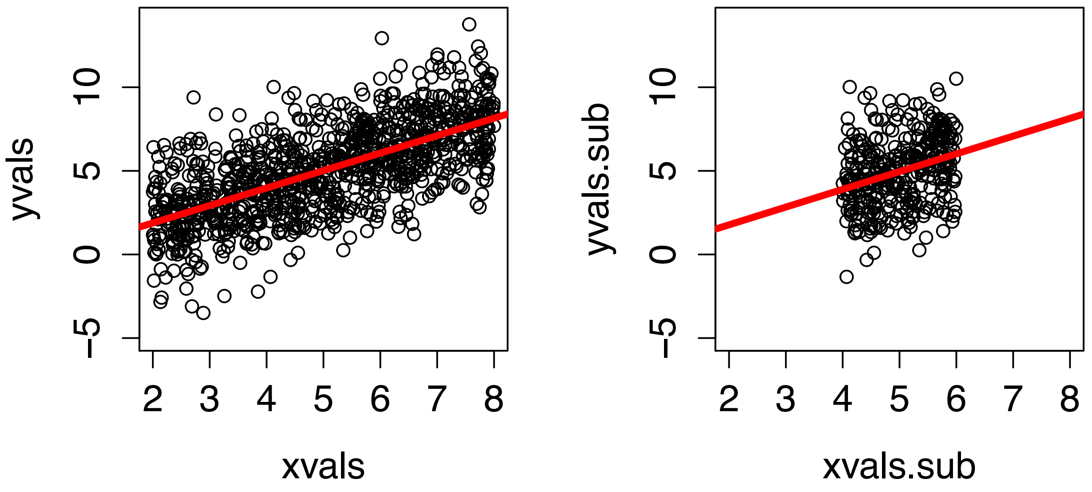
Lecture 13. Prediction, cross-validation, model selection

"Prediction is very difficult, especially about the future" – variously attributed to Niels Bohr, Yogi Berra, others

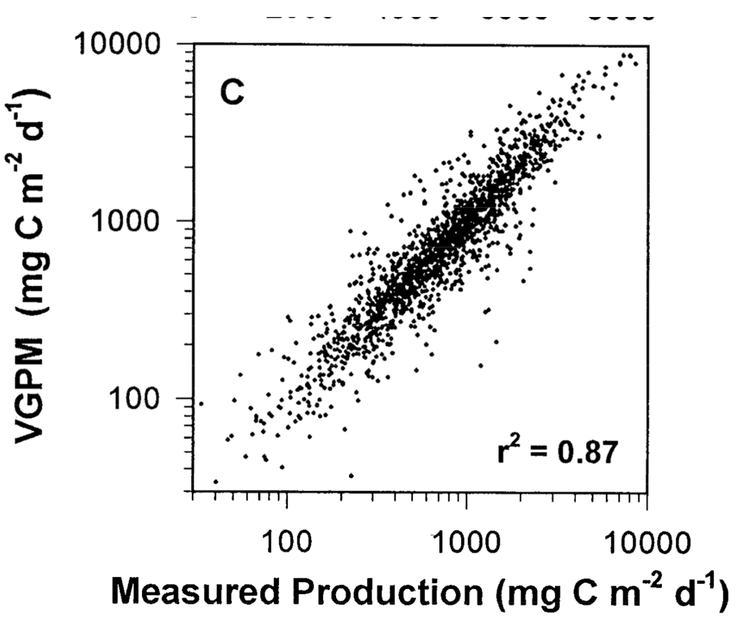
So far we've spent a lot of time doing inference on the individual predictors that we put in a model, looking at coefficient estimates, fitted means and slopes and their confidence intervals, and likelihood ratio tests. We've spent less time evaluating the model as a whole. This emphasis is in part due to my own bias (I am almost always interested in particular predictors, not the whole model), and in part because that's how academic scientists tend to use statistical models. Nonetheless, thinking about the whole model in terms of 'goodness' of fit and predictive performance is important for many applications where the goal is to make predictions, and even when the focus is not on prediction, the predictive performance of models is closely related to methods for model selection.

When we a fit a model to data, there are several different ways to think about how well the model 'fits' the data. An important one, which we've spent plenty of time on already, is whether the data agrees with the assumptions of the model. These include the deterministic structure of the model (whether the linear or nonlinear functions that relate the predictors to the response are accurate descriptions of the relationship) and the stochastic part of the model (are the data independently distributed; is the residual variation homoscedastic).

We can also ask how well does the model 'explain' the data. The classic R^2 is the most common way to quantify this aspect of the model, and earlier I introduced one pseudo-R^2 for GLMs, $1-\frac{\text{residual deviance}}{\text{null deviance}}$, which is one option among many pseudo options. R^2 certainly good for a rough sense of how much variation in the data is explained by the model, though there are some limitations that should be kept in mind. One of these is the fact that R^2 will always increase as more terms are added to a model, regardless of whether those terms are 'real' or only explaining subtle random patterns in the data. This is the issue of overfitting, which we will return to shortly. Another issue is that R^2 can be greatly affected by the range of the data. Consider the two regressions below:



For the plot on the left, I simulated 1000 values with an intercept of 0, a slope of 1, and a residual standard deviation of 2. The fitted regression has an R^2 of ~0.44. For the plot on the right, I used the same simulated data but took the subset for which the x-values range from 4 to 6. This regression has essentially the same coefficient estimates, and the same amount of residual variation, but R^2 ~ 0.09. Why does the plot on the left have a much larger R^2? Because the range of the data is greater, the total variance in the data is large relative to the unexplained (residual) variance. Recall the formula for R^2: $1 - \frac{\text{residual sums of squares}}{\text{total sums of squares}}$. If there is a certain amount of residual variance that is not explained by the model, then increasing the total range of the data will increase the total variance and decrease R^2. Here is an empirical example, from a model used to predict primary production in the ocean based on chlorophyll concentration and some other easily measured variables:

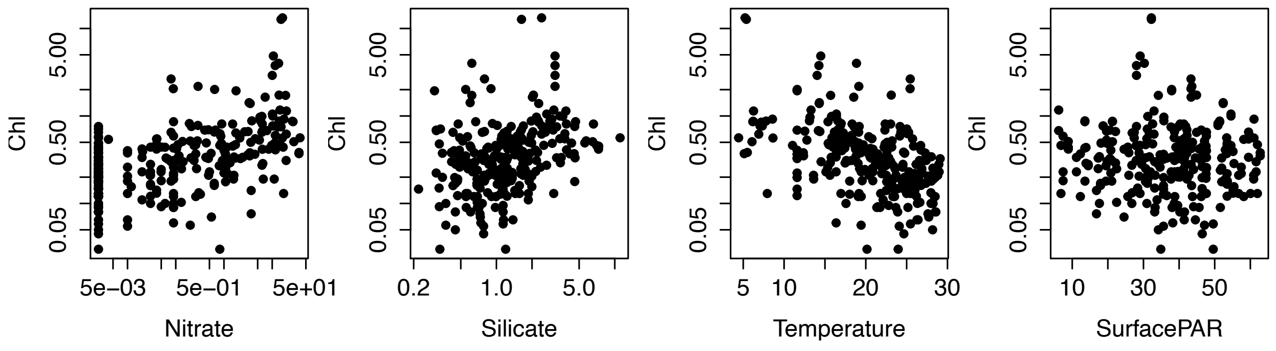


On the y-axis is the model prediction, and on the x-axis is the empirical measurement of NPP using the ^{14}C method. R^2 for prediction vs. reality is 0.87, which is pretty impressive for an ecological model. However, it should be noted that the range of production covers the whole range of oligotrophic to eutrophic conditions in the ocean, over 2 orders of magnitude. So this model will do a good job of predicting variation in production between productive coastal waters and unproductive open-ocean waters, for example. What if we wanted to predict variation in production over time at an oligotrophic site like the open-ocean surrounding Hawaii, where year-to-year variation ranges from about 300 to 600 mg C m^{-2} d^{-1}? In that case the model could perform quite poorly, because the residual variation in the relationship has a standard deviation of about 100 (by eye).

Prediction error, RMSE

R^2 is a relative metric, which ranges from 0 to 1, and quantifies how well a model predicts the data, relative to a perfect model. If we're constructing a model with the ultimate goal of actually making predictions, then R^2 is good to know but we probably want an *absolute* metric of how well the model predicts the data. The most common absolute metric of predictive performance is probably the *root mean squared error*. If a model makes predictions μ_i for data Y_i, then the root mean squared error (RMSE) is $\sqrt{\frac{\sum_i^n (Y_i-\mu_i)^2}{n}} = \sqrt{\text{mean}((Y_i - \mu_i)^2)}$. So RMSE is quantifying how well each data point is predicted in terms of the squared distance between the prediction and the data, or the 'squared error'. This squared distance is averaged across all data points to get a mean prediction error for all the data in the analysis; and the square root of the average is taken so that the result is on the same scale as the data. Clearly a model that does a better job of predicting the data will have a lower RMSE. Whether a particular value of RMSE is considered good or OK or bad will depend on the purpose of the model and the units in which the data is measured.

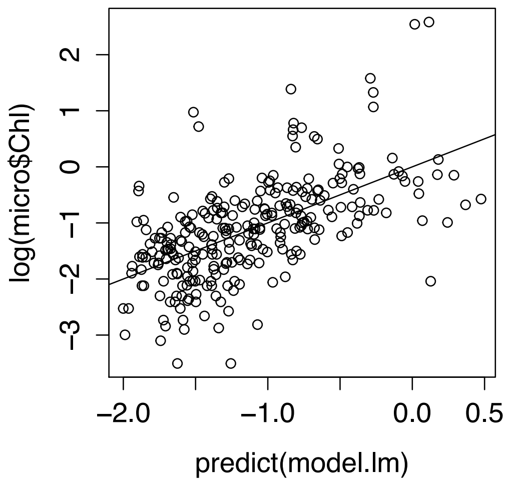
The earlier of primary production in the ocean is a good example of creating a model with the goal of prediction. We can take few measurements of primary production, relative to the size of the ocean. It would be incredibly useful, for scientific and societal applications, if we could predict NPP from remotely sensed variables like chlorophyll and temperature. The models that are used to predict NPP are a little too involved for the purposes of this lecture, so let's try to make a simpler model to predict something similar. I'll use an oceanographic dataset compiled from a number of time series and transects to see how well chlorophyll concentration can be predicted from environmental conditions.



Here I've plotted a bunch of chlorophyll measurements against simultaneous measurements of nitrate, silicate, temperature, and PAR (photosynthetically active radiation) at the surface. Note that I've plotted Chl on a log scale, as well as nitrate and silicate. It looks like collectively these variables could do a decent job of predicting chlorophyll concentration in a statistical model. Let's see how well they do.

```
model.lm = lm(log(Chl) ~ log(Nitrate) + log(Silicate) + Temperature + S
urfacePAR, data = micro)
summary(model.lm)

##
## Call:
## lm(formula = log(Chl) ~ log(Nitrate) + log(Silicate) + Temperature +

##     SurfacePAR, data = micro)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.2507 -0.4557  0.0012  0.3764  2.5256
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.07967    0.20987    0.38  0.70452
## log(Nitrate)    0.07212    0.01927    3.74  0.00022 ***
## log(Silicate)  0.21919    0.06906    3.17  0.00168 **
## Temperature   -0.04772    0.00959   -4.97  1.2e-06 ***
## SurfacePAR    -0.00340    0.00325   -1.05  0.29637
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.731 on 268 degrees of freedom
## Multiple R-squared:  0.351,  Adjusted R-squared:  0.341
## F-statistic: 36.3 on 4 and 268 DF,  p-value: <2e-16
```

It looks like the total R^2 is 0.351, which is not terrible but not that impressive either. Let's compare the observed vs predicted values:



I've added a 1:1 line. The predicted values are extracted with the predict() function. In this case, fitted() would have given the same values. Predict() can also be used to make *new predictions* with a model, i.e. you give the function value(s) of the predictor variables, and predict() will return what the predicted response is. We'll use that shortly.

What is the RMSE corresponding the the above relationship between observed and predicted values?

```
rmse = sqrt(sum((log(micro$Chl) - predict(model.lm))^2)/nrow(micro))
rmse
[1] 0.7245591
```

The RMSE is about 0.72. That means that the typical prediction differs from the observed value by 0.72; because we analyzed Chl with a lognormal model, that means that the typical prediction for Chl differs from the observed value by a factor of exp(0.72), or about 2. So does this model to a good job of predicting Chl in the ocean? It depends on how precise you need to be. If you're OK with predicting Chl within a factor of about 2-4, then it's not that bad, but if you want to get closer then you need a better model.

Overfitting redux

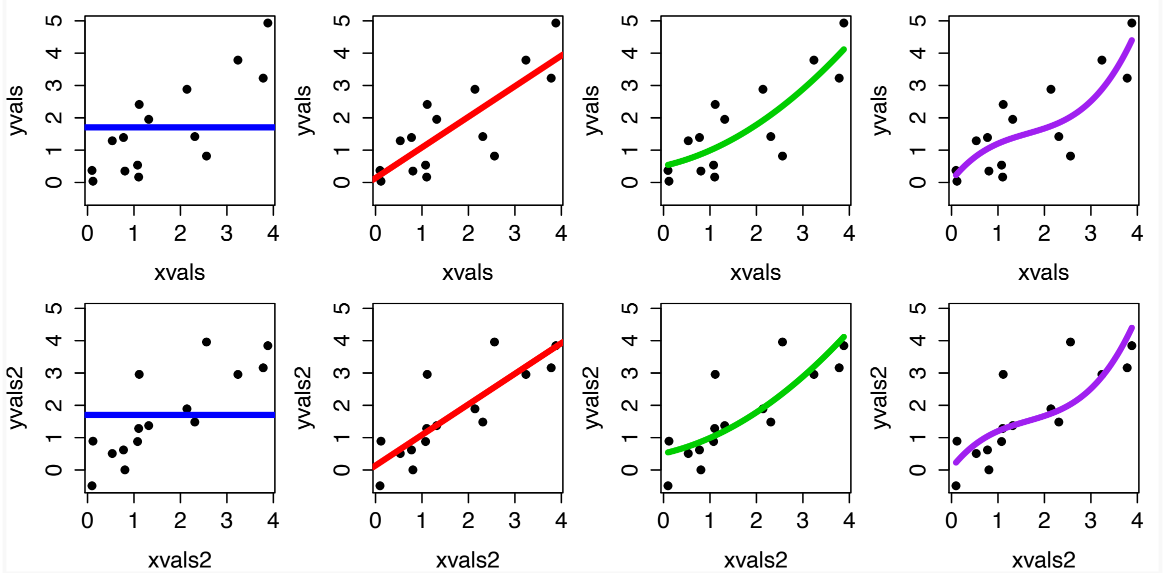
In the chlorophyll example, I used a dataset to fit a model, and then calculated RMSE using that same dataset. This is OK, but when we care about prediction we want to know how well a model is going to predict *new* observations. Is there a reason why prediction error for the modeled data, measured as RMSE, wouldn't be the same as prediction error for new data? Yes, the reason is *overfitting*.

I introduced overfitting a while back, and new we'll dig deeper into how it works and how it affects prediction error and model selection. Let's use a simple simulated example to make it clear. This is similar to the example I showed previously. Here's the process:

1) Simulate data from a linear relationship between *y* and *x*
2) Fit four models to this data: a model with no terms (just an intercept); a linear model; a quadratic model; and a cubic model
3) Calculate RMSE for each model, which quantifies how well the predictions from the model match the data
4) Simulate new data with the same underlying linear relationship between *y* and *x*. This is like taking a new sample from the same 'population'.
5) See how well the four models fit to the first dataset predict the second dataset, using RMSE to calculate prediction error

```r
#overfitting and cross validation
set.seed(1035)

#how many values to simulate
N = 15

#simulate one set of x and y values
xvals = runif(N, 0, 4)
yvals = rnorm(N, sd = 0.8) + xvals*1
fakedata = data.frame(yvals, xvals)

#simulate a second set with the same underyling relationship
xvals2 = xvals
yvals2 = rnorm(N, sd = 0.8) + xvals2*1

#fit models with 0, 1, 2, and 3 polynomial terms
mod0 = glm(yvals ~ 1, data = fakedata)
mod1 = glm(yvals ~ xvals, data = fakedata)
mod2 = glm(yvals ~ xvals + I(xvals^2), data = fakedata)
mod3 = glm(yvals ~ xvals + I(xvals^2) + I(xvals^3), data = fakedata)

#make a function to calculate RMSE
```

```
rmse = function(observed, predicted) {
  sqrt(mean((observed - predicted)^2))
}

#calculate RMSE for simulated dataset 1, vs. the predictions from the models f
it to those data
rmse(yvals, predict(mod0, data.frame(xvals = xvals)))

## [1] 1.42

rmse(yvals, predict(mod1, data.frame(xvals = xvals)))

## [1] 0.8365

rmse(yvals, predict(mod2, data.frame(xvals = xvals)))

## [1] 0.8129

rmse(yvals, predict(mod3, data.frame(xvals = xvals)))

## [1] 0.7789

#calculate RMSE for simulated dataset 2, vs. the predictions from the models f
it to dataset 1
rmse(yvals2, predict(mod0, data.frame(xvals = xvals2)))

## [1] 1.34

rmse(yvals2, predict(mod1, data.frame(xvals = xvals2)))

## [1] 0.7361

rmse(yvals2, predict(mod2, data.frame(xvals = xvals2)))

## [1] 0.7927

rmse(yvals2, predict(mod3, data.frame(xvals = xvals2)))

## [1] 0.8316
```



I omitted the code for plotting the data and curves, but the rest of the code is there. In the plot, the first row plots the same data four times, and plots the four model fits on top of the data. There definitely seems to be a relationship between the variables. We know that the linear model (red) is the true model; the quadratic and cubic curves (green and purple) are fitting the subtle random patterns that are particular to this draw of data. When we compare the four models with RMSE, it looks like the models fit the data better as more polynomial terms are added to the relationship: the RMSE from left to right are 1.42, 0.84, 0.81, and 0.78. So based on these numbers the cubic curve has the lowest RMSE and therefore is the best at predicting the data.

The second row in the plot shows what happens when we use the fitted curves to try and predict *new data generated by the same process*. I.e., I've plotted a second simulated dataset, with the curves fit to the first dataset. None of the curves look terrible in comparison to the data, but whatever random patterns fit by the quadratic and cubic curves in the first row are now gone. How well do the curves from the first row predict the data in the second row? We can calculate RMSE again for each case: 1.34, 0.74, 0.79, and 0.83. It looks like the linear model (red) is the best at predicting new data, because it has the lowest RMSE. Why is the cubic model the best in the first row, but the linear model is the best in the second row? We know that the underlying process generating the data is just a linear model. In the first row the more complex models have *overfit* the data, adjusting to whatever random patterns are there. But those overfit adjustments don't generalize to new data, *and that make the overly complex models worse at predicting new data*. Another way of saying this is that the overly complex models have fit the 'noise' in addition to the 'signal'. It's no coincidence that the model that predicts new data the best is also the 'true' model. We will return to this point when we discuss procedures for model selection.

Cross-validation

The previous example showed how overfitting reduces the ability of a model to predict new data. But this was a simulated example, and in reality we will fit a model with the intention of actually making predictions for data that we don't have, and may never have (e.g. what will happen with primary production in the future, or how much primary production is there in a bunch of places where we haven't measured it). Can we somehow figure out in advance how well a model will predict new data, by accounting for any overfitting that might be happening in the model?

If we had a *very* large dataset, and we wanted to construct a predictive model using that data, we could use an approach like this:

1) Take half of the data, and call this the *training* dataset. Use this dataset to fit a set of models that you think might work.
2) Use the other half of the data to see how well each model actually predicts the data. This reserved data, which was not used to fit the model, is called the *validation* dataset or *testing* dataset. This process will tell you two things: A) The predictive error for each model (e.g. RMSE), with overfitting accounted for because we are not calculating RMSE on the same data used ot fit the models; B) The difference between models in predictive error (useful for model selection, i.e. picking the best model).

For example, companies like Amazon, Target, etc. would like to predict what you're likely to buy. They would like to predict this so that they can make recommendations that will entice you to buy things from them, and so that they can adjust their supply chain to produce correct amount of different products. They have a lot of data (all the data on their customer's buying histories), and they construct complex models where the potential predictors include everything you've purchased in the past, when you bought it, what your age/gender is, etc. Because they have so much data they could use half of it (or ¾, whatever) to fit a set of models, and then use the reserved validation data to test predictive performance.

This kind of approach is great, but often data is scarce and we want to maximize the information that goes into the model(s). The compromise approach is to use *cross-validation*. There are many variants of cross-validation, which are beyond the scope of this class. The method we will focus on is called *leave-one-out cross-validation (LOOCV)*. The algorithm works like this:

1) Remove a single observation from the dataset. Fit the model with the remaining data, and calculate prediction error ($Y_i - \mu_i$) for the removed observation.
2) Repeat this process for each observation in the dataset, and calculate the root mean squared prediction error across all iterations.

In each iteration of this method, the model is trying to predict an observation that wasn't used in the fitting process, and so any effect of overfitting on predictive performance will be accounted for. Therefore, the mean prediction error across all observations will approximate what the prediction error would look like if the model were applied to a truly novel situation. At the same time, we maximize the amount of information used to fit the model.

Another common cross-validation method is called *k-fold cross-validation*. In this method, the dataset is partition into *k* equally sized subsets (e.g. *k* = 5), and in each iteration one of the subsets is withheld as the validation data, the model is fit using the rest of the data, and the prediction error is calculated for the validation data.

The relative performance of leave-one-out vs. k-fold cross-validation depends on sample size and some other subtleties we won't get into.

Leave-one-out cross-validation example

Let's go back to the example of overfitting with the different polynomial curves. We found that the linear model, which happens to be the true model, is also the best at prediction new data (lowest RMSE). Do we get a similar result using LOOCV on the first simulated dataset, i.e. the one with which the models were fit? It is fairly simple make a loop that does LOOCV. For example, for the cubic polynomial the code would look like this:

```r
#vector to save the prediction errors
errors = vector()
#the loop: N for total number of observations
for (i in 1:N) {
  #make a y-vector that removes observation i from the original y-vector
  yuse = yvals[-i]
  #make a x-vector that removes observation i from the original x-vector
  xuse = xvals[-i]
  #make a dataframe with the new y and x vectors
  datause = data.frame(xuse, yuse)
  #fit the cubic polynomial model
  mod = lm(yuse ~ xuse + I(xuse^2) + I(xuse^3), data = datause)
  #calculate the prediction error for the withheld observation
  errors[i] = yvals[i] - predict(mod, newdata = data.frame(xuse = xvals[i]))
}
#calculate the root mean squared prediction error
sqrt(mean(errors^2))
```

```
## [1] 1.061
```

I made a vector to save the prediction error for each observation. Then I loop through *N* iterations, and in each iteration I make a y-vector (the response) and an x-vector (the predictor) by removing observation *i*. This is easily done with the index [-i], which makes a new vector that removes element *i*. Then I make a dataframe with yuse and xuse, and fit the cubic polynomial model to those data. Finally I calculate the prediction error for the one observation I withheld from the model. The prediction from the model can be calculated with the predict() function. The first argument is the fitted model we want to use for prediction, and the second argument is a new dataframe that has the value(s) of the predictor variable for which I want a prediction. I did data.frame(xuse = xvals[i]), because I want a prediction for xvals[i], which is the observation that was withheld, and I need that column in the dataframe to be named 'xuse' because that is the name of the predictor in the fitted model. After the loop is done I calculate the root mean squared error across all observations, and it is 1.061.

In a homework assigment I will ask you to code a similar LOOCV example, so that you can get a sense for how it works. But fortunately we can in general use a function already created in the 'boot' package, called cv.glm(). This function automatically performs LOOCV for a glm model. In this case we are fitting models with gaussian error, which can be done with lm() or glm(), and so we will use glm() because that's what cv.glm() takes. So let's take the four models fit to the simulated data, and calculate their LOOCV error:

```r
#fit models with 0, 1, 2, and 3 polynomial terms
mod0 = glm(yvals ~ 1, data = fakedata)
mod1 = glm(yvals ~ xvals, data = fakedata)
mod2 = glm(yvals ~ xvals + I(xvals^2), data = fakedata)
mod3 = glm(yvals ~ xvals + I(xvals^2) + I(xvals^3), data = fakedata)
```

```r
#calculate the leave-one-out cross-validation error for each model
library(boot)
sqrt(cv.glm(fakedata, mod0)$delta[1])

## [1] 1.522

sqrt(cv.glm(fakedata, mod1)$delta[1])

## [1] 0.9657

sqrt(cv.glm(fakedata, mod2)$delta[1])

## [1] 1.025

sqrt(cv.glm(fakedata, mod3)$delta[1])

## [1] 1.061
```

cv.glm() returns a bunch of info; to get the LOOCV mean squared error you can do `cv.glm(fakedata, mod0)$delta[1]`. This function returns the mean of the squared prediction errors, but I wanted the *root* mean squared error, so I took the square root.

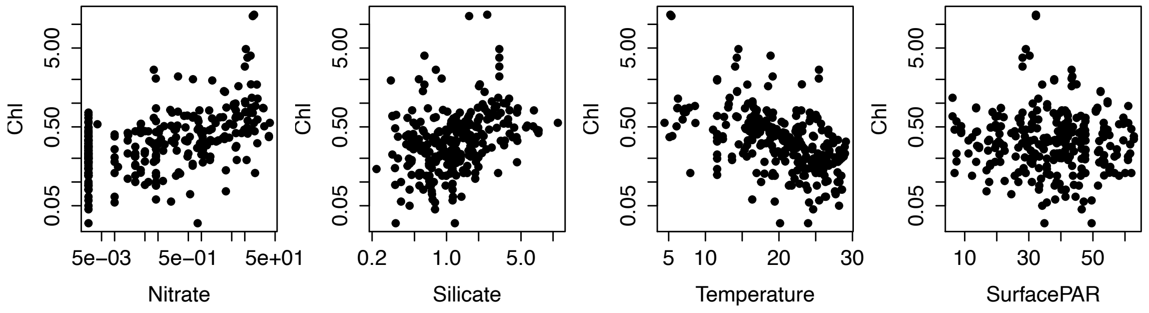
What do the results look like? Let's compare the LOOCV RMSE to the RMSE we got when we compared model predictions to a new simulated dataset:

Model	**LOOCV RMSE**	**New data RMSE**
Intercept only	1.52	1.34
Linear	0.97	0.74
Quadratic	1.03	0.79
Cubic	1.06	0.83

In terms of the relative ranking of the models, the LOOCV results are similar to the results when trying to predict new simulated data. The intercept-only model is the worst, the linear model is the best, and the more complex quadratic and cubic models are somewhat worse than the linear model. The exact numbers for the two columns are a little different, which is not surprising. The 'new data' example just uses one simulated dataset, and so the predictive performance of the models would bounce around a bit if we were to generate additional new data and calculate RMSE. But the magnitude of the prediction error and the relative ranking of the models is similar, so that's reassuring.

LOOCV with some real data

Let's do another example of LOOCV with some real data. We'll use the chlorophyll example again, where the data looks like this:



And the model we fit previously looks like this:

```
model.lm = lm(log(Chl) ~ log(Nitrate) + log(Silicate) + Temperature +
SurfacePAR, data = micro)
```

I want to know how well we can predict chlorophyll using the environmental variables nitrate, silicate, temperature, and surfacePAR. For the fitted model the RMSE is 0.72, but this may have been affected by overfitting. We can make a better estimate prediction error with LOOCV. We can also use LOOCV to ask whether any of the model terms is actually reducing predictive performance. Why would this happen? Just like the polynomial example, if a term is not explaining any 'real' variation in the data, then it will just fit the noise instead of the signal, which tends to reduce predictive performance.
There are a lot of different ways one could compare the performance of the different model terms, and we will explore these in the near future. For now I will compare the following models:

1) The full model, to see how well it predicts the data according to LOOCV
2) A model with just an intercept (no predictors), as a worst case scenario
3) Four more models where each term is dropped, while leaving the others in the model. This will give sense for whether an individual term is just fitting the noise.

```
model.full = glm(log(Chl) ~ log(Nitrate) + log(Silicate) + Temperature
+ SurfacePAR, data = micro)

model.intercept = glm(log(Chl) ~ 1, data = micro)

model.nopar = glm(log(Chl) ~ log(Nitrate) + log(Silicate) +
Temperature, data = micro)
```

```
model.notemp = glm(log(Chl) ~ log(Nitrate) + log(Silicate) +
SurfacePAR, data = micro)
model.nosilicate = glm(log(Chl) ~ log(Nitrate) + Temperature +
SurfacePAR, data = micro)
model.nonitrate = glm(log(Chl) ~ log(Silicate) + Temperature +
SurfacePAR, data = micro)
```

```
sqrt(cv.glm(micro, model.full)$delta[1])
## [1] 0.7395
sqrt(cv.glm(micro, model.intercept)$delta[1])
## [1] 0.9028
sqrt(cv.glm(micro, model.nopar)$delta[1])
## [1] 0.739
sqrt(cv.glm(micro, model.notemp)$delta[1])
## [1] 0.7685
sqrt(cv.glm(micro, model.nosilicate)$delta[1])
## [1] 0.7495
sqrt(cv.glm(micro, model.nonitrate)$delta[1])
## [1] 0.7553
```

The LOOCV RMSE for the full model is 0.74. This is very similar to what we got (0.72) when we calculated RMSE by just doing a standard comparison of the observed data and the model predictions, without any cross-validation. So this suggests overfitting is not a big issue for the model.

The model with just an intercept has prediction error of 0.90. So the full model reduces prediction error by about (0.9 – 0.74) = 0.16. This is better than nothing, though not amazing.

When surfacePAR is removed from the model, prediction error goes down very slightly. This suggest that this term adds no predictive power to the model, and is slightly overfit. But removing it would help that much in either case. For the other model terms, removing them increases the prediction error, which gives a sense for how much predictive power they add to the model, albeit when the other terms are already in the model.

To sum up, a procedure like this is commonly used when the focus of an analysis is making a model that has good predictive performance. Cross-validation allows you to estimate how well a model would predict new data. It also provides one way to select among different models and choose the 'best', in the sense of best predictive performance. We will see that this is essentially the same thing that AIC does when comparing among models.

A note on prediction

The prediction error for a model calculated with LOOCV is approximately equal to what you would see if you actually made new observations (in the future, or from another location) and compared them to the models predictions. However, *this is only true if the new data is generated by the same process*. If the relationship between the predictors and the response changes, or if other unmeasured variables that affect the response change, then all bets are off. This, of course, is why prediction is hard, especially in biology. If global change affects how nutrient limitation works, so that the relationship between nitrate and chlorophyll changes, then the predictive performance of a model fit to current data will be diminished. Likewise if you try to apply a model fit in one region of the world to another region, the underyling processes might change and therefore RMSE calculated with cross-validation will be optimistic.