Resampling

We'll be covering **resampling** in this lecture using aggregated data from Andrew Gelman's speed dating dataset.

Check `speedDatingSimple.csv` for the data and `Speed Dating Data Key.doc` for an explanation of the variables; the data has been aggregated on the level of each person being rated, so each row corresponds to a particular person's average ratings by their partners (on five different scales) as well as their self-ratings of interest in 17 different activities.

In machine learning in general, we have the problem of overfitting, where:

> a problem where a functional form or algorithm performs substantially better on the data used to train it than on new data drawn from the same distribution. It occurs when the parameters used to describe the functional form end up fitting the noise or random fluctuations in the training data rather than the attributes that are common between the training data and test data.

Exercise. Given n data points and a model $a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$ where we fit the coefficients a_i to the data, how large does n need to be before we can come up with a model that goes through every data point precisely? (Think about the linear case, where only a_0 and a_1 are nonzero.)

Our models run the risk of being *overly biased* toward the structure of the data which we have in a non-generalizable fashion. As such, we want to use *resampling* techniques, most of which involve splitting data into *train* and *test* sets. Instead of training the model on the entire dataset, we'll in general train it on a *subset* of the data and estimate the quality of the model against the data which was fed into the model.

Random number generation in R

We'll begin with a brief discussion of the random number generator (RNG) in R.

The RNG can be *seeded* with `set.seed(n)` for any integer `n`. The values that the RNG outputs will depend on its seed, and setting the seed "resets" it to the

initial state corresponding to that particular seed.

- Try using `set.seed()` in conjunction with `runif(5)` to get a sense of how this works.

This is important because we may want, *e.g.*, to generate the same random partitioning of our data consistently, in which case we would put a `set.seed(1)` call before our shuffling of the indices with `sample()`.[1] Or, alternatively, you may want a sequence of reproducibly different calls of `sample()`, etc.

A single test/train split

- Load the speed dating dataset (using `read.csv()`) and restrict to a gender of your choice.
- Write a function that splits the data randomly into a train set and a test set (of equal sizes), trains a linear model on the train set to predict *attractiveness* from the 17 *activities*, and uses `predict(model, newdata)` to generate predictions for the test set.
  - How does the performance on the train set compare to the performance on the test set?
  - Run the function a large number of times to generate predictions for many different train/test splits.
  - For each prediction, use the formula $R^2_{\text{adj}} = 1 - (1 - R^2)\frac{n-1}{n-p-1}$ to calculate the adjusted R^2.[2]
  - Plot the distribution of adjusted R^2 values and calculate their standard error (the standard deviation divided by $sqrt(n)$ for n samples).

n-fold cross validation

In n-fold cross validation, we try to avoid the randomness of a single train/test split. The process goes as follows:

1. We randomly split the data into n different subsets.
2. For each of the n subsets, we train the model against *all the other subsets* and use that model to make predictions on our held-out subset.

[1] In some cases, you may want to (1) save the state of the RNG for later, (2) set the seed to something specific and generate a consistent splitting of the data, and (3) change the RNG back to its saved state. This is possible using `.Random.seed` and is described in Cookbook for R – we won't need this for this lesson, but it's important to be aware of (as it will eventually surely come up).

[2] See StackExchange; this is the *Wherry Formula$.

3. We combine all of the predictions and calculate some measure of model quality, like adjusted R^2 or root-mean-square error.

We'll be continuing with predicting attractiveness from activities. Now, choose one of two different approaches:

- Implement a function that uses n-fold cross validation to generate a set of predictions for any n, or
- Implement 2-fold and 10-fold cross validation separately in two different functions.

Either way,

- Use