Bayesian Multiclass Classification - Complete Hand Calculations

1. Model Setup

Consider a multiclass classification problem with K classes and N training examples:

- Training data: $D = \{(x_n, y_n)\}_{n=1}^N$ where $x_n \in \mathbb{R}^D$ and $y_n \in \{0, 1, ..., K-1\}$
- Feature expansion: $\varphi(x_n) \in \mathbb{R}^D$ (often just x_n with intercept)
- Weight matrix: $W \in \mathbb{R}^{K\times D}$ where row k contains weights w_k for class k

2. Prior Distribution

We place independent Gaussian priors on each weight:

Prior on individual weight:

```
p(w_ij) = N(w_ij | 0, 1/α) = √(α/(2π)) exp(-α w_ij²/2)
```

Prior on weight vector w_k:

```
p(w_k) = N(w_k | 0, (1/α)I) = (α/2π)^{D/2} exp(-α ||w_k||²/2)
```

Prior on full weight matrix W:

```
p(W) = ∏_{k=1}^K ∏_{j=1}^D p(w_kj) = (α/2π)^{KD/2} exp(-α/2 ∑_{k,j} w_kj²)
```

Log prior:

```
log p(W) = KD/2 log(α/2π) - α/2 ||W||²_F
```

where $||W||^2_F = \sum_{k,j} w_{kj}^2$ is the Frobenius norm.

3. Likelihood Function

The likelihood uses a categorical distribution with softmax link:

Softmax function:

```
softmax(f)_k = exp(f_k) / ∑_j exp(f_j)
```

For linear model f_k(x_n) = w_k^T φ(x_n):

```
p(y_n = k | x_n, W) = softmax(Wϕ(x_n))_k = exp(w_k^T ϕ(x_n)) / Σ_j exp(w_j^T ϕ(x_n))
```

Likelihood for all data:

```
p(y | X, W) = Π_n p(y_n | x_n, W) = Π_n Π_k [p(y_n = k | x_n, W)]^{I(y_n = k)}
```

Log likelihood using one-hot encoding Y_nk = I(y_n = k):

```
log p(y | X, W) = Σ_n Σ_k Y_nk log p(y_n = k | x_n, W)
                = Σ_n Σ_k Y_nk [w_k^T ϕ(x_n) - log Σ_j exp(w_j^T ϕ(x_n))]
```

4. Posterior Distribution

Log posterior (up to constant):

```
log p(W | y, X) = log p(y | X, W) + log p(W) + const
                = Σ_n Σ_k Y_nk log softmax(Wϕ(x_n))_k - α/2 ||W||²_F + const
```

5. MAP Estimation

To find W_MAP, we maximize log p(W | y, X) or minimize the negative log posterior.

Gradient with respect to w_k:

```
∇_{w_k} log p(W | y, X) = Σ_n [Y_nk - p(y_n = k | x_n, W)] ϕ(x_n) - α w_k
```

Gradient in matrix form:

```
∇_W log p(W | y, X) = Φ^T (Y - P) - αW
```

where:

- Φ is N×D design matrix with rows φ(x_n)^T
- Y is N×K one-hot encoded label matrix
- P is N×K matrix with P_nk = p(y_n = k | x_n, W)

6. Laplace Approximation

The Hessian is computed as:

Block structure of Hessian: For weights w_k and w_l, the Hessian block is:

```
H_{kl} = -∂²log p(W | y, X) / ∂w_k ∂w_l^T

If k = l:
H_{kk} = Σ_n p_nk(1 - p_nk) ϕ(x_n) ϕ(x_n)^T + αI

If k ≠ l:
H_{kl} = -Σ_n p_nk p_nl ϕ(x_n) ϕ(x_n)^T
```

Laplace approximation:

```
p(W | y, X) ≈ N(W | W_MAP, Σ)
```

where Σ^{-1} is the Hessian at W_MAP.

7. Predictive Distribution

*For latent functions at x:**

```
f_k(x*) | y, X ~ N(μ_k*, σ²_k*)
```

where:

```
μ_k* = w_k^T_MAP ϕ(x*)
σ²_k* = ϕ(x*)^T Σ_k ϕ(x*)
```

and Σ_k is the k-th diagonal block of Σ.

For class probabilities (via Monte Carlo):

```
p(y* = k | y, x*) ≈ 1/S Σ_{s=1}^S softmax(W^(s) ϕ(x*))_k
```

where $W^{(s)} \sim N(W_{MAP}, \Sigma)$.

8. Example: K=3 classes, D=2 features

Weight matrix:

```
W = [[w_11, w_12],   # weights for class 0
     [w_21, w_22],   # weights for class 1
     [w_31, w_32]]   # weights for class 2
```

Flattened representation:

```
w_flat = [w_11, w_12, w_21, w_22, w_31, w_32]^T
```

For a single data point x_n = [1, x_n1]^T (with intercept):

Linear outputs:

```
f_0(x_n) = w_11 + w_12 x_n1
f_1(x_n) = w_21 + w_22 x_n1
f_2(x_n) = w_31 + w_32 x_n1
```

Softmax probabilities:

```
p_0 = exp(f_0) / (exp(f_0) + exp(f_1) + exp(f_2))
p_1 = exp(f_1) / (exp(f_0) + exp(f_1) + exp(f_2))
p_2 = exp(f_2) / (exp(f_0) + exp(f_1) + exp(f_2))
```

If y_n = 1 (one-hot: [0, 1, 0]):

```
log likelihood contribution = 0·log(p_0) + 1·log(p_1) + 0·log(p_2) = log(p_1)
```

Gradient for w_1:

```
∇_{w_1} log p(y_n | x_n, W) = (1 - p_1) x_n
```

Gradient for w_0 and w_2:

```
∇_{w_0} log p(y_n | x_n, W) = -p_0 x_n
∇_{w_2} log p(y_n | x_n, W) = -p_2 x_n
```

9. Numerical Computation Steps

1. **Initialize:** W^(0) = 0
2. **Iterate until convergence:**

- Compute P = softmax(XW^T)
- Gradient: g = X^T(Y - P) - αW
- Hessian: H (block structure as above)
- Update: W^(t+1) = W^(t) - H^{-1}g

3. **At convergence:** W_MAP = W^(final)
4. **Posterior covariance:** Σ = H^{-1}
5. **Predictions:** Monte Carlo sampling

10. Decision Theory

Expected utility for predicting class k:

```
EU(k) = Σ_j p(y* = j | y, x*) U(j, k)
```

Optimal decision:

```
k* = argmax_k EU(k)
```

For 0/1 utility (U(j,k) = I(j=k)):

```
k* = argmax_k p(y* = k | y, x*)
```

11. Complete Numerical Example

Let's work through a small example with K=3 classes, N=3 data points, and D=2 features.

Data

```
X = [[1, 0],    # x_0 (intercept=1, feature=0)
     [1, 1],    # x_1 (intercept=1, feature=1)
     [1, -1]]   # x_2 (intercept=1, feature=-1)

y = [0, 1, 2]   # Class labels

Y = [[1, 0, 0], # One-hot encoding
     [0, 1, 0],
     [0, 0, 1]]
```

Initial weights (W_0 = 0)

```
W = [[0, 0],    # w_0 for class 0
     [0, 0],    # w_1 for class 1
     [0, 0]]    # w_2 for class 2
```

Step 1: Compute linear outputs f = XW^T

```
f = [[0, 0, 0],   # f(x_0) for all classes
     [0, 0, 0],   # f(x_1) for all classes
     [0, 0, 0]]   # f(x_2) for all classes
```

Step 2: Apply softmax

For each row, softmax gives:

```
p_00 = exp(0)/(exp(0)+exp(0)+exp(0)) = 1/3
p_01 = exp(0)/(exp(0)+exp(0)+exp(0)) = 1/3
p_02 = exp(0)/(exp(0)+exp(0)+exp(0)) = 1/3

P = [[1/3, 1/3, 1/3],
     [1/3, 1/3, 1/3],
     [1/3, 1/3, 1/3]]
```

Step 3: Compute log likelihood

```
log L = Σ_n Σ_k Y_nk log P_nk
      = 1×log(1/3) + 0×log(1/3) + 0×log(1/3) +   # n=0
        0×log(1/3) + 1×log(1/3) + 0×log(1/3) +   # n=1
        0×log(1/3) + 0×log(1/3) + 1×log(1/3)     # n=2
      = 3 × log(1/3) = -3.296
```

Step 4: Compute gradient

```
∇_W log p(W|y,X) = X^T(Y - P) - αW

Y - P = [[2/3, -1/3, -1/3],
         [-1/3, 2/3, -1/3],
         [-1/3, -1/3, 2/3]]

X^T(Y - P) = [[1, 1, 1]   [[2/3, -1/3, -1/3],
              [0, 1, -1]] × [-1/3, 2/3, -1/3],
                           [-1/3, -1/3, 2/3]]

           = [[0, 0, 0],
              [0, 1, -1]]

With α = 1:
∇_W = [[0, 0],
       [0, 1],
       [0, -1]]
```

Step 5: Update weights (gradient ascent)

```
W_1 = W_0 + η × ∇_W   (with η = 0.1)

W_1 = [[0, 0],
       [0, 0.1],
       [0, -0.1]]
```

Step 6: Compute new predictions

```
f = XW_1^T = [[0, 0, 0],
              [0, 0.1, -0.1],
              [0, -0.1, 0.1]]

For x_1 = [1, 1]:
p_10 = exp(0)/(exp(0)+exp(0.1)+exp(-0.1)) = 0.315
p_11 = exp(0.1)/(exp(0)+exp(0.1)+exp(-0.1)) = 0.347
p_12 = exp(-0.1)/(exp(0)+exp(0.1)+exp(-0.1)) = 0.338
```

After Convergence (hypothetical)

Suppose after optimization we get:

```
W_MAP = [[0.5, -0.3],
         [0.2, 0.8],
         [-0.1, -0.5]]
```

Predictions at x* = [1, 0.5]

```
f*(x*) = x* × W_MAP^T = [1, 0.5] × [[0.5, 0.2, -0.1],
                                    [-0.3, 0.8, -0.5]]
                      = [0.35, 0.6, -0.35]

Softmax:
p_0 = exp(0.35)/(exp(0.35)+exp(0.6)+exp(-0.35)) = 0.331
p_1 = exp(0.6)/(exp(0.35)+exp(0.6)+exp(-0.35)) = 0.427
p_2 = exp(-0.35)/(exp(0.35)+exp(0.6)+exp(-0.35)) = 0.242

Decision: argmax_k p_k = 1
```

Laplace Approximation Example

For the Hessian at W_MAP, each block H_kl has form:

```
H_kk = Σ_n p_nk(1-p_nk)x_n x_n^T + αI

Example for k=0:
If p_00 = 0.7, p_10 = 0.2, p_20 = 0.1:

H_00 = 0.7×0.3×[[1],[0]][[1,0]] + 0.2×0.8×[[1],[1]][[1,1]] +
       0.1×0.9×[[1],[-1]][[1,-1]] + 1×I

     = [[0.21+0.16+0.09+1, 0+0.16-0.09],
        [0+0.16-0.09, 0+0.16+0.09+1]]

     = [[1.46, 0.07],
        [0.07, 1.25]]
```

Monte Carlo Prediction

Draw samples W^(s) ~ N(W_MAP, Σ), then:

```
p(y*=k|x*,y) ≈ 1/S ∑_s softmax(W^(s)x*)_k

Example with S=3 samples:
W^(1): p^(1) = [0.32, 0.44, 0.24]
W^(2): p^(2) = [0.35, 0.41, 0.24]
W^(3): p^(3) = [0.33, 0.43, 0.24]

Average: p = [0.333, 0.427, 0.240]
```

This completes the mathematical derivation with concrete numerical examples for Bayesian multiclass classification.