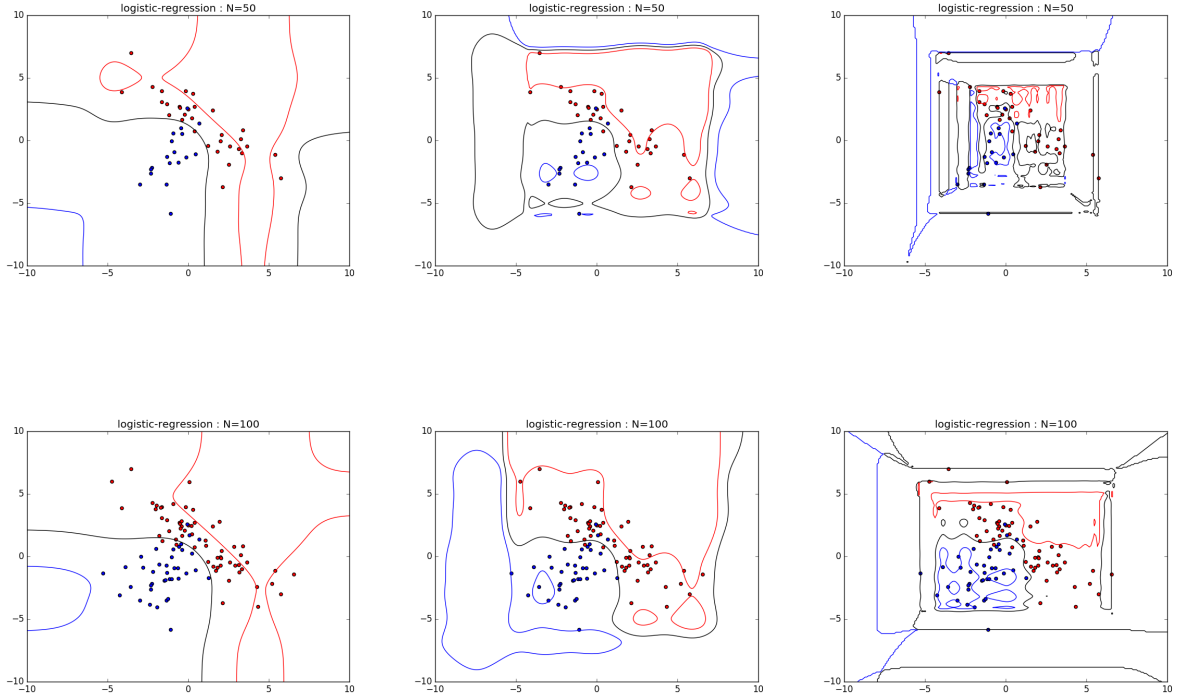
```
K=3
P=np.zeros((K,N,M2))
for k in range(K):
  for n in range(N):
    for m in range(M2):
      if m<M:
        P[k,n,m]=func(x[n,0],m,mu[m],s,k)
      else:
        P[k,n,m]=func(x[n,1],m-M+1,mu[m-M+1],s,k)
y=np.zeros((N,K))
W=np.zeros((K,M2))
PI=np.ones(K)/K
gamma=np.zeros((N,K))
diff1=1
loop1=0
while diff1>10**-6:
  for n in range(N):
    for k in range(K):
      y[n,k]=sig(dot(W[k,:],P[k,n,:]))

  for n in range(N):
    tmp=np.zeros(K)
```

```
    for k in range(K):
      tmp[k]=PI[k]*cross(y[n,k],t[n])
    for k in range(K):
      gamma[n,k]=tmp[k]/np.sum(tmp)

  tmp_PI=PI
  for k in range(K):
    PI[k]=np.sum(gamma[:,k])/N

  tmp_W=W
  H=np.zeros((K,M2,M2))
  D=np.zeros((K,M2))
  for k in range(K):
    diff2=1
    while diff2>10**-6:
      tmp2_W=W[k,:]
      for n in range(N):
        y[n,k]=sig(dot(tmp2_W,P[k,n,:]))
      for n in range(N):
        D[k,:]+=gamma[n,k]*(t[n]-y[n,k])*P[k,n,:]
      for n in range(N):
        H[k,:,:]+=gamma[n,k]*y[n,k]*(1-y[n,k])*outer(P[k,n,:],P[k,n,:])
      W[k,:]=tmp2_W+dot(inv(H[k,:,:]),D[k,:])
      diff2=norm(W[k,:]-tmp2_W)

  diff1=norm(PI-tmp_PI)+norm(W-tmp_W)

def model(z):
  tmp=np.zeros(K)
  for k in range(K):
    for m in range(M2):
      if m<M:
        tmp[k]+=W[k,m]*func(z[0],m,mu[m],s,k)
      else:
        tmp[k]+=W[k,m]*func(z[1],m-M+1,mu[m-M+1],s,k)
  ans=0
  for k in range(K):
    ans+=PI[k]*sig(tmp[k])
  return ans
```

3.3 結果

ガウス, シグモイド, tanh 基底を用いたロジスティック回帰の混合モデルを考えた. $M = 5, 10, 20$ としている.





3.4　まとめ

それぞれのモデルが簡単なとき, 混合するといい具合に作用することが分かった. ただ, それぞれのモデルが複雑になったとき回帰問題とは異なり, 過学習を見せることがわかる. 加えて言うなら, どの場合も tanh 基底を用いたモデルに劣る結果となった. この原因はガウス基底によると考える.