Statistical Machine Learning Notes 9

Boosting

Instructor: Justin Domke

1 Additive Models

The basic idea of boosting is to greedily build *additive* models. Let $b_m(\mathbf{x})$ be some predictor (a tree, a neural network, whatever), which we will call a base learning. In boosting, we will build a model that is the sum of base learners as

$$f(\mathbf{x}) = \sum_{m=1}^{M} b_m(\mathbf{x}).$$

The obvious way to fit an additive model is *greedily*. Namely, with start with the simple function $f_0(\mathbf{x}) = 0$, then iteratively add base learners to minimize the risk of $f_{m-1}(\mathbf{x}) + b_m(\mathbf{x})$.

Forward stagewise additive modeling

- $f_0(\mathbf{x}) \leftarrow 0$
- For $m = 1, ..., M$
  - $b_m \leftarrow \arg\min_b \sum_{(\hat{\mathbf{x}}, \hat{y})} L(f_{m-1}(\hat{\mathbf{x}}) + b(\hat{\mathbf{x}}), \hat{y})$
  - $f_m(\mathbf{x}) \leftarrow f_{m-1}(\mathbf{x}) + v\, b_m(\mathbf{x})$

Notice here that once we have fit a particular base learner, it is "frozen in", and is not further changed. Once can also create procedures for additive modeling that re-fit the base learners several times.

2 Additive Regression Trees

It is quite easy to do forward stagewise additive modeling when we are interested in minimizing the least-squares regression loss.

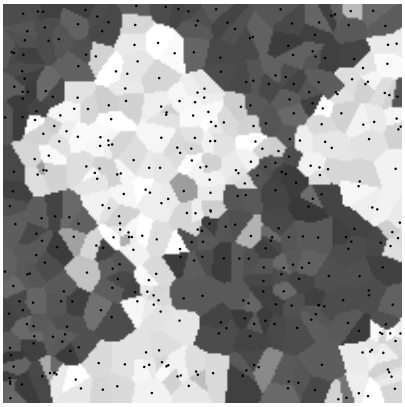
$$\begin{aligned} \arg\min_b \sum_{(\hat{\mathbf{x}},\hat{y})} L\big(f_{m-1}(\hat{\mathbf{x}}) + b(\hat{\mathbf{x}}), \hat{y}\big) &= \arg\min_b \sum_{(\hat{\mathbf{x}},\hat{y})} \big(f_{m-1}(\hat{\mathbf{x}}) + b(\hat{\mathbf{x}}) - \hat{y}\big)^2 \\ &= \arg\min_b \sum_{(\hat{\mathbf{x}},\hat{y})} \big(b(\hat{\mathbf{x}}) - r(\hat{\mathbf{x}}, \hat{y})\big)^2 \end{aligned} \tag{2.1}$$

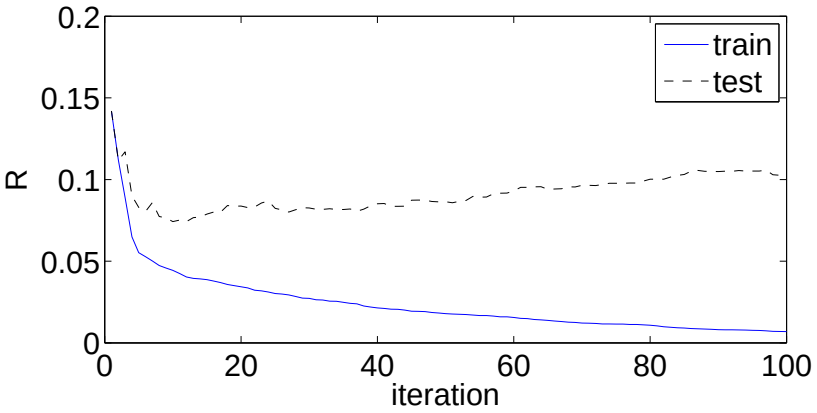
where $r(\hat{\mathbf{x}}, \hat{y}) = \hat{y} - f_{m-1}(\hat{\mathbf{x}})$.

However, minimizing a problem like Eq. 2.1 is exactly what all our previous regression methods do. Thus, we can apply any method for least-squares regression essentially unchanged. The only difference is that the target values are given by $\hat{y} - f_{m-1}(\hat{\mathbf{x}})$– the difference between $\hat{y}$ and the additive model so far.

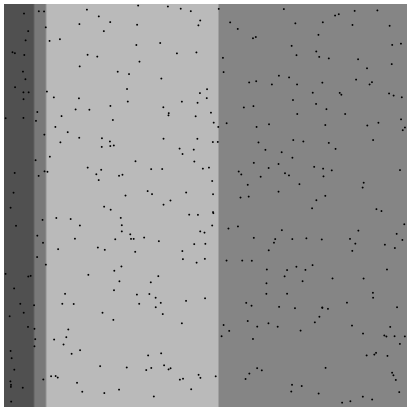
Here are some examples applying this to two datasets. Here (as in all the examples in these notes) our base learner is a regression tree with two levels of splits.

data

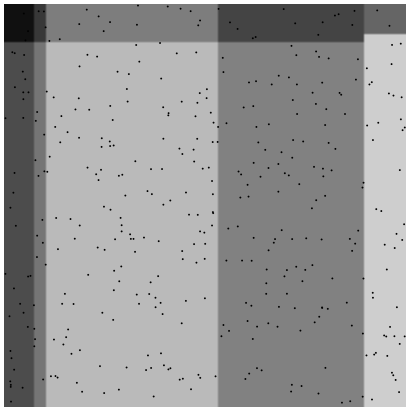




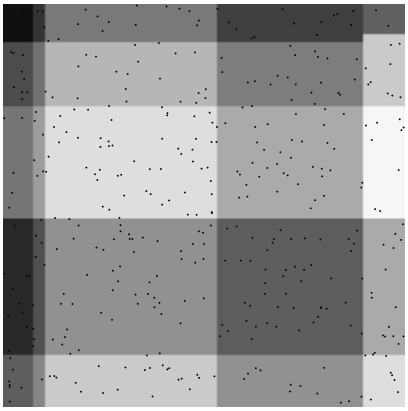
1 iteration



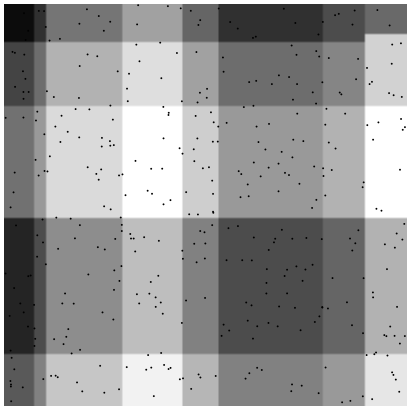
2 iterations



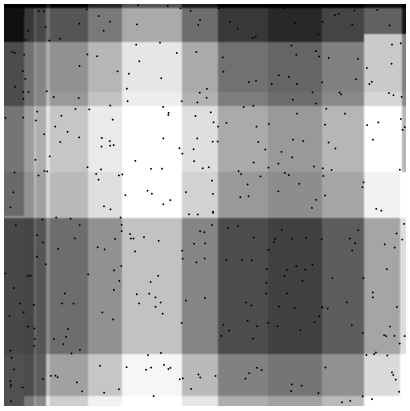
3 iterations



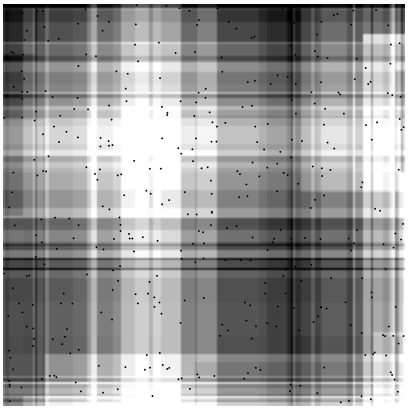
4 iterations

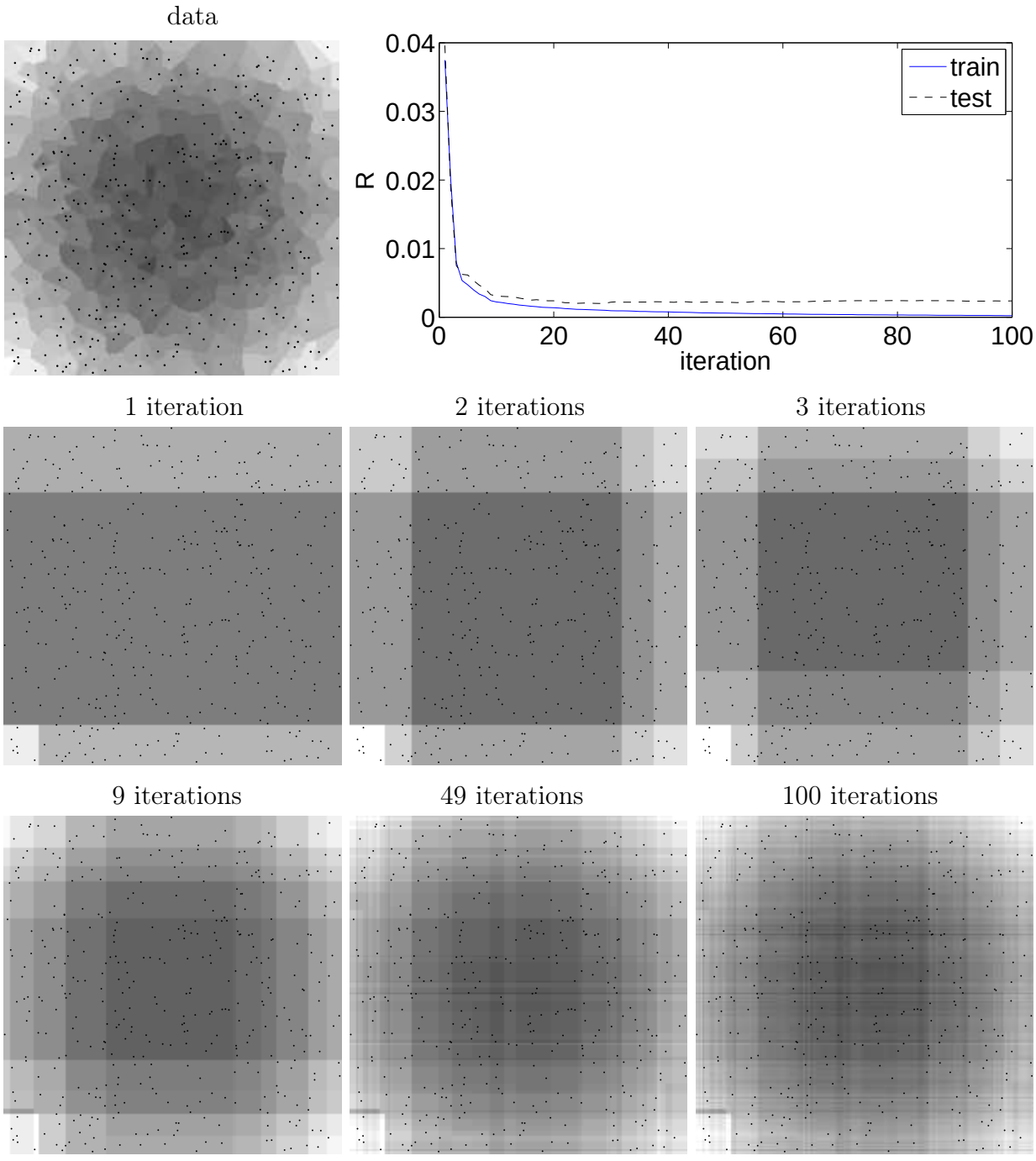


9 iterations



50 iterations





3 Boosting

Suppose we want to minimize some other loss function. Unfortunately, if we aren't using the least-squares loss, the optimization

$$\arg\min_b \sum_{(\hat{\mathbf{x}},\hat{y})} L(f_{m-1}(\hat{\mathbf{x}}) + b(\hat{\mathbf{x}}), \hat{y}) \tag{3.1}$$

will not reduce to a least-squares problem. One alternative would be to use a different base-learner, that fits a different loss. With boosting, we don't want to change our base learner. This could be because it is simply inconvenient to create a new algorithm for a different loss function. The general idea of boosting is to fit a new base learner b to, if not *minimize* the risk for f_m, to *decrease* it. As we will see, we can do this for a large range of loss functions, all based on a least-squares base learner.

4 Gradient Boosting

Gradient boosting is a very clever algorithm. In each iteration, we set as target values *the negative gradient of the loss* with respect to f. So, if the base learner closely matches the target values, when we add some multiple v of the base learner to our additive model, it should decrease the loss.

Gradient Boosting

- $f_0(\mathbf{x}) \leftarrow 0$
- For $m = 1, ..., M$
  - For all$(\hat{\mathbf{x}}, \hat{y})$
    - $r(\hat{\mathbf{x}}, \hat{y}) \leftarrow -\frac{d}{df} L(f_{m-1}(\hat{\mathbf{x}}), \hat{y})$
  - $b_m \leftarrow \arg\min_b \sum_{(\hat{\mathbf{x}},\hat{y})} \big(b(\hat{\mathbf{x}}) - r(\hat{\mathbf{x}}, \hat{y})\big)^2$
  - $f_m(\mathbf{x}) \leftarrow f_{m-1}(\mathbf{x}) + v\, b_m(\mathbf{x})$

Consider changing $f(\mathbf{x})$ to $f(\mathbf{x}) + vb(\mathbf{x})$ for some small v. We want to minimize the loss, i.e.

$$\sum_{(\hat{\mathbf{x}},\hat{y})} L(f_{m-1}(\hat{\mathbf{x}}) + b(\hat{\mathbf{x}}), \hat{y}) \approx \sum_{(\hat{\mathbf{x}},\hat{y})} L(f_{m-1}(\hat{\mathbf{x}}), \hat{y}) + \sum_{(\hat{\mathbf{x}},\hat{y})} \frac{dL(f_{m-1}(\hat{\mathbf{x}}), \hat{y})}{df} b(\hat{\mathbf{x}}). \tag{4.1}$$

However, it doesn't make much sense to simply minimize the right hand side of Eq. 4.1, since we can always make the decrease bigger by multiplying b by a larger constant. Instead, we choose to minimize

$$\sum_{(\hat{\mathbf{x}},\hat{y})} \frac{dL(f_{m-1}(\hat{\mathbf{x}}),\hat{y})}{df} b(\hat{\mathbf{x}}) + \frac{1}{2}\sum_{(\hat{\mathbf{x}},\hat{y})} b(\hat{\mathbf{x}})^2$$

which prevents b from becoming too large. It is easy to see that

$$\begin{aligned}
\arg\min_b \sum_{(\hat{\mathbf{x}},\hat{y})} \frac{dL(f_{m-1}(\hat{\mathbf{x}}),\hat{y})}{df} b(\hat{\mathbf{x}}) + \frac{1}{2}\sum_{(\hat{\mathbf{x}},\hat{y})} b(\hat{\mathbf{x}})^2 &= \arg\min \sum_{(\hat{\mathbf{x}},\hat{y})} (b(\hat{\mathbf{x}}) + \frac{-dL(f_{m-1}(\hat{\mathbf{x}}),\hat{y})}{df})^2 \\
&= \arg\min \sum_{(\hat{\mathbf{x}},\hat{y})} \big(b(\hat{\mathbf{x}}) - r(\hat{\mathbf{x}},\hat{y})\big)^2
\end{aligned}$$

Some example loss functions, and their derivatives:

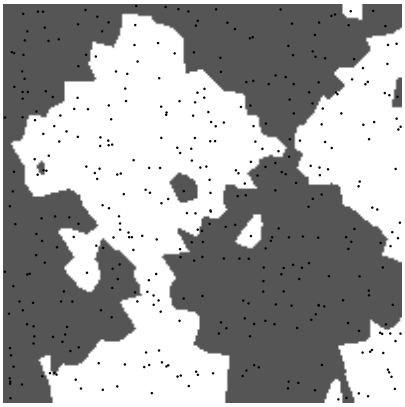
- least-squares. $L = (f-\hat{y})^2 \rightarrow dL/fd = 2(f-\hat{y})$
- least-absolute-deviation. $L = |f-\hat{y}| \rightarrow dL/df = \text{sign}(f-\hat{y})$
- logistic regression. $L = \log(1+\exp(-\hat{y}f)) \rightarrow dL/df = \frac{-\hat{y}}{1+\exp(f\hat{y})}$.
- exponential loss. $L = \exp(-\hat{y}f) \rightarrow dL/df = -\hat{y}\exp(-\hat{y}f)$

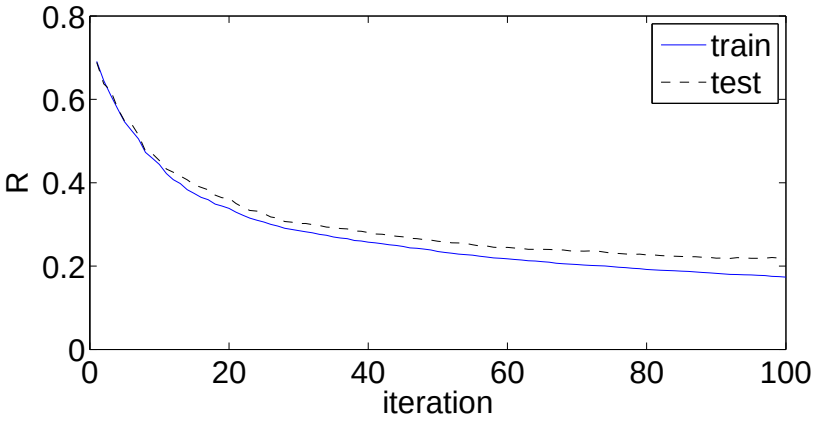
Fun fact: gradient boosting with a step size of $v = \frac{1}{2}$ on the least-squares loss gives exactly the forward stagewise modeling algorithm we devised above.

Here are some examples of gradient boosting on classification losses (logistic and exponential). The figures show the function $f(\mathbf{x})$. As usual, we would get a predicted class by taking the sign of $f(\mathbf{x})$.

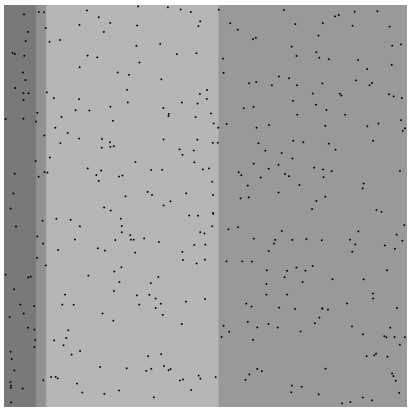
Gradient Boosting, Logistic Loss, step size of $v = 1$.

data

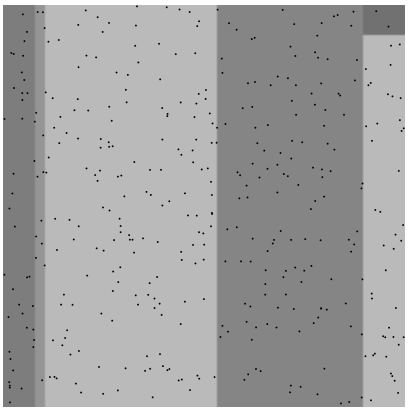




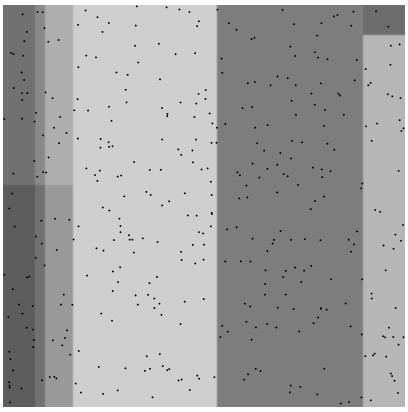
1 iteration



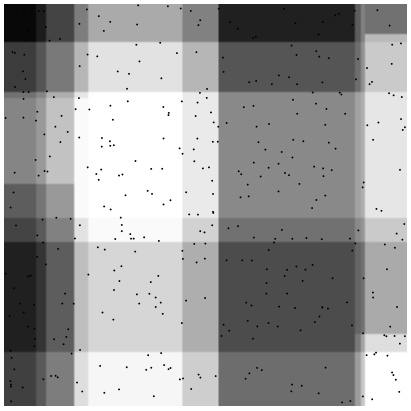
2 iterations



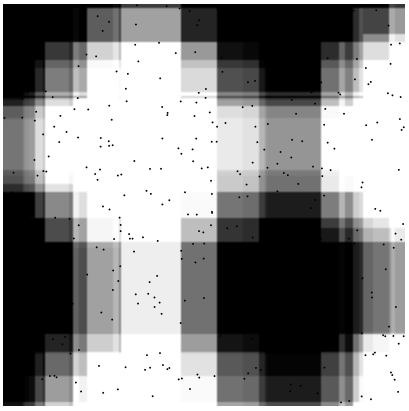
3 iterations



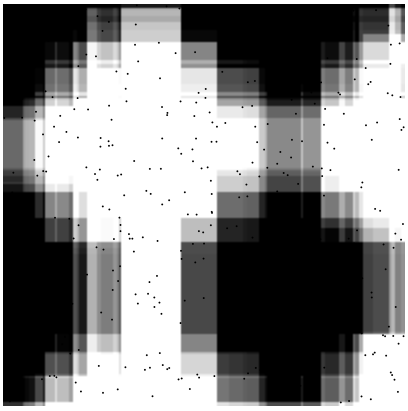
9 iterations



49 iterations

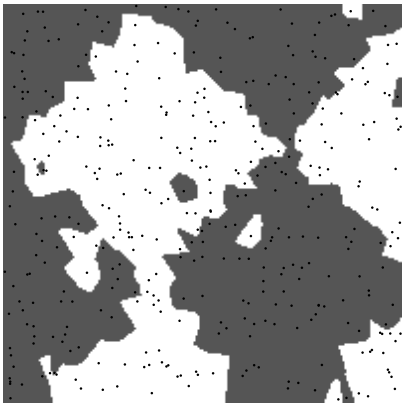


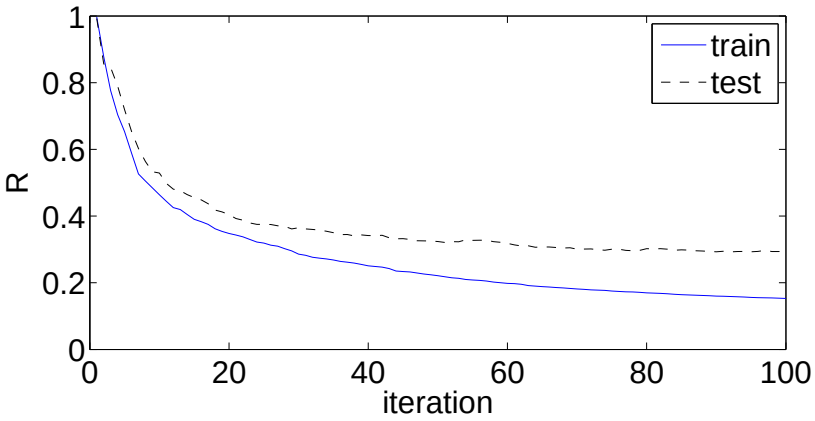
100 iterations



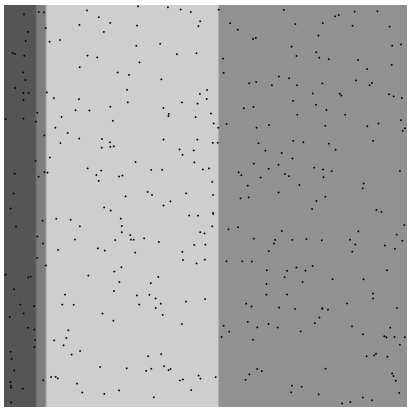
Gradient Boosting, Exponential loss, step size of $v = 1$.

data

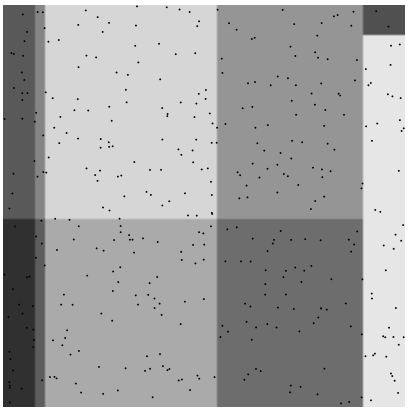




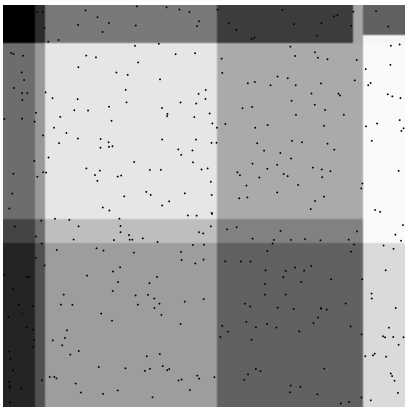
1 iteration



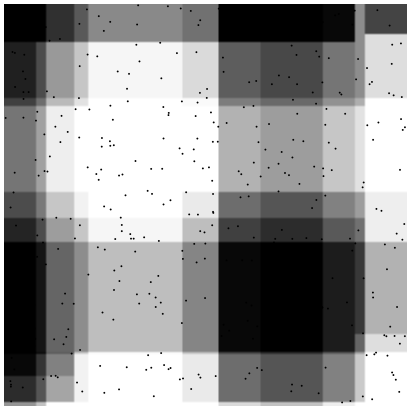
2 iterations



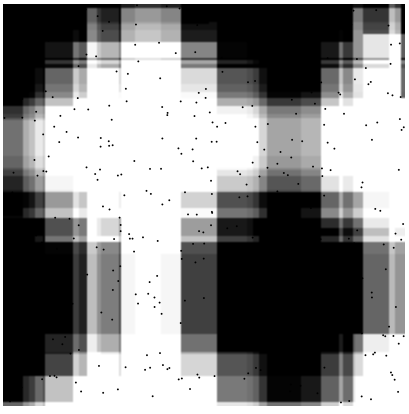
3 iterations



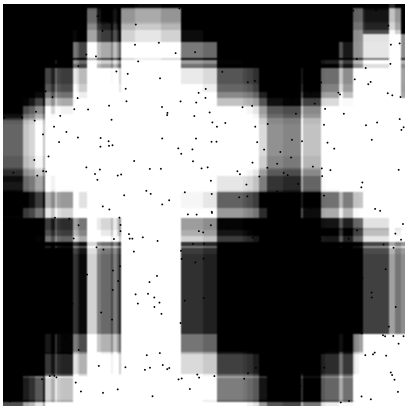
9 iterations



49 iterations



100 iterations



5 Newton Boosting

The basic idea of Newton boosting is, instead of just taking the gradient, to build a local *quadratic model*, and then to minimize it. We can again do this using any least-squares regressor as a base learner.

(Note that "Newton boosting" is not generally accepted terminology. The general idea of fitting to a quadratic optimization is used in several algorithms for specific loss function (e.g. Logitboost), but doesn't seem to have been given a name. The class voted to use "Newton Boosting" in preference to "Quadratic Boosting".)

Newton Boosting

- $f_0(\mathbf{x}) \leftarrow 0$
- For $m = 1, ..., M$
  - For all$(\hat{\mathbf{x}}, \hat{y})$
    - $h(\hat{\mathbf{x}}, \hat{y}) \leftarrow \dfrac{d^2}{df^2} L(f_{m-1}(\hat{\mathbf{x}}), \hat{y})$
    - $g(\hat{\mathbf{x}}, \hat{y}) \leftarrow \frac{d}{df} L(f_{m-1}(\hat{\mathbf{x}}), \hat{y})$
    - $r(\hat{\mathbf{x}}, \hat{y}) \leftarrow -\dfrac{1}{h(\hat{\mathbf{x}}, \hat{y})} \dfrac{d}{df} L(f_{m-1}(\hat{\mathbf{x}}), \hat{y})$
  - $b_m \leftarrow \arg\min_b \sum_{(\hat{\mathbf{x}}, \hat{y})} h(\hat{\mathbf{x}}, \hat{y}) \big(b(\hat{\mathbf{x}}) - r(\hat{\mathbf{x}}, \hat{y})\big)^2$
  - $f_m(\mathbf{x}) \leftarrow f_{m-1}(\mathbf{x}) + v\, b_m(\mathbf{x})$

To understand this, consider search searching for the b that minimizes the empirical risk for f_m. To start with, we make a local second-order approximation. (Here, $g(\hat{\mathbf{x}}, \hat{y}) = \frac{dL(f_{m-1}(\hat{\mathbf{x}}), \hat{y})}{df}$, and $h(\hat{\mathbf{x}}, \hat{y}) = \frac{d^2 L(f_{m-1}(\hat{\mathbf{x}}), \hat{y})}{df^2}$.)

$$
\begin{aligned}
\arg\min_b \sum_{(\hat{\mathbf{x}}, \hat{y})} L(f_{m-1}(\hat{\mathbf{x}}) + b(\hat{\mathbf{x}}), \hat{y}) &\approx \arg\min_b \sum_{(\hat{\mathbf{x}}, \hat{y})} \big(L(f_{m-1}(\hat{\mathbf{x}}), \hat{y}) + b(\hat{\mathbf{x}}) g(\hat{\mathbf{x}}, \hat{y}) + \frac{1}{2} b(\hat{\mathbf{x}})^2 h(\hat{\mathbf{x}}, \hat{y})\big) \\
&= \arg\min_b \frac{1}{2} \sum_{(\hat{\mathbf{x}}, \hat{y})} h(\hat{\mathbf{x}}, \hat{y}) \big(2 \frac{g(\hat{\mathbf{x}}, \hat{y})}{h(\hat{\mathbf{x}}, \hat{y})} b(\hat{\mathbf{x}}) + b(\hat{\mathbf{x}})^2\big) \\
&= \arg\min_b \sum_{(\hat{\mathbf{x}}, \hat{y})} h(\hat{\mathbf{x}}, \hat{y}) \big(b(\hat{\mathbf{x}}) - \frac{-g(\hat{\mathbf{x}}, \hat{y})}{h(\hat{\mathbf{x}}, \hat{y})}\big)^2 \\
&= \arg\min_b \sum_{(\hat{\mathbf{x}}, \hat{y})} h(\hat{\mathbf{x}}, \hat{y}) \big(b(\hat{\mathbf{x}}) - r(\hat{\mathbf{x}}, \hat{y})\big)^2
\end{aligned}
$$

This is a *weighted* least-squares regression, with weights $h(\hat{\mathbf{x}}, \hat{y})$ and targets $\frac{-g(\hat{\mathbf{x}},\hat{y})}{h(\hat{\mathbf{x}},\hat{y})}$.

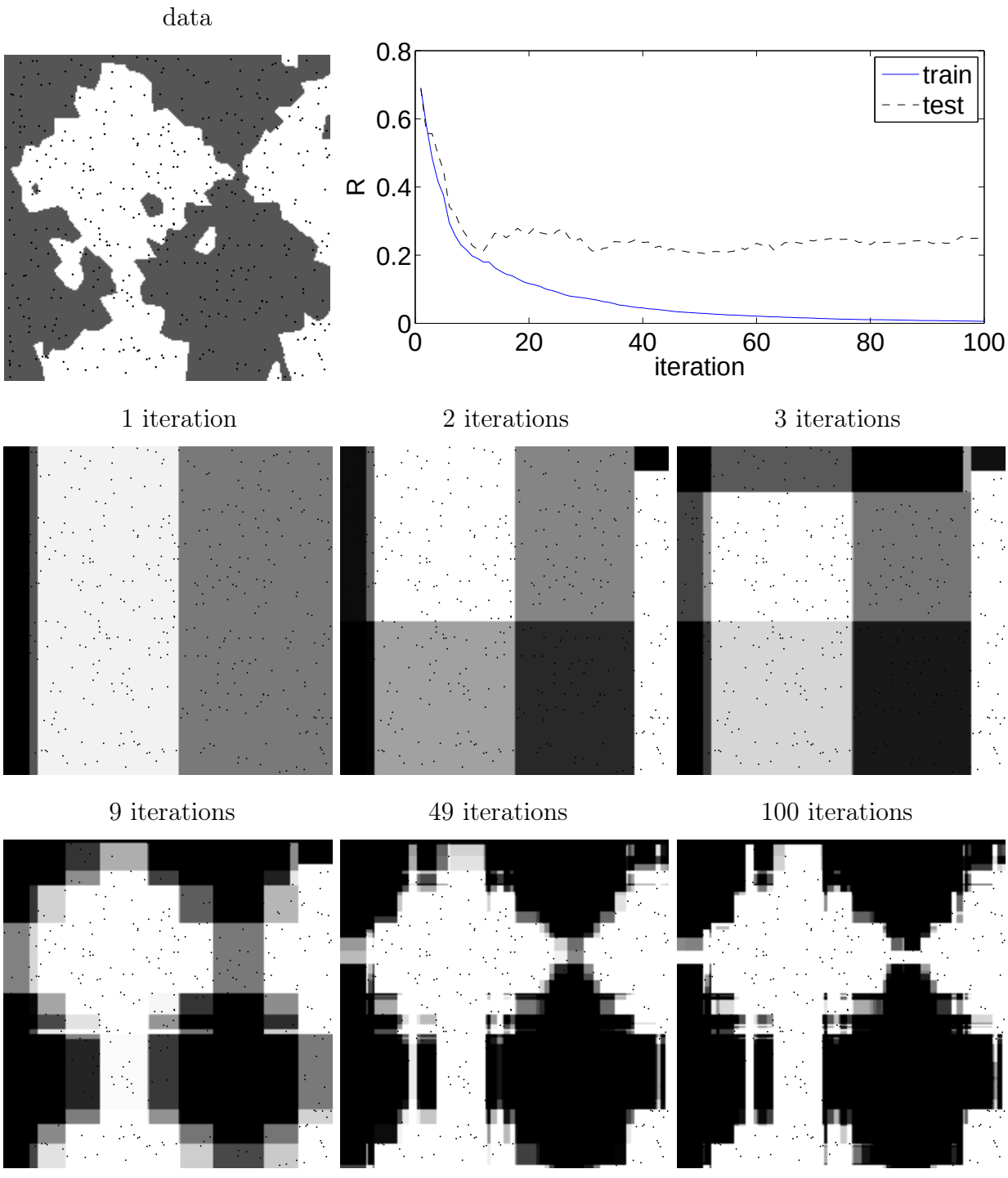
Here, we collect in a table all of the loss functions and derivatives we might use.

Loss Functions $L(f, \hat{y})$

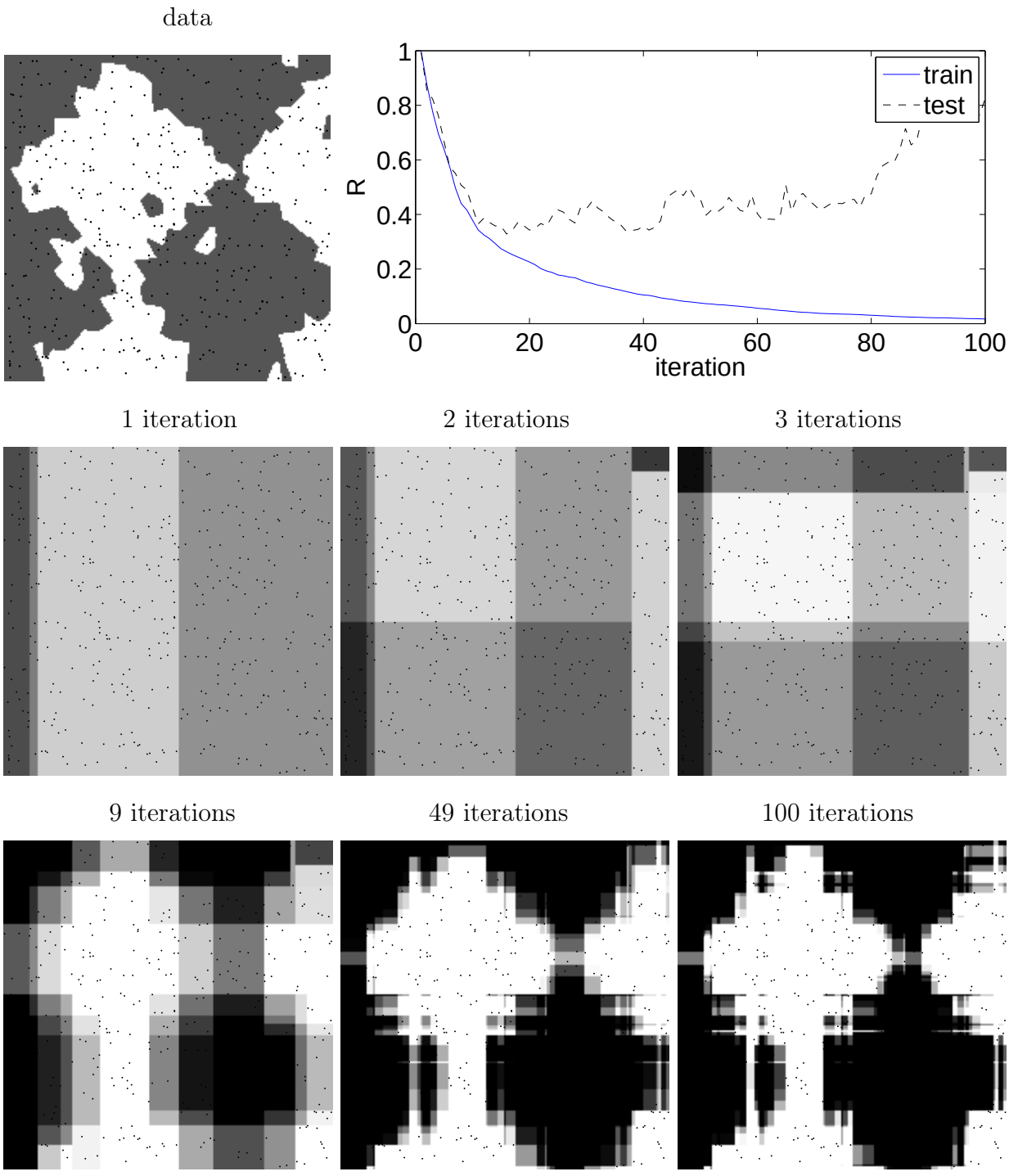
Name	L	$\frac{dL}{df}$	$\frac{d^2L}{df^2}$
least-squares	$(f-\hat{y})^2$	$2(f-\hat{y})$	2
least-absolute-deviation	$\lvert f'-\hat{y}\rvert$	$\operatorname{sign}(f'-\hat{y})$	
logistic	$\log\big(1+\exp(-\hat{y}f)\big)$	$\frac{-\hat{y}}{1+\exp(f\hat{y})}$	$\frac{\hat{y}^2}{2+\exp(f\hat{y})+\exp(-f\hat{y})}$
exponential	$\exp(-\hat{y}f)$	$-\hat{y}\exp(-\hat{y}f)$	$y^2\exp(-f\hat{y})$

Here are some experiments using Newton boosting on the same dataset as above. Generally, we see that Newton boosting is more effective than gradient boosting in quickly reducing the training error. However, this does also means it begins to overfit after a smaller number of iterations. Thus, if running time is an issue (either at training or test time), Newton boosting may be preferable, since it more quickly reduces training error.

Newton Boosting, Logistic Loss, step size of $v = 1$.



Newton Boosting, Logistic Loss, step size of $v = 1$.

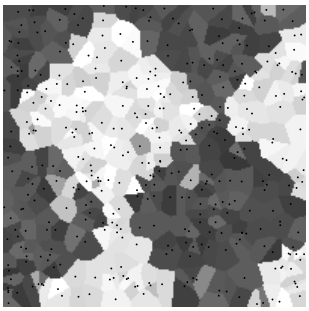


6 Shrinkage and Early Stopping

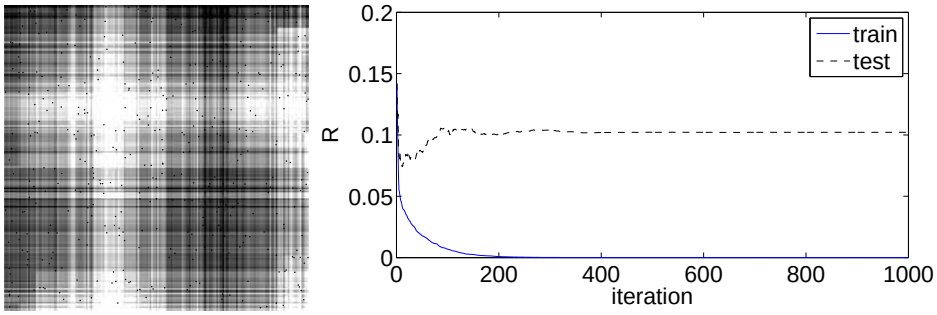
Now, we have not talked about regularization. What is suggested in practice is to regularize by "early stoppping". Namely, repeatedly boost, but hold out a test set. Select the number of base learners M in the model by the number that leads the minimum error on the test set. (It would be best to select this M by cross validation, then re-fit with all models, though this is often not done is practice to reduce running times.)

Notice that we have another parameter controlling complexity, however: the step size v. For a small step size, we will need more models M to achieve the same complexity.

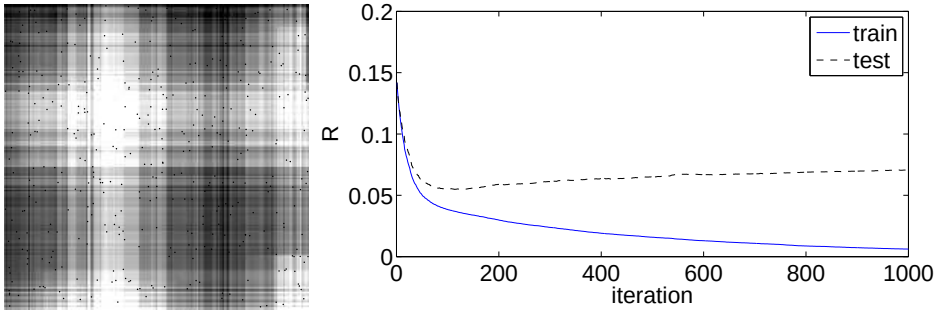
Experimentally, better results are almost always given by choosing a small step size v, and a larger number of base learners M. The following experiments suggest that this is a product of the fact that it yields "smoother" models. Note also from the following experiments that Newton boosting with a given step size performs similarly to gradient boosting with a smaller step size.



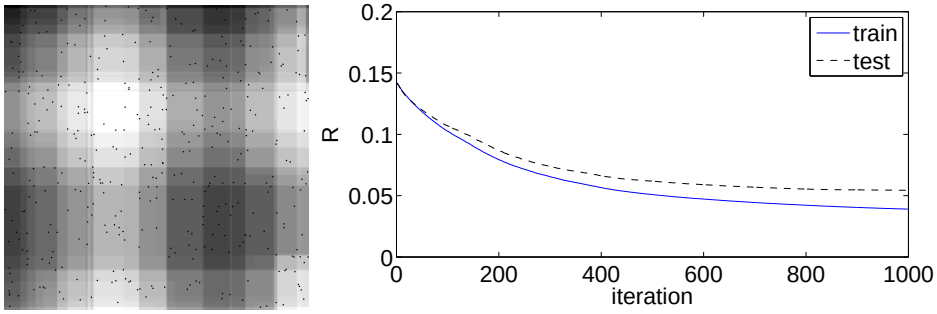
Gradient boosting, least-squares loss, $v = \frac{1}{2}$



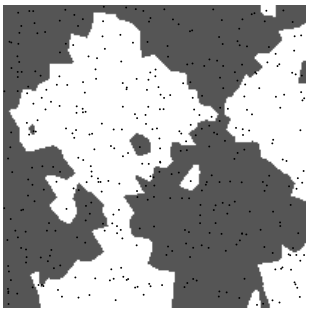
Gradient boosting, least-squares loss, $v = \frac{1}{20}$



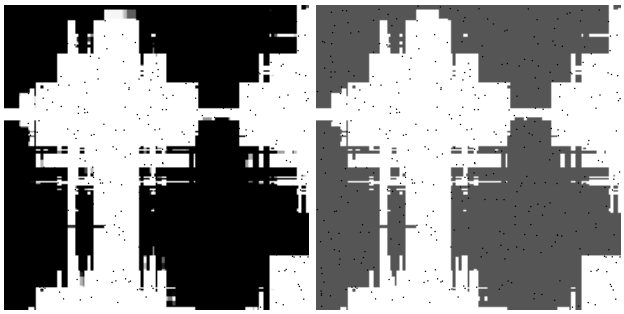
Gradient boosting, least-squares loss, $v = \frac{1}{200}$

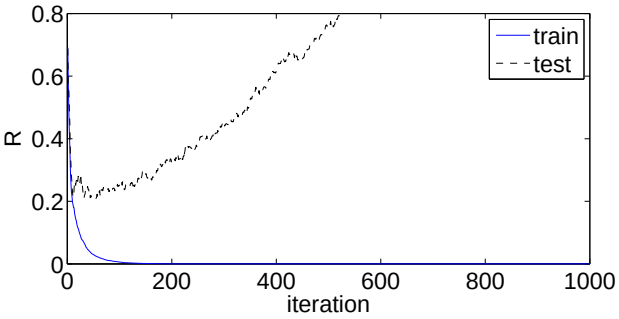


In the following figures, the left column shows $f(\mathbf{x})$ after 100 iterations, while the middle column shows $\text{sign}(f(\mathbf{x}))$.

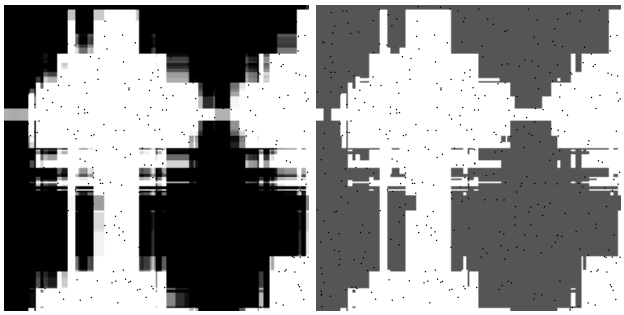


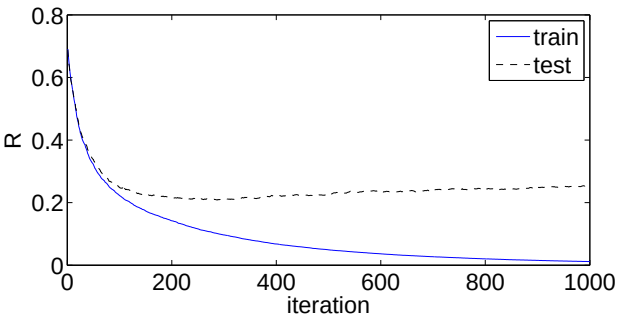
Newton boosting, logistic loss, $v = 1$



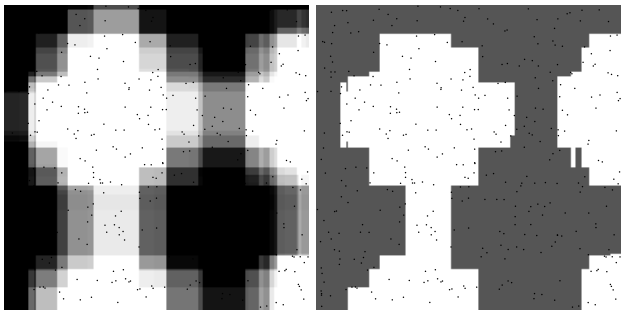


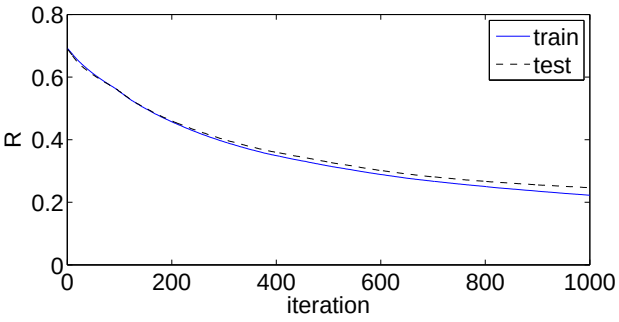
Newton boosting, logistic loss,$v = \frac{1}{10}$



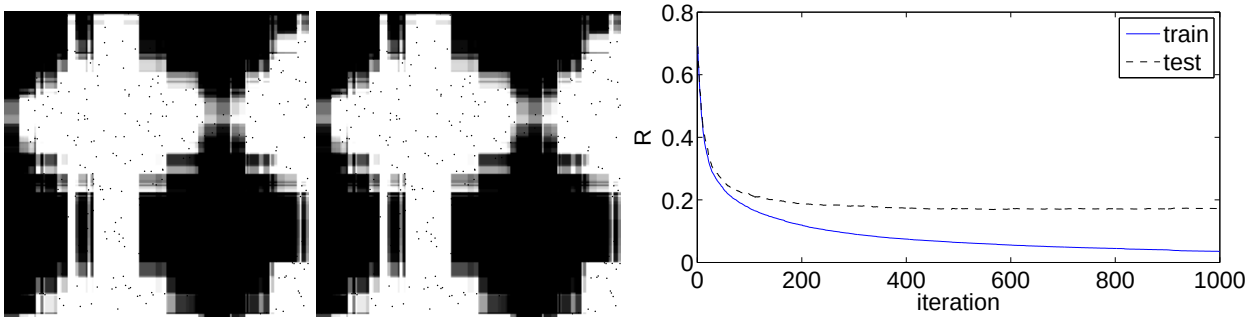


Newton boosting, logistic loss,$v = \frac{1}{100}$

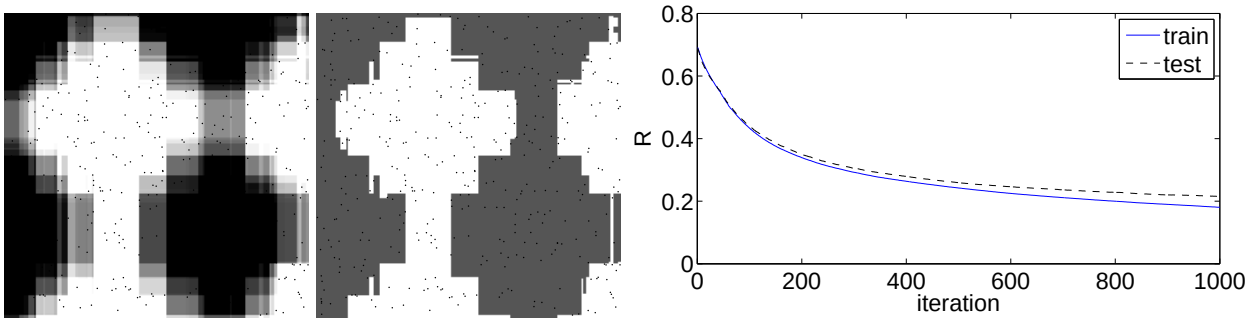




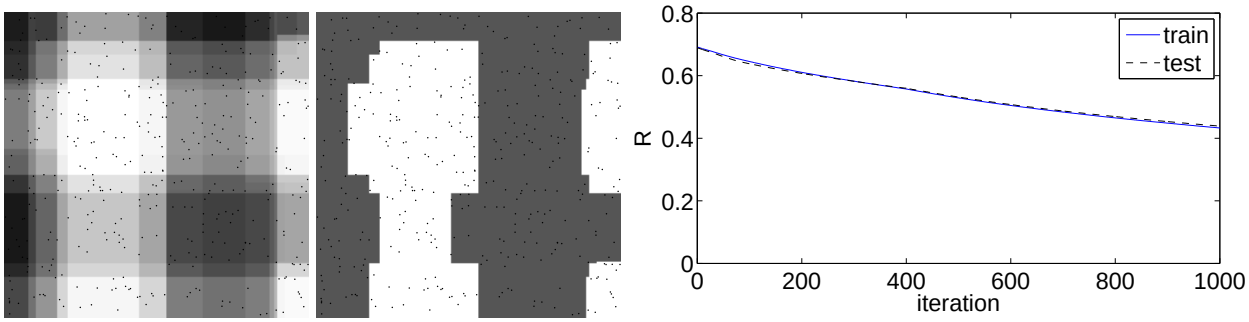
Gradient boosting, logistic loss,$v = 1$



Gradient boosting, logistic loss, $v = \frac{1}{10}$



Gradient boosting, logistic loss, $v = \frac{1}{100}$



7 Discussion

Note that while our experiments here have used trees, any least-squares regressor could be used at the base learner. If so inclined, we could even use *a boosted regression tree* as our base learner.

Most of these loss functions can be extended to multi-class classification. This works by, in each iteration, not fitting a single tree, but C trees, where there are C classes. For newton boosting, this is made tractable by taking a diagonal approximation of the Hessian.

Our presentation here has been quite ahistorical. The first boosting algorithms were presented from a very different, and more theoretical perspective.