CLL Crash Course

Leif Johnson

August 12, 2022

1 Notation

- window – a group of pixels in the image. A window is usually denoted with the letter A.
- $\mathcal{A}$ – A collection of windows.
- C – set of colors in the image. $n_{color} = n_c =\mid C \mid$. In practice $C = \{1, \ldots, n_c\}$.
- C^A – set of possible arrangements of colors in window A. What this set is depends on the size and shape of A, and the number of colors in C. We frequently enumerate the elements of C^A as $c_1, \ldots, c_{|C|^{|A|}}$ or $y_1, \ldots, y_{|C|^{|A|}}$. c_1 is taken to be the "base case".
- θ – the parameter for the model.
- $t(x)$ – the canonical statistic for the model, $t_{c_1}(x), \ldots, t_{c_{|C|}}(x)$ are the number of pixels that match the corresponding colors. $t_*(x)$ is the number of pixel neighbor pairs that have the same color.

2 CLL Basics

Composite Likelihood (CL) is a method of approximating the Likelihood, used when maximizing the Likelihood would be unfeasible using normal methods. Instead we calculate the Maximum Composite Likelhood Estimator (MCLE), which approaches the MLE in the limit. We do all of the maximizing on the log scale.

In the potts setting, the image is first divided up into a collection of windows, $\mathcal{A}$. The Likelihood is approximated by multiplying the conditional PMFs for each window together. This is the Composite Likelihood. For each

$A \in \mathcal{A}$, C^A is the set of possible colors to fill window A. Then for $A \in \mathcal{A}$ the conditional PMF for A is

$$\begin{aligned} f_A(x_A \mid Rest) &= \frac{f(x_A \cup Rest)}{\sum_{y \in C^A} f(y \cup Rest)} \\ &= \frac{e^{\langle t(x_A \cup x_{L \setminus A}), \theta \rangle}}{\sum_{y \in C^A} e^{\langle t(y \cup x_{L \setminus A}), \theta \rangle}}. \end{aligned}$$

We calculate the MCLE by maximizing the Composite Log Likelihood (CLL), given by

$$\log_{\mathcal{A}}(\theta) = \sum_{A \in \mathcal{A}} \log(f_A(x_A \mid Rest)) \tag{1}$$

3 Identifiability

It should be noted that the model is not identifiable, as we have the constraint

$$\sum_{c \in C} t_c(x) = n_{row} * n_{col}$$

We solve this issue by assuming the parameter for the first color to be zero, and dropping the appropriate statistic whenever we are doing calculations.

4 Realizing a model

```
> library(potts)
> nrow <- 32
> ncol <- 32
> ncolor <- 4
> theta.true <- rep(0, ncolor+1)
> theta.true[ncolor+1] <- log(1 + sqrt(ncolor))
> x <- matrix(sample(ncolor, nrow*ncol, replace=TRUE), nrow=nrow,
+             ncol=ncol)
> out <- potts(packPotts(x, ncolor), theta.true, nbatch=1000, blen=1)
> x <- unpackPotts(out$final)
```

The realized image for this run is given in figure 1.

If we are going to estimate some MCLE's, we need to calculate the canonical statistics for all of the windows. The functions *composite.ll* and *gr.composite.ll* calculate (1) and the gradient of (1) respectively. They each take three arguments

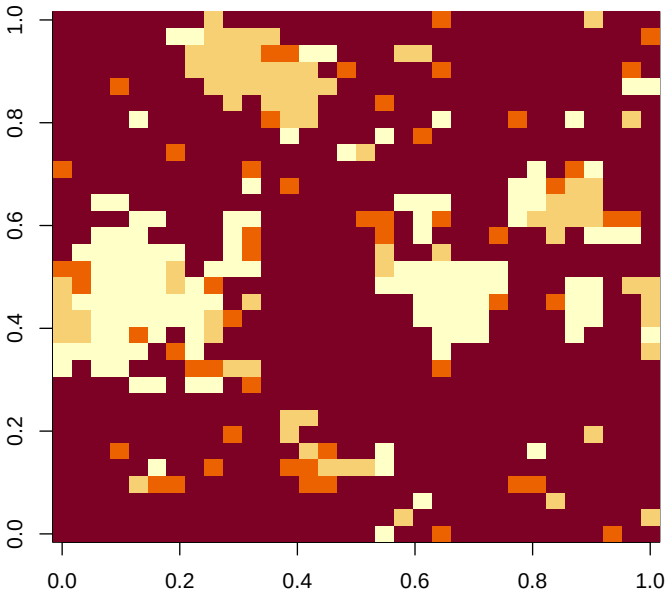

Figure 1: Realized Potts Image

- *theta* – the value of θ to evaluate.
- *t_stat* – the value of the canonical statistic for the entire image
- *t_cache* – three dimensional array. Index 1 goes across elements of $\mathcal{A}$. Index 2 goes across the elements of C^A. Hence *t_cache[i,j,]* contains $t(y_j \cup Rest)$ for the i^{th} element of $\mathcal{A}$.

So before we can calculate the CLL, we need to calculate canonical staistics.

5 Calculating Canonical Statistics

To calculate (1) we need to calculate the value of $t(y \cup Rest)$ for all $y \in C^A$ for all $A \in \mathcal{A}$. Before we calculate any, we will load the library and

generate an image to work on.

Once we have the image, we can calculate $t(x)$ using the function, *calc_ t*.

```
> t_stat <- calc_t(x, ncolor)
> t_stat

   1    2    3    4    *
 127   77   54  766 1549
```

Next we need to calculate *t_ cache*, however, this calculation depends on what collection of windows we are going to use. For this Vignette, we are going to use two collections

1. The collection of all singletons. This corresponds to Besag's Pseudo-likelihood, so we will actually be calculating the MPLE.
2. The collection consisting of non-overlapping windows of horizontal pixel pairs.

The next code chunk shows how to calculate the *t_ cache* for these collections, and to see the value of $t(y \cup Rest)$ across the first window.

```
> t_cache_mple <- generate_t_cache(x, ncolor, t_stat, nrow*ncol, 1,
+                                  singleton)
> t_cache_mple[[1]]

     [,1] [,2] [,3] [,4]
[1,]   77   54  765 1545
[2,]   78   54  765 1545
[3,]   77   55  765 1545
[4,]   77   54  766 1549

> t_cache_two <- generate_t_cache(x, ncolor, t_stat, nrow*ncol/2, 2,
+                                 twopixel.nonoverlap)
> t_cache_two[[1]]

      [,1] [,2] [,3] [,4]
 [1,]   77   54  764 1544
 [2,]   78   54  764 1544
 [3,]   77   55  764 1543
 [4,]   77   54  765 1545
 [5,]   78   54  764 1543
```

```
 [6,]   79   54  764 1545
 [7,]   78   55  764 1543
 [8,]   78   54  765 1545
 [9,]   77   55  764 1543
[10,]   78   55  764 1544
[11,]   77   56  764 1544
[12,]   77   55  765 1545
[13,]   77   54  765 1546
[14,]   78   54  765 1547
[15,]   77   55  765 1546
[16,]   77   54  766 1549
```

6 Calculating the CLL

Once we have calculated the canonical statistics, we can use them to evaluate and/or optimize the CLL. We can first compute the value of the CLL at the known parameter value.

```
> composite.ll(theta.true[-1], t_stat, t_cache_mple)

[1] -542.7564

> gr.composite.ll(theta.true[-1], t_stat, t_cache_mple)

[1]  -4.657440   3.686012  -3.830060 -52.982143

> composite.ll(theta.true[-1], t_stat, t_cache_two)

[1] -545.6967

> gr.composite.ll(theta.true[-1], t_stat, t_cache_two)

[1]  -4.356461   3.099829  -4.988654 -43.640630
```

Or we could optimize it using the usual techniques.

```
> theta.initial <- 1:ncolor
> optim.mple <- optim(theta.initial, composite.ll, gr=gr.composite.ll,
+                     t_stat, t_cache_mple, method="BFGS",
+                     control=list(fnscale=-1))
> optim.mple$par
```

```
[1] -0.1995467 -0.1026661  0.1100409  0.9643537

> optim.two <- optim(theta.initial, composite.ll, gr=gr.composite.ll,
+                    t_stat, t_cache_two, method="BFGS",
+                    control=list(fnscale=-1))
> optim.two$par

[1] -0.17350470 -0.10088773  0.04708712  1.00006222

> theta.true

[1] 0.000000 0.000000 0.000000 0.000000 1.098612
```

If your system has the *multicore* installed, you can take advantage by passing mclapply in as the fapply argument to the CLL family of functions.

```
library(multicore)
t_stat <- calc_t(x, ncolor)
t_cache_mple <- generate_t_cache(x, ncolor, t_stat, nrow*ncol, 1,
                                 singleton, mclapply)
theta.initial <- 1:ncolor
optim.mple <- optim(theta.initial, composite.ll, gr=gr.composite.ll,
                    t_stat, t_cache_mple, mclapply, method="BFGS",
                    control=list(fnscale=-1))
optim.mple$par
```