

# CLL Crash Course

Leif Johnson

March 15, 2017

## 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.  $n_{color} = n_c = |C|$ . In practice  $C = \{1, \dots, n_c\}$ .
- $C^A$  – 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  $c_1, \dots, c_{|C|^{|A|}}$  or  $y_1, \dots, y_{|C|^{|A|}}$ .  $c_1$  is taken to be the “base case”.
- $\theta$  – the parameter for the model.
- $t(x)$  – the canonical statistic for the model,  $t_{c_1}(x), \dots, t_{c_{|C|}}(x)$  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 Likelihood 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

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

### 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

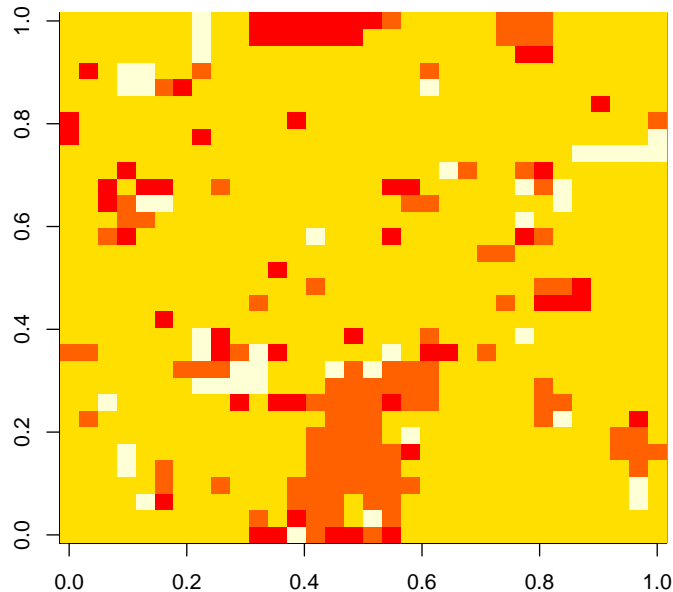


Figure 1: Realized Potts Image

- *theta* – the value of  $\theta$  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 statistics.

## 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      *
58  102  819  45 1615
```

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,]  102  818  45 1611
[2,]  103  818  45 1611
[3,]  102  819  45 1615
[4,]  102  818  46 1611
```

```
> 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,]  102  817  45 1609
[2,]  103  817  45 1608
[3,]  102  818  45 1611
[4,]  102  817  46 1608
[5,]  103  817  45 1608
```

```

[6,] 104 817 45 1609
[7,] 103 818 45 1611
[8,] 103 817 46 1608
[9,] 102 818 45 1611
[10,] 103 818 45 1611
[11,] 102 819 45 1615
[12,] 102 818 46 1611
[13,] 102 817 46 1608
[14,] 103 817 46 1608
[15,] 102 818 46 1611
[16,] 102 817 47 1609

```

## 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] -459.076

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

[1] 5.208333 -2.958333 -2.596429 -8.042857

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

[1] -473.1667

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

[1] 7.532177 -2.779513 -4.373245 -10.354890

```

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] 0.08757543 0.01221397 -0.07386329 1.08207695

> 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] 0.12089672 0.05372912 -0.08853104 1.07009843

> 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

```