Stat 8054 Lecture Notes: Isotonic Regression

Charles J. Geyer

January 06, 2022

Contents

1 License

2 R

- The version of R used to make this document is 4.1.2.
- The version of the `Iso` package used to make this document is 0.0.18.1.

- The version of the `rmarkdown` package used to make this document is 2.11.

3 Lagrange Multipliers

3.1 A Theorem

The following theorem originally comes from Shapiro (1979).

Theorem 3.1. *Consider the following constrained optimization problem*

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } g_i(x) = 0, \qquad i \in E, \\ &\qquad\qquad g_i(x) \le 0, \qquad i \in I, \end{aligned} \tag{1}$$

where E and I are disjoint finite sets, and its Lagrangian function

$$L(x, \lambda) = f(x) + \sum_{i \in E \cup I} \lambda_i g_i(x) \tag{2}$$

If there exist x^* and λ such that

a. *x^* minimizes $L(\,\cdot\,, \lambda)$,*
b. *$g_i(x^*) = 0$, $i \in E$ and $g_i(x^*) \le 0$, $i \in I$,*
c. *$\lambda_i \ge 0$, $i \in I$, and*
d. *$\lambda_i g_i(x^*) = 0$, $i \in I$.*

Then x^* solves the constrained problem (1).

The conditions in the theorem are called

a. Lagrangian minimization,
b. primal feasibility,
c. dual feasibility, and
d. complementary slackness.

Say x is *feasible* if the constraints in (1) are satisfied. We say λ is *feasible* if the constraints in condition (c) hold. That is, primal feasibility is feasibility of x, and dual feasibility is feasibility of λ.

The components of λ are called Lagrange multipliers. From the eponym, this theorem is old. In the case where there are only equality constraints it was invented by Lagrange. Having inequality constraints is a twentieth century innovation.

Proof. By (a)

$$L(x, \lambda) \ge L(x^*, \lambda), \tag{3}$$

for all x, which is equivalent to

$$f(x) \ge f(x^*) + \sum_{i \in E \cup I} \lambda_i g_i(x^*) - \sum_{i \in E \cup I} \lambda_i g_i(x) \tag{4}$$

So for feasible x

$$f(x) \ge f(x^*) - \sum_{i \in E \cup I} \lambda_i g_i(x) \ge f(x^*) \tag{5}$$

The first equality is conditions (b) and (d), and the second inequality is (c) and feasibility of x. □

3.2 Kuhn-Tucker Conditions

When condition (a) is replaced by derivative equal to zero, that is

$$\nabla f(x^*) + \sum_{i \in E \cup I} \lambda_i \nabla g_i(x^*) = 0, \tag{6}$$

then we no longer have a theorem because we know that derivative equal to zero is neither necessary nor sufficient for even a local minimum, much less a global one. Nevertheless, it is the conditions in this form that have an eponym. They are called the Kuhn-Tucker conditions or the Karush-Kuhn-Tucker conditions.

For convex problems the Kuhn-Tucker conditions often guarantee a global minimum. See the section about convex programming below.

4 Isotonic Regression

Isotonic regression is the competitor of simple linear regression (univariate response, univariate predictor) when it is assumed that the regression function is monotone rather than linear (of course linear is also monotone, but not vice versa; we are making a more general, weaker assumption).

4.1 Homoscedastic Normal Errors

We start with assuming homoscedastic normal errors (as in the usual theory of linear models). It will turn out that exactly the same algorithm that solves this problem also does logistic regression or Poisson regression if link functions are chosen to make the problem exponential family (logit link for binomial response, log link for Poisson response).

As usual, we assume the components Y_i of the response vector are random and the components x_i of the predictor vector are fixed (if they are random, then we condition on their observed values).

Also, until further notice we assume the distribution of Y_i is $\text{Normal}(\mu_i, \sigma^2)$, which is the usual linear models assumption. Now the monotonicity constraints are

$$x_i \le x_j \quad \text{implies} \quad \mu_i \le \mu_j$$

The unknown parameters are the vector μ having components μ_i and the scalar σ^2.

What are the maximum likelihood estimates?

4.2 Rewriting to Deal with Duplicate Predictor Values

We see that $x_i = x_j$ implies $\mu_i = \mu_j$. So rewrite the problem so we only have one mean parameter for each unique predictor value.

Let $z_1, \ldots, z_k$ be the unique predictor values *in sorted order.*

Define

$$w_j = \sum_{\substack{i=1 \\ x_i = z_j}}^{n} 1$$

$$V_j = \frac{1}{w_j} \sum_{\substack{i=1 \\ x_i = z_j}}^{n} Y_i$$

Then

$$V_j \sim \text{Normal}(\nu_j, \sigma^2 / w_j)$$

where

$$\nu_1 \le \nu_2 \le \cdots \le \nu_k$$

and where

$$\nu_j = \mu_i \quad \text{whenever} \quad z_j = x_i$$

4.3 Minus Log Likelihood

$$f(\mu, \sigma^2) = n \log(\sigma) + \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \mu_i)^2$$

4.4 Lagrangian

$$L(\mu, \sigma^2, \lambda) = n \log(\sigma) + \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \mu_i)^2 + \sum_{j=1}^{k-1} \lambda_j (\nu_j - \nu_{j+1})$$

4.5 Kuhn-Tucker Conditions

We find, as usual in least squares problems, that the equations for the means don't involve the variance, so our estimates for the means don't involve the variance.

In particular,

$$\begin{aligned} \frac{\partial L(\mu, \sigma^2, \lambda)}{\partial \nu_m} &= \lambda_m - \lambda_{m-1} - \frac{1}{\sigma^2} \sum_{i=1}^{n} (y_i - \mu_i) \frac{\partial \mu_i}{\partial \nu_m} \\ &= \lambda_m - \lambda_{m-1} - \frac{w_m (v_m - \nu_m)}{\sigma^2} \end{aligned} \tag{7}$$

Note that $\partial \mu_i / \partial \nu_m$ is equal to one if $x_i = z_m$ and zero otherwise, so the terms in the sum in the first line are nonzero only if $x_i = z_m$.

As it stands this formula is only valid for $1 < m < k$. In order to make it valid for all m, we define $\lambda_0 = \lambda_k = 0$.

Setting (7) equal to zero gives us the first Kuhn-Tucker condition. So set these equal to zero, and multiply through by σ^2 giving

a. $-w_m(v_m - \nu_m) + \kappa_m - \kappa_{m-1} = 0$, $m = 1, \ldots, k$

where we have introduced $\kappa_m = \sigma^2 \lambda_m$.

Since κ_m is zero, positive, or negative precisely when λ_m is, the rest of the Kuhn-Tucker conditions (for the mean parameters) are

b. $\nu_j \le \nu_{j+1}$, $j = 1, \ldots, k-1$.

c. $\kappa_j \ge 0$, $j = 1, \ldots, k-1$.

d. $\kappa_j(\nu_j - \nu_{j+1}) = 0$, $j = 1, \ldots, k-1$.

And, we see that, as usual, the equations for maximum likelihood estimation of the mean parameters do not involve σ^2 (after some rewriting).

4.6 Blocks

We first consider applying only conditions (a) (derivative of Lagrangian equal to zero) and (b) (complementary slackness).

At any vector ν, we divide it up into blocks of equal consecutive components. Such blocks have length one if some ν_j is not equal to the nu on either side.

So consider such a block. Suppose

$$\nu_{j^*-1} \neq \nu_{j^*} = \cdots = \nu_{j^{**}-1} \neq \nu_{j^{**}}$$

where to make this work for the edge cases, we define $\nu_0 = -\infty$ and $\nu_{k+1} = +\infty$.

Complementary slackness implies $\kappa_{j^*-1} = \kappa_{j^{**}-1} = 0$. Hence, now using condition (a),

$$\begin{aligned} 0 &= \sum_{j=j^*}^{j^{**}-1} \left[- w_j (v_j - \nu_j) + \kappa_j - \kappa_{j-1} \right] \\ &= - \sum_{j=j^*}^{j^{**}-1} w_j (v_j - \nu) \end{aligned}$$

where $\nu = \nu_{j^*} = \cdots = \nu_{j^{**}-1}$.

Solving we get

$$\nu_{j^*} = \cdots = \nu_{j^{**}-1} = \frac{\sum_{j=j^*}^{j^{**}-1} w_j v_j}{\sum_{j=j^*}^{j^{**}-1} w_j}$$

In summary, applying conditions (a) and (d) only (so far), the mean values in a block of equal means is the weighted average of the v_j values, which is the unweighted average of the y_i values for the block.

4.7 The Pool Adjacent Violators Algorithm (PAVA)

PAVA does the following

1. [Initialize] Set ν and κ to any values satisfying conditions (a), (c), and (d).
2. [Terminate] If condition (b) is satisfied, stop. [*Kuhn-Tucker conditions are satisfied.*]
3. [PAV] Choose any j such that $\nu_j > \nu_{j+1}$. And "pool" the blocks containing j and $j + 1$, that is, make them one block (and the nu values for this pooled block will again be the weighted average of vee values for this pooled block). [*This step maintains conditions (a), (c), and (d).*]
4. Go to step 2.

The initialization step is easy. One way to do it is to set $\nu = v$ and $\kappa = 0$.

4.8 Non-Determinism

PAVA is a non-deterministic algorithm. In step 3, if there is more than one pair of adjacent violators, then any one of them can be chosen to be pooled in that step. The choice can be made in any way: leftmost, rightmost, random, whatever.

4.9 Example

To keep things simple we will assume the predictor values are the indices of the response vector (so there are no repeated predictor values and they are in sorted order)

```r
pava <- function(y) {
    blocks <- as.list(y)
    repeat {
        block.means <- sapply(blocks, mean)
        block.diffs <- diff(block.means)
        if (all(block.diffs >= 0)) return(unlist(blocks))
        j <- which(block.diffs < 0)
        if (length(j) > 1) {
            # non-determinism !!!
            # never call R function sample with length of first arg equal to one
            # its behavior is one of the bad parts of R
            j <- sample(j, 1)
        }
        violators <- blocks[c(j, j + 1)]
        pool.length <- length(unlist(violators))
        pool.mean <- mean(unlist(violators))
        i <- seq_along(blocks)
        blocks <- c(blocks[i < j], list(rep(pool.mean, pool.length)),
            blocks[i > j + 1])
    }
}
```

So let's try it out.

```
y <- rnorm(20, mean = 1:20, sd = 3)
plot(y)
points(1:20, pava(y), pch = 19)

points(1:20, Iso::pava(y, long.out = TRUE)$y, pch = 19, col = "red")
```

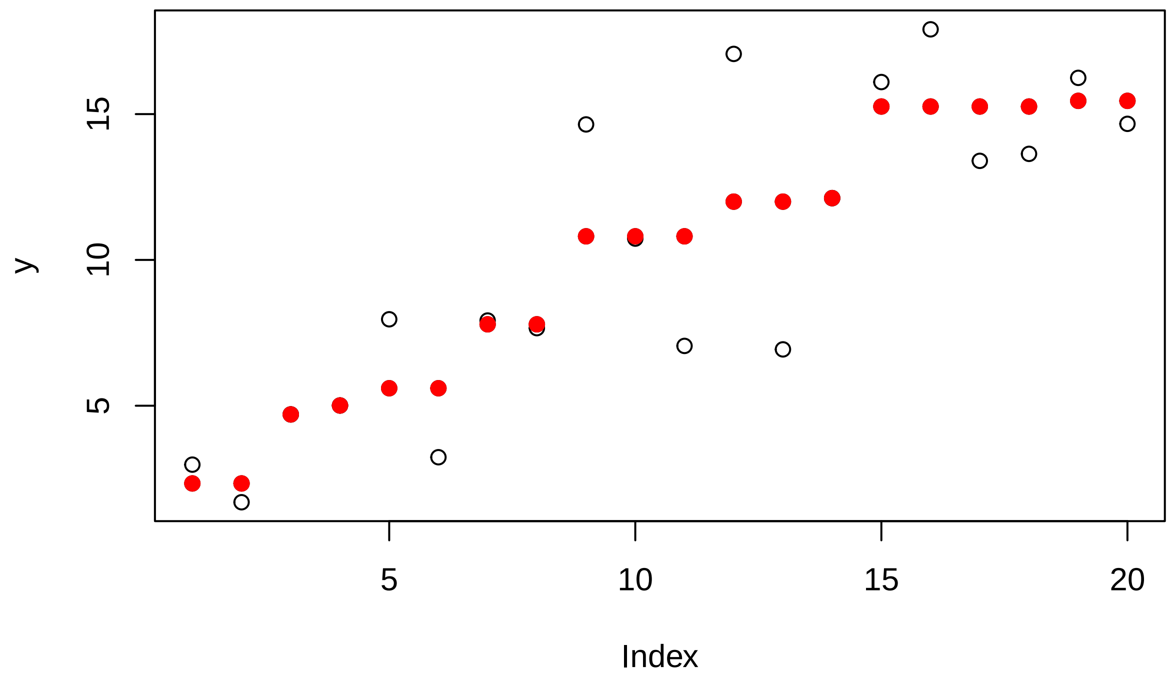

Figure 1: Data (hollow dots) and Isotonic Regression (solid dots)

As a check, our implementation of PAVA above prints black dots and the implementation of PAVA in CRAN package `Iso` prints red dots. They agree (the red dots are right on top of the black dots).

4.10 Comment on Our Implementation

When first written, our implementation had two serious bugs that took a lot of time to find.

- We forgot (or didn't clearly think through what was needed) the `list` in the last statement of the repeat loop. Omitting `list` causes all the blocks to always be length 1 and to always have `length(y)` blocks. Not how we intended it to work.
- We forgot about one of the worst "bad parts" of R: the crazy behavior of R function `sample` when its first argument has length one, even though our 3701 lecture notes and the R help page for this function warn about this "undesirable behavior" (to quote the help page). Omitting the `if` governing the invocation of `sample` was a bad bug.

4.11 Generalities about Proving Programs Correct

The best book ever written about computer programming is Dijkstra (1976).

It was not the first work about proving programs correct and certainly not the last. But it is by far the most elegant and has all the basic principles except for parallel processing.

Parallel processing is much harder; see Feijen and van Gasteren (1999) or Misra (2001) for that.

There are two fundamental ideas of proving programs correct:

- loop invariants, and
- proofs of termination.

Here we have a very complicated loop (steps 2, 3, 4 in the algorithm description, the `repeat` construct in the R implementation). We do not know in advance how many times it will execute. We do not know what it will do in each iteration (that has deliberately been made random so we cannot know).

For loops of that sort, the most important reasoning tool is the loop invariant. This is a *logical statement* (something that is math and either true or false) that is asserted to be and can be proved to be true every time through the loop at the point where the decision is made whether or not to leave the loop.

Here the loop invariant is the assertion that Kuhn-Tucker conditions (a), (c), and (d) (Lagrangian derivative zero, dual feasibility, complementary slackness) hold every time the program arrives at step 2 of the algorithm description, arriving there either from step 1 or step 4.

Figuring out a valid loop invariant may be easy or very hard, but always necessary. It is the only good proof method for programs having complicated loops.

Proving termination (that program does not do an infinite loop) requires that we can define some nonnegative integer quantity that decreases in each iteration of the loop. That quantity gives an upper bound on the number of iterations left to do.

Here the termination bound is the number of blocks. This decreases by one each time through the loop. If the loop does not terminate before there is only one block, then it must terminate when there is only one block, because then the estimated regression function is constant (all the constraints hold with equality) so condition (b) (primal feasibility) is satisfied.

The final bit of proof methodology is to put together the loop invariant and the termination condition. When the loop terminates, the termination condition is true and the loop invariant is true. Together

$$\text{loop invariant AND termination condition}$$

should imply whatever we are trying to prove the program does.

Here we claim that PAVA established the Kuhn-Tucker conditions for this problem. The loop invariant is conditions (a), (c), and (d). The termination condition is condition (b). So together that is all the conditions.

4.12 Proof that PAVA Works

Our proof has already been sketched in the preceding section. We only need to fill in the details. There are two remaining conditions. We must check

- the initialization step establishes the loop invariant, and
- the PAV step maintains the loop invariant (if the loop invariant holds at the beginning of the step, then it holds at the end; in our R implementation, if it holds at the top of the `repeat` construct, then it also holds at the bottom).

The other parts of the proof are obvious and were argued in the preceding section (number of blocks is a bound on the number of iterations, and loop invariant and termination condition implies what was to be proved).

4.12.1 Initialization

It is claimed in the algorithm description section that the initialization $\nu = v$ and $\kappa = 0$ satisfies conditions (a), (c), and (d) of the Kuhn-Tucker conditions.

That it satisfies (c) and (d) is obvious from all the kappas being zero. And all the kappas being zero and $v_j = \nu_j$ for all j makes condition (a) also hold.

4.12.2 Loop Invariant

It remains to be shown that the computations in each loop iteration (pooling of adjacent violators) maintains the loop invariant. The discussion in the section about blocks above shows that making the estimates for the means in a block the average for the response values in the block satisfies Kuhn-Tucker conditions (a) and (d) (in fact, this is both necessary and sufficient for that). Thus Kuhn-Tucker conditions (a) and (d) will clearly hold at the bottom of each iteration. The blocks that are unchanged in the iteration continue to have estimates that are means for the block. The new block created by pooling also has this property.

So it only remains to be shown that the dual feasibility part of the loop invariant is maintained. And to do that we have to learn more about Lagrange multipliers.

4.12.2.1 Blocks Revisited

So we return to the notation of the section about blocks above. Now we find out what the Lagrange multipliers of the block are. As in that section about blocks above, we are only assuming Kuhn-Tucker conditions (a) and (d) (Lagrangian derivative zero and complementary slackness hold). We are not at this point assuming dual feasibility.

Now instead of summing over the whole block, we only sum over part. As in that section, we have by complementary slackness $\kappa_{j^*-1} = \kappa_{j^{**}-1} = 0$. Now for $j^* \le r < j^{**}$

$$\begin{aligned} 0 &= \sum_{j=j^*}^{r} \left[- w_j(v_j - \nu_j) + \kappa_j - \kappa_{j-1} \right] \\ &= - \left[\sum_{j=j^*}^{r} w_j(v_j - \nu_j) \right] + \kappa_r \end{aligned}$$

so

$$\kappa_r = \sum_{j=j^*}^{r} w_j(v_j - \nu_j) \qquad (*)$$

Thus the Lagrange multipliers are the running sums of residuals for the block. They are what we have to check are nonnegative in order to prove dual feasibility.

4.12.2.2 Characterization of Solution

As an aside, we note that this gives us a complete characterization of solutions. Primal feasibility we know how to check (the means form a monotone sequence). Now we just learned how to check dual feasibility (the running sums $(*)$ over each block are nonnegative).

Let's check that for our example.

```r
mu <- pava(y)
blocks <- split(seq_along(y), mu)
names(blocks) <- NULL
kappa <- lapply(blocks, function(i) cumsum(y[i] - mu[i]))
kappa <- lapply(kappa, zapsmall)
kappa
```

```
## [[1]]
## [1] 0.6444772 0.0000000
##
## [[2]]
## [1] 0
##
## [[3]]
## [1] 0
##
## [[4]]
## [1] 2.365389 0.000000
##
## [[5]]
## [1] 0.1303541 0.0000000
##
## [[6]]
## [1] 3.836762 3.758576 0.000000
##
## [[7]]
## [1] 5.065948 0.000000
##
## [[8]]
## [1] 0
##
## [[9]]
## [1] 0.839142 3.486207 1.623130 0.000000
##
## [[10]]
## [1] 0.7883482 0.0000000
```

```r
all(unlist(kappa) >= 0)
```

```
## [1] TRUE
```

4.12.2.3 Return to Proof

So now what remains to be shown are two things.

- Initialization establishes dual feasibility (running sums nonnegative).
- Dual feasibility (running sums nonnegative) is a loop invariant.

The first of these is trivial. Initialization starts us with all blocks having length one, and the running sum for a block of length one is trivially zero (because for a block of length one we have $\nu = v_j$ by Kuhn-Tucker conditions (a) and (d)).

So it remains to be shown that if running sums are nonnegative for all blocks before pooling, they remain nonnegative for the new pooled block that is created in the PAV step of the algorithm. So now we look at adjacent violator blocks. For some j^*, j^{**}, and j^{***} we have

$$\nu_{j^*-1} \neq \nu_{j^*} = \cdots = \nu_{j^{**}-1} > \nu_{j^{**}} = \cdots = \nu_{j^{***}-1} \neq \nu_{j^{***}}$$

This is in the middle of the computation. We have identified that the block starting at j^* and the block starting at j^{**} are adjacent violators and have decided to pool these blocks. We don't know anything about other blocks. They are not being touched in this iteration.

We also know the mean values for each block are the averages of the response vector for each block. When we pool the mean values for the block starting at j^* will decrease and the mean values for the other block will increase.

Hence the kappas for the block starting at j^* will increase, and the kappas for the other block will decrease. The increase cannot cause them to go negative. So we only have to check what happens in the other block.

Write

$$\begin{aligned} \nu^* &= \nu_{j^*} \\ \nu^{**} &= \nu_{j^{**}} \\ w^* &= \sum_{j=j^*}^{j^{**}-1} w_j \\ w^{**} &= \sum_{j=j^{**}}^{j^{***}-1} w_j \end{aligned}$$

Then the mean for the new pooled block is

$$\nu_{\text{new}} = \frac{w^* \nu^* + w^{**} \nu^{**}}{w^* + w^{**}}$$

The Lagrange multipliers (running sums) are given by (*) above. For the new pooled block these are

$$\kappa_r = \sum_{j=j^*}^{r} w_j (v_j - \nu_{\text{new}}), \qquad j^* \leq r < j^{***}$$

We know these are nonnegative for $j^* \leq r < j^{**}$. We are still trying to show they are nonnegative for $j^{**} \leq r < j^{***}$.

Before pooling we had (by the hypothesis that the loop invariant holds at the top of the loop)

$$\begin{aligned} &\sum_{j=j^*}^{j^{**}-1} w_j (v_j - \nu^*) = 0 \\ &\sum_{j=j^{**}}^{r} w_j (v_j - \nu^{**}) \geq 0, \qquad j^{**} \leq r < j^{***} \end{aligned}$$

So for $j^{**} \leq r < j^{***}$ we have

$$
\begin{aligned}
\sum_{j=j^*}^{r} w_j(v_j - \nu_{\text{new}}) &= \sum_{j=j^*}^{j^{**}-1} w_j(v_j - \nu_{\text{new}}) + \sum_{j=j^{**}}^{r} w_j(v_j - \nu_{\text{new}}) \\
&= \sum_{j=j^*}^{j^{**}-1} w_j(v_j - \nu^*) + \sum_{j=j^*}^{j^{**}-1} w_j(\nu^* - \nu_{\text{new}}) + \sum_{j=j^{**}}^{r} w_j(v_j - \nu^{**}) + \sum_{j=j^{**}}^{r} w_j(\nu^{**} - \nu_{\text{new}}) \\
&\geq \sum_{j=j^*}^{j^{**}-1} w_j(\nu^* - \nu_{\text{new}}) + \sum_{j=j^{**}}^{r} w_j(\nu^{**} - \nu_{\text{new}}) \\
&= w^*(\nu^* - \nu_{\text{new}}) + (\nu^{**} - \nu_{\text{new}}) \sum_{j=j^{**}}^{r} w_j \\
&\geq w^*(\nu^* - \nu_{\text{new}}) + w^{**}(\nu^{**} - \nu_{\text{new}})
\end{aligned}
$$

because $\nu^* > \nu_{\text{new}} > \nu^{**}$. And the last line is zero by definition of ν_{new}. So we have checked the last thing that needed to be checked. The proof is done!

4.13 The Proof and Inexact Computer Arithmetic

Now that we have a proof, we must reconsider it in light of inexact computer arithmetic. Does it describe, even approximately, what will happen when implemented in a real computer with the computer arithmetic provided by the computer hardware (described in the unit on that)?

Here this reconsideration is fairly trivial. First, the proof of termination survives. Each time through the loop, the number of blocks is reduced by one (whether violators are chosen correctly or not), and if we ever get to one block, then there is no violation of primal feasibility and we can stop without checking anything. So PAVA does terminate, even if inexact computer arithmetic is used.

This is actually the big worry. Some other algorithms can work perfectly if real real numbers (with infinite precision) are (hypothetically) used, but infinite loop, crash, or return garbage if inexact computer arithmetic is used.

Second, does PAVA calculate the right answers if inexact computer arithmetic is used? The answer to that is "obviously not," if we mean exactly the right answers; computer arithmetic cannot even represent the exact right answers. Furthermore, detection of violating blocks in the PAV step cannot be done exactly correctly. In general, whenever the computer's so-called "real" numbers are compared for equality or inequality, we can get wrong answers. That is why R has its function `all.equal` to compare numbers with a tolerance. But a tolerance won't help us here. We have to make decisions, and we have to do it based on inexact computer arithmetic. So these decisions may just be wrong. And if we cannot correctly detect adjacent violating blocks in the PAV step, then PAVA may

- either fail to detect adjacent violating blocks that it should pool,
- or incorrectly "detect" and pool non-violating adjacent blocks

and so not have the correct block structure in the answer it calculates.

So how bad is that? The answer is not bad. That is only what we must accept as the price of using inexact arithmetic. With inexact computer arithmetic, when we make either kind of error, the error we make here is small, a few machine epsilons. So we will obtain *almost* the same MLE mean values as if we used infinite precision arithmetic. If we make an incorrect pooling decision (either pool two blocks that we should not have or fail to pool two blocks that we should), then the means for the blocks before pooling were nearly equal (otherwise we would not have made a mistake about the order of their means), hence their means are nearly equal to the mean after pooling. Thus the incorrect pooling decision makes only a negligible difference between the means calculated by the algorithm with exact or inexact arithmetic. Thus we argue that PAVA works even with inexact computer arithmetic. PAVA with inexact arithmetic may get the block structure wrong and any of the equalities or inequalities in the Kuhn-Tucker conditions may fail due to inexactness of computer arithmetic. But the solution (the vector of means that are the MLE) nearly agrees with what would be computed using exact arithmetic.

Even though our re-analysis in this section turns out to find no issues, we had to do it. That there are no problems caused by inexact computer arithmetic is only obvious after we have thought it through.

We just mention that we could use infinite precision rational arithmetic provided by CRAN package gmp, but our analysis in this section says that isn't necessary.

An example of what we were worried about but didn't find is provided by CRAN package rcdd. It does have to use infinite precision rational arithmetic. The help pages for its functions warn

> If you want correct answers, use rational arithmetic. If you do not, this function may (1) produce approximately correct answers, (2) fail with an error, (3) give answers that are nowhere near correct with no error or warning, or (4) crash R losing all work done to that point. In large simulations (1) is most frequent, (2) occurs roughly one time in a thousand, (3) occurs roughly one time in ten thousand, and (4) has only occurred once and only with the 'redundant' function. So the R floating point arithmetic version does mostly work, but you cannot trust its results unless you can check them independently.

The analysis of this section says we do not need such a warning for PAVA.

4.14 Estimating the Variance

For the same reasons as in linear regression, we do not want to use the MLE for the variance, which is, for the same reasons as in linear regression, residual sum of squares divided by n.

We could divide by $n - p$ where p is the number of parameters estimated (number of blocks), but this is iffy because p is a random quantity (depends on the data). So this idea certainly does not give an unbiased estimator of variance (in contrast to linear regression where it does give an unbiased estimator).

Nor does the usual theory that gives t and F distributions go through. All the sampling distributions are known (Barlow *et al.*, 1972; Robertson *et al.*, 1988), but going into all that would take us too much into theory and too far from computing.

4.15 Exponential Family Response

PAVA also does isotonic logistic regression or isotonic Poisson regression (log link) or isotonic regression when the response distribution is any one-parameter exponential family. The monotonicity constraints can be imposed on either mean value parameters or canonical parameters (because the link function that maps between them is necessarily monotone).

So here we just do an example, isotonic logistic regression (the place kicker problem). These are the data for kicker Dan Bailey of the Minnesota Vikings in 2018 (source: http://www.nflsavant.com/about.php).

```r
distance <- c(37,39,40,28,37,45,22,52,37,48,26,42,22,43,39,36,36,48,56,
    37,48,39,47,36,34,24,29,45)
response <- c(1,1,1,0,1,0,1,1,1,1,1,0,1,1,1,1,1,0,0,1,0,1,0,1,1,1,1,1)
```

Since R function `pava` in CRAN package `Iso` that we used before does not easily handle repeated predictor values (it does handle them but the user has to put the data in the form it wants, essentially do all the work described in the section about dealing with duplicate predictor values above), we will use R function `isoreg` in the R core. It too has its infelicities, but they are not quite so annoying. One infelicity is that it cannot do decreasing, so we regress on minus distance rather than distance. A second infelicity is that it returns the predicted mean values corresponding to the ordered predictor values, not the given predictor values. However, it also gives the order of the predictor values.

```r
iout <- isoreg(- distance, response)
plot(distance, response, ylab = "response", xlab = "distance (yards)")
points(distance[iout$ord], iout$yf, pch = 19)
```

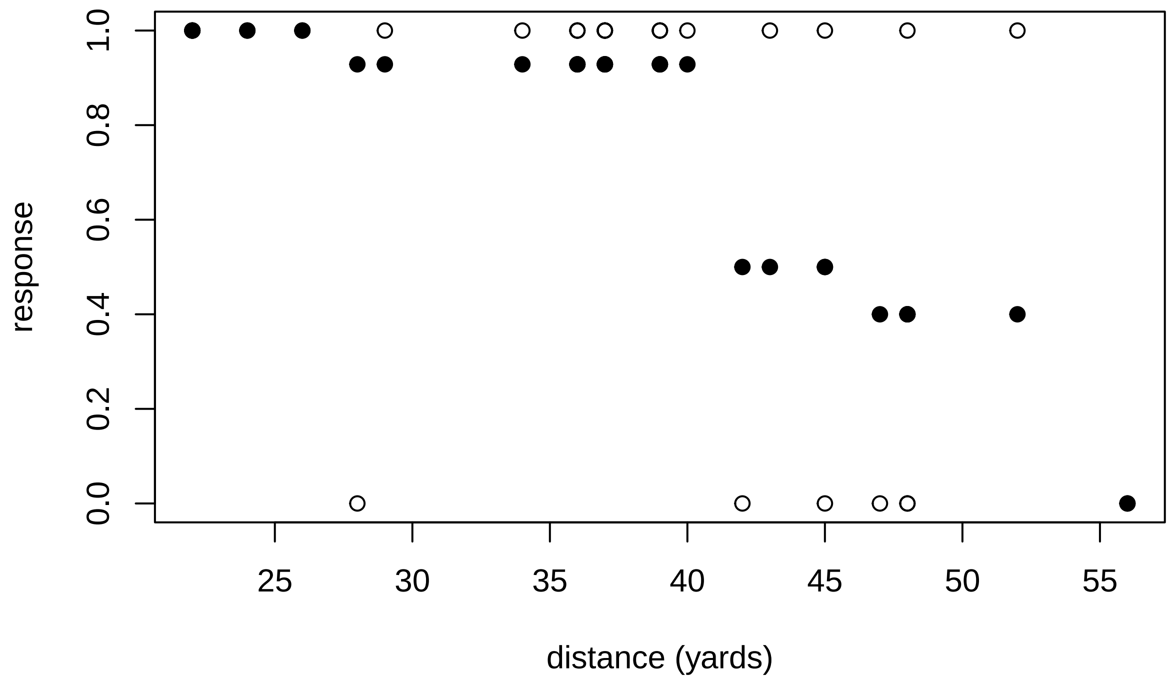

Figure 2: Data (hollow dots) and Isotonic Regression (solid dots)

4.16 Philosophy

Note that the isotonic regression assumption is undisputable. The response is Bernoulli, and we know the success probability decreases with distance. Note that we are using mean value parameters to model the Bernoullis, so no assumption is made about link functions. Now consider than any parametric assumptions about the mean function would be highly disputable. And any nonparametric methods (smoothing splines or whatever) would also be disputable. You can do something else, but only isotonic regression is philosophically clean.

5 Convex Programming

We say the problem described in the section defining the problem the Lagrange multiplier methodology solves above is a *convex programming* problem if

- the objective function f is convex,
- the constraint functions g_i are convex for the inequality constraints (for $i \in I$), and
- the constraint functions g_i are affine for the equality constraints (for $i \in E$).

For a convex programming problem, there is no difference between the conditions in our Theorem 3.1 and the Kuhn-Tucker conditions. This is because of the subgradient inequality. For any convex function f and any point x where its derivative exists

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle, \qquad \text{for all } y$$

The right-hand side of this equality is the best affine approximation to the function near x, the Taylor series for f expanding around x with only constant and linear terms kept. So if $\nabla f(x) = 0$, then $f(y) \geq f(x)$ for all y.

For a convex programming problem, the Lagrangian is a convex function of x. Hence any point where the gradient of the Lagrangian is zero is a global minimizer of the Lagrangian.

A second issue about convex programming problems is whether we are guaranteed that the method of Lagrange multipliers can always find a solution.

Rockafellar (1970) is the authoritative source for the theory of convex optimization. It has multiple theorems about the existence of Lagrange multipliers. One which tells us about isotonic regression is Corollary 28.2.2, which says that if a problem has a solution (the minimum exists) and all of the constraints are affine, then the method of Lagrange multipliers works.

So this tells us that PAVA is guaranteed to find a global maximizer of the log likelihood. Moreover, since the log likelihood is strictly concave, this global maximizer is unique.

Bibliography

Barlow, R. E., Bartholomew, D. J., Bremner, J. M., et al. (1972) *Statistical Inference Under Order Restrictions: The Theory and Application of Isotonic Regression.* London: Wiley.

Dijkstra, E. W. (1976) *A Discipline of Programming.* Englewood Cliffs NJ: Prentice-Hall.

Feijen, W. H. J. and van Gasteren, A. J. M. (1999) *On a Method of Multiprogramming.* New York: Springer.

Misra, J. (2001) *A Discipline of Multiprogramming: Programming Theory for Distributed Applications.* New York: Springer.

Robertson, T., Wright, F. T. and Dykstra, R. (1988) *Order Restricted Statistical Inference.* Chichester: Wiley.

Rockafellar, R. T. (1970) *Convex Analysis.* Princeton: Princeton University Press.

Shapiro, J. F. (1979) *Mathematical Programming: Structures and Algorithms.* New York: Wiley.