In practice, of course, β_0 and β_1 are unknown, and we need to use data to estimate the coefficients. We assume we have a data set with inputs and outputs: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

and we want to find the values of β_0 and β_1 that make our predicted values $Y(x_i) = \hat{y}_i$ as close to the actual values y_i as possible. How do we define "close"? Well, there are a number of ways, but by far the most common is the sum of squares error $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.

The least squares approach chooses β_0 and β_1 to minimize the sum of squares error - in other words it is the *least squares coefficient estimate*, and it's a moderately involved but straightforward multivariable calculus problem to prove that the coefficients that satisfy this requirement are:

$beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$,

$beta_0 = \bar{y} - \beta_1 \bar{x}$.

"Residual sum of squares", and is equal to: $RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.

An issue here is that this number will naturally get larger with more observations, and so we'd like a way to normalize this relative to the number of observations. This is called the "residual squared error" and is given by:

$$RSE = \sqrt{\frac{1}{n-2}RSS} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}.$$

Now, this is a decent measurement of the accuracy of our model, but it still has a problem. It depends on the scaling of the data. For example, if we switched to measuring distances in centimeters and not meters, the actual numeric value calculated here would be 100 times greater, even though the model didn't change. So, a numeric value here doesn't have much meaning unless you're comparing it against other values with the same scaling. This can be overcome with the R^2 statistic.

The R^2 statistic is a proportion, is always between 0 and 1 (for a regression model), and is independent of the scale of your measurements.

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}.$$

Here, TSS is the *total sum of squares* and represents the variance in the response Y independent of X. The R^2 statistic measures what proportion of this variance can be explained by the predictive variable X. If there is no relation (X provides no predictive power), then R^2 will be 0. If X perfectly preficts Y, then R^2 will be equal to 1.

Well, as you might imagine, the world isn't always so simple, and frequently (almost always) there are multiple input factor that affect the output you're wanting to predict. We can take into account multiple factors with *multiple linear regression*, where our model still assumes a linear relationship between the inputs and the outputs, but instead of being a line, it is a plane (if there are two inputs), or a "hyperplane" (if there are more than two inputs).

$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots \beta_n X_n$

How are these coefficients $\beta_0, \beta_1, \ldots, \beta_n$ determined? Well, we can use the same approach as we used with simple linear regression - we can find the choice of coefficients that minimizes the residual sum of squares:

$$\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{m}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_{i2} x_{i2} \cdots - \hat{\beta}_{in} x_{in})^2$$

Here, m is the number of data points, and n is the number of input variables.

The formulas for calculating these coefficients exactly can be fairly involved, and we won't go over them (or even worse, DERIVE them), here, but know that exact formulas are known and can be used by computer systems. If the number of data points or the number of input variables is *huge*, then calculating these formulas exactly can be quite time consuming, and sometimes approximation methods like gradient descent (which we'll cover later) are used.

Now, we're not limited to linear relationships - and in fact neither is the class of "linear models". We could instead have a polynomial model of the form:

$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots \beta_d X^d$,

and find the values of the coefficients that minimize the residual sum of squares. This model will *always* do better than a linear model on your training data, but it runs the risk of overfitting your data - particularly for higher degree polynomials. We'll talk much more about this later.

Note that we would actually *still* call this a linear model? Why, because there are no non-linear interaction terms between the coefficients we're trying to estimate. There are no terms of, for example, the forms β_2^3 or $\beta_1\beta_3$.

However, you'll quickly run into two issues:

- *Model Interpretability* - If you want to try to understand why and how your model is predicting the way it is, the more input variables you have, the more complicated, convoluted, and contrived your explanation will become.
- *Prediction Accuracy* - This is even worse. With more variables, your model may start to overfit the data you use to build your model - doing well on data it's seen before, but poorly on the new data that you really want to predict.

OK, so if RSS and R^2 won't allow us to differentiate among potential models - how should we do it? Well, there are a few statistical measures that do take into account the size of our model and penalize larger models. Some of these are:

Mallow's Cp

Named after Colin Lingwood Mallows it is defined as: $C_p = \frac{1}{n}(RSS + 2d\hat{\sigma}^2)$

where $\hat{\sigma}^2$ is an estimate of the variance of the error associated with each response measurement. Typically ^2 is estimated using the full model containing all predictors. $\hat{\sigma}^2 = \frac{RSS}{n-p-1}$

where p is the number of predictors.

Bayesian Information Criterion (BIC)

BIC is derived from a Bayesian point of view, and looks similar to Mallow's C_p. It is defined (up to irrelevant constants) as:

$BIC = \frac{1}{n}(RSS + \log(n) d\hat{\sigma}^2)$

Like C_p, the BIC will tend to take small values for a model with low test error.

Adjusted R^2

Since the R^2 always increases as more variables are added, the adjusted R^2 accounts for that fact and introduces a penalty. The intuition is that once all the correct variables have been included in the model, additional noise variables will lead to a very small decrase in RSS, but an increase in k and hence will decrease the adjusted R^2. In effect, we pay a price for the inclusion of unnecessary variables in the model. $R_a^2 = 1 - \frac{RSS/(n-k-1)}{TSS/n-1}$

C_p and BIC have rigorous theoretical justification that rely on asymptotic arguments, i.e. when the sample size n grows very large, they become precise, whereas the adjusted R^2, although quite intuitive, is not as well motivated in statistical theory. However... it works.

Based on these measures, how can we then determine an optimal subset of predictor variables to use? Well, generally speaking, there are three approaches:

- Check *all* the possible subsets for the best option.
- Find the best possibility with 1 predictor, and then add on the best possible second predictor based on this first predictor, and so on. This is called forward stepwise selection.
- Find the best possibility with all predictors, and then subtract the least importand predictor based on the model with all predictors, and so on. THis is called backward stepwise selection.

Best Subset Selection To perform best subset selection, we fit separate models for each possible combination of the n predictors and then select the best subset. That is we fit:

- All models that contains exactly one predictor
- All models that contain 2 predictors at the second step: $\binom{n}{2}$
- Until reaching the end point where all n predictors are included in the model

This results in 2^n possibilities. In our case there are $2^9 = 512$ possible combinations

Algorithm

- Let M_0 denote the null model which contains no predictors, this model simply predicts the sample mean of each observation
- For $k = 1, 2, ..., n$
  - Fit all $\binom{n}{k}$ models that contain exactly k predictors
  - Pick the best among these $\binom{n}{k}$ models, and call it M_k. Here the best is having the smallest RSS, or an equivalent measure. mo
- Select the single best model among $M_0, M_1, ..., M_n$ using C_p, BIC, adjusted R^2 or any other method.

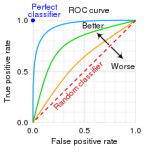
Forward Stepwise Selection

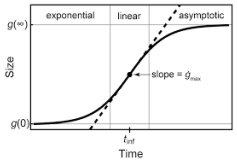
For computational reasons, the best subset cannot be applied for large n due to the 2^n complexity. Forward stepwise selection begins with a model containing no predictors, and then adds predictors to the model, one at the time. At each step, the variable that gives the greatest additional improvement to the fit is added to the model.

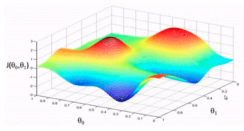
Algorithm

- Let M_0 denote the null model which contains no predictors.
- For $k = 1, 2, ..., n-1$
  - Consider all $n-k$ that augment the predictors in M_k with one additional predictor.
  - Pick the best among these $n-k$ models, and call in M_{k+1}.
- Select the single best model among $M_0, M_1, ..., M_n$ using C_p, BIC, adjusted R^2 or any other method.

SEE LASSO AND RIDGE







Both models penalize large values of the coefficients, and so are biased towards 0. This means the approach will tend to favor underestimating the coefficients.

Why would we want to do this? Well, if we recall from our first lecture, generally speaking the introduction of bias tends to decrease the variance within our model (the bias-variance tradeoff), and so by decreasing the variance, you decrease the chance that your model will be far removed from the actual "correct" model.

We call the penalty for Ridge regression an ℓ_2 norm, and the penalty for the Lasso is ℓ_1 norm. The different numbers refer to the exponent of the individual β_i terms. (Note technically to be an ℓ_2 norm, the penalty for ridge regression should have a square root, but don't worry about that here.)

Now, one of the first thing to notice here is that term λ. What is it, and how do we know its value? Well, the term λ is something called a *hyperparameter*. You don't determine it as part of the modeling process - you define it before you start the modeling process. So, you could view both ridge and lasso regression not as single models, but as families of models parametrized by λ.

At one extreme, $\lambda = 0$, both ridge and lasso regression reduce to standard regression, in which the goal is to minimize $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$. On the other extreme, as $\lambda \to \infty$, the penalty for any positive coefficient goes to ∞, and so the model will tend towards a constant model $Y = \beta_0$, where β_0 will just be the average of the output values.

The validation set approach is conceptually simple and easy to implement. However, it has two potential drawbacks:

1. The validation estimate of the test error rate can be highly variable, and can be highly dependent on precisely which observations are included in the training set vs. the validation set.
2. Only a subset of the observations, those that are included in the training set, are used to fit the model. Since statistical methods tend to perform worse when trained on fewer observations, this suggests that the validation set error rate may *overestimate* the test error rate for the model when fit on the entire data set.

Leave-One-Out Cross-Validation (LOOCV)

For LOOCV, instead of creating two subsets of comparable size as in the validation set approach, a single observation (x_i, y_i) is used for the validation set, and the remaining observations make up the training set. The model is then fit on the $n-1$ training observations, and its predicted value $\hat{y}_i$ for x_i is tested against the actual output y_i. So, $MSE_i = (y_i - \hat{y}_i)^2$. This procedure is repeated for all n observations, and the estimated MSE is the average of the errors for each model:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i.$$

This approach has far less bias than the validation set approach, and removes the randomness and risk inherent in the training / validation set splits. However, it can be computationally expensive.

k-Fold Cross-Validation

An alternative to LOOCV is k-fold cross-validation (CV). In this approach, we randomly divide the set of observations into k groups, or *folds*, of approximately equal size. The first fold is treated as the validation set, and the model is fit on the remaining $k-1$ folds. The mean squared error of the model on this first fold, MSE_1, in then computed. Thes is repeated k times, producing $MSE_1, MSE_2, \ldots, MSE_k$. The MSE estimate is then computed as: $CV_{(k)} = \frac{1}{k}\sum_{i=1}^{n} MSE_i$.

It isn't hard to see that the validation set approach and LOOCV are just special cases of k-fold CV, with $k = 2$ and $k = n$ respectively. Which value of k should you use for your model? Well, that's another hyperparameter.

The problem with using a large value of k isn't just the increase in computation time, which for modern computers is rarely a huge concern. Yet again, the problem with using a large value for k is the bias-variance trade-off. For the extreme case of LOOCV, the training data for each model is almost exactly the same, which means the models are highly correlated. This means that, while a larger value of k will decrease the bias in the individual estimates (the tendency to overestimate the error), it will also increase the variance. There's basically no way of getting out of that tradeoff.

Alright, now we're going to do a bunch of ridge regressions, and record the coefficients we get. Note, there's no concensus as to whether the hyperparameter for ridge or lasso regression is a λ or an α. Most of the teaching literature uses λ, so I've used it here. Unfortunately, Python uses α, which you'll see in the code below.

Some of the nice properties of the sigmoid include:

- Its values are constrained between 0 and 1, so no matter how large (positive) the input to the function is, the output is always less than 1, and no matter how small (negative) the input to the function is, the output is always greater than 0. That's something we want if we're going to treat these outputs as probabilities.
- The sigmoid function is increasing, which means that if something makes an outcome more likely, even more of that something makes the outcome even more likely. This makes intuitive sense, and aligns with what we'd want for a straightforward probability model.
- The sigmoid function is smooth, which mathematically means it's "differentiable", but more qualitatively means it doesn't have any weird jumps or spikes in the probability, which is again what we'd want for a straightforward model.
- The sigmoid function has a region where the probabilities are very low, a region where the probabilities are very high, and a transition region where the probabilities are more in-between, which is what we'd expect intuitively for this type of classification problem.

Accuracy - This one is pretty straightforward. The accuracy of a model is how often it's right, regardless of whether it's right about a true positive, or a true negative. The formula for accuracy is: $\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$

Precision - The precision is concerned with the positive. Specifically, it is the fraction of the time the model's positive result is accurate. Defined precisely, the formula for precision is: $\text{Precision} = \frac{\text{True Positives}}{\text{Predicted Positives}} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$

Recall - The recall is concerned with how often the model correctly identifies an actual positive. Its formula is: $\text{Recall} = \frac{\text{True Positives}}{\text{Real Positives}} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$

Specificity - The specificity is concerned with how often the model correctly identifies an actual negative. Its formula is: $\text{Specificity} = \frac{\text{True Negatives}}{\text{Real Negatives}} = \frac{\text{True Negatives}}{\text{True Negatives + False Positives}}$

The ROC curve (which stands for the receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. The curve plots two parameters, the true positive rate (TPR), which is the recall, on the y-axis, and the false positive rate (FPR), which is 1 minus the specificity, on the x-axis. When the threshold is p = 1, so as strict as it could be, then nothing will be classified as positive, and both the TPR and FPR will be 0. When the threshold is p = 0, so as relaxed as it could be, then everything will be classified as positive, and both the TPR and FPR will be 1. As we move from p = 1 down to p = 0, the TPR and FPR will grow at their own rates, and we can plot their values on a curve. This is the ROC curve.

We measure the performance of the model by looking at the area under the ROC curve. This performance measure is called AUC, which just stands for "Area Under the Curve". This is a number between 0 and 1, and would be 1 for a perfect model (one that gave every real positive probability 1, and every real negative probability 0). For a random model, you'd get an AUC of about .5. A good AUC is usually between .8 and .9.

If we're given a set of input data and we want to find the best predictor, we don't try to minimize the sum of squares error as we did with linear regression. Instead, with logistic regression, we try to *maximize the likelihood*. In other words, we try to find the values of our coefficients such that the probability of our dataset is maximized.

Now, unfortunately, the exact values that achieve this aren't always easy to find. For the sigmoid function and logistic regression, it's not really possible to find a closed-form solution the way we can with linear regression. So, to get around this difficulty, we use *gradient descent*.

Stated more mathematically, at any given point it's frequently straightforward to find the gradient (the vector of partial derivatives) of our function. This gradient tells us the direction of greatest increase at that point, and so if we're looking to maximize a function, we can take a step in that direction. If we're looking to minimize the function, we can take a step in the other direction.

For the sigmoid function, the partial derivative with respect to a coefficient is relatively easy to calculate: $\frac{\partial S}{\partial c_i} = \frac{X_i e^{-(c_1X_1+c_2X_2+\cdots+c_nX_n+b)}}{\left(1+e^{-(c_1X_1+c_2X_2+\cdots+c_nX_n+b)}\right)^2} = X_i\left(1-\frac{1}{1+e^{-(c_1X_1+c_2X_2+\cdots+c_nX_n+b)}}\right)\left(\frac{1}{1+e^{-(c_1X_1+c_2X_2+\cdots+c_nX_n+b)}}\right) = X_i(1-S)S$

Here $c_0 = b$ and $X_0 = 1$.

$$\textit{RIDGE} \quad \sum_{i=1}^{n}(y_i-\hat{y}_i)^2 + \lambda\sum_{j=1}^{p}\beta_j^2.$$

With lasso regression the model attempts to minimize the sum of squares error plus an absolute value term for the coefficients:

$$\textit{LASSO} \quad \sum_{i=1}^{n}(y_i-\hat{y}_i)^2 + \lambda\sum_{j=1}^{p}|\beta_j|.$$

Both models approach a constant model as λ gets large, and approach standard (unbiased) regression as λ gets small. Lasso regression tends to set the coefficients of unimportant variables to 0, while ridge tends to make them small but still non-zero.

Sometimes there are too many input variables and the desire is to create a model with a subset of them, eliminating variables that don't add much predictive value to the model, but might significantly increase the model's variance. To choose a subset of variables there are various (ha!) techniques. One way is best subset selection, in which literally every possible subset it explored. This is exhaustive and optimal when it's possible, but the number of possible subsets increases exponentially with the number of variables, and so this might not always be possible. Forward stepwise selection first finds the best model with 1 variable, the first step, and then finds the best model with 2 variables that includes the variable found on the first step. At each step a new variable is chosen, and at step k one additional variable is added to the $k-1$ found in the previous steps. This method doesn't try every combination of variables, but usually does a good job of finding the optimal subset or getting close to the optimal subset. Backward stepwise selection is the same idea, just starting with every variable, and removing one at each step.

A model will always have a lower RSS than a model built with a subset of its parameters (because there's always the option to set a parameter's coefficient to 0). Consequently, RSS and related measures cannot be used to differentiate among these models. In class we learned a few ways to differentiate among them, the most notable being k-fold cross-validation. k-fold cross-validation is a method for breaking the modeling data into k disjoint segments. One segment is left out, and the model is trained on the $k-1$ remaining segments, and then tested on the one left out. This is repeated leaving each segment out once, so a total of k times. The success of the model is its average performance on the data that is left out.

This approach can be used to determine the optimal value of a *hyperparameter*, which is a parameter that is set before modeling takes place, and not as a part of modeling. The term λ in ridge and lasso regression is a hyperparameter, as is the subset of variables left out in variable selection, and the number of knots to use for a regression spline.

4 Bootstrapping

Bootstrapping is a technique that looks like cheating, but it's not. The basic idea is to start with a set of sample data, and build other samples from this original data using sampling with replacement. So, some of the observations from the original data might show up multiple times in a derived sample, while others won't show up at all. In bootstrapping, these derived samples are used for modeling, and the variation in the modeling parameters are recorded for the various derived samples. This can give us a very good idea of how the modeling parameters vary among samples from the original dataset, and this in turn can give us usually pretty good estimate of how the modeling parameters vary among samples from the actual population.

5 Regression vs. Classification

The fundamental distinction between regression and classification is that with regression the model is attempting to predict a number, while with classification the model is attempting to predict a category. There may or may not (usually not) be any natural notion of a distance between the categories.

A given prediction is either right or wrong, but a more reasonable approach is to predict probabilities instead of specific predictions. The specific prediction is then just the category to which the model assigns the highest probability.

6 Logistic Regression

For binary prediction (yes/no, true/false, 1/0, etc...) one of the more popular methods is *logistic regression*. With logistic regression we start with a linear combination of our input variables as we have in traditional regression:

$$c_1X_1 + c_2X_2 + \cdots + c_nX_n.$$

We then plug this linear combination into the *sigmoid function*, which is a function that takes any real number and predicts an increasing number between 0 and 1 (which can be interpreted as a probability).

$$\frac{1}{1+e^{-c_1X_1+c_2X_2+\cdots+c_nX_n}}$$

To figure out the coefficients for a given set of training data, we don't minimize the sum of square error like we do with regression. Instead, we calculate the maximum likelihood. The maximum likelihood is the value of the coefficients such that, if the sigmoid is a probability, it would make the observed data most likely. For example, if you have a coin that comes up heads with probability p, and you flip it 10 times, and it comes up heads 7 times, the maximum likelihood estimate for p would be 0.7.

In standard regression calculating the values of the coefficients that minimize the residual sum of squares can be done exactly, in logistic regression it's almost never possible to calculate the maximum likelihood estimate exactly, and so it needs to be approximated numerically using *gradient descent*.

7 Gradient Descent

A situation that comes up all the time with machine learning is that you've got a predictive function, and you have some measurement of how good it is (this measurement is usually called a *loss function*), and you want to optimize this predictive function relative to the loss function. In logistic regression, the predictive function is the sigmoid, and the measure you want to optimize is the likelihood. Finding the global optimal value is generally impossible, however it's typically straightforward to calculate the gradient of your measurement function with respect to the parameters of your model. This gradient tells you the direction you should move if, locally, you want to increase your measurement function the most. If you want to decrease your measurement function, you'd move in the direction opposite the gradient. Gradient descent is, essentially, taking a bunch of small steps in the direction of the gradient at each step in search of an optimal value. There are a few things about which you need to be concerned when using this process:

- If your step size is too great, you could step over and miss your optimal value.
- If your step size is too small, it could take you a very, very long time to find your optimal value.
- You could get stuck in a local extrema, which would mean it's the largest (or smallest) value close to where you are, but it's not the globally optimal value.

Also, typically gradient descent requires you to calculate your measurement function over the entire training dataset at each step. Depending on the size of your training data, this can take a long time. One way to get around this is with *stochastic gradient descent*, in which case the gradient is calculated with respect to just a single randomly chosen observation at each step. It's called stochastic gradient descent because the observation is chosen at random. This can significantly speed up the process of gradient descent, but entails some loss of accuracy.

Stochastic Gradient Descent One way to get around this issue is with *stochastic gradient descent*. The idea behind stochastic gradient descent is that (in the extreme) we only see how our prediction works on one single observation, and then we adjust our parameters accordingly based upon our prediction for that observation. In other words, we move either forward or backward depending on whether our prediction was right or wrong.

This can make the optimization problem *much* less computationally intensive, although it does introduce some potential problems, and makes your optimization dependent on the order in which you evaluate the observations. A middle ground between pure gradient descent and extreme stochastic gradient descent is batched stochastic gradient descent, where we divide our data into disjoint groups, and run gradient descent over each group individually.

---

1. (2 points) The parameter λ is not determined when the model is fit, but determined before the model is fit. What is that type of a parameter called?

   A *hyperparameter*

2. (3 points) Is the ridge regression model biased? If so, in what way?

   Yes. The ridge regression model biases the coefficients towards 0.

3. (4 points) Why would you possibly want to use a biased model?

   A biased model can be useful in decreasing variance, which can help if the number of parameters in the model relative to the amount of training data is high enough to potentially cause overfitting issues.

4. (4 points) Explain what k-fold cross-validation is, and how it could be used to determine the value of λ with, for example, $k = 10$.

   k-fold cross-validation is a method for determining the value of a hyperparameter by breaking the modeling data into k disjoint segments. One segment is left out, and the model is trained on the $k-1$ remaining segments, and then tested on the one left out. This is repeated leaving each segment out once, so a total of k times. The success of the model is its average performance on the data that is left out.

   To use this method to determine the value of λ, the performance of the model for different values of λ could be tested using 10-fold cross-validation, and the value of λ that performs the best would be the chosen one.

5. (2 points) What type of model does ridge regression become as λ gets very large? What type of model does ridge regression become as λ gets very small?

   As $\lambda \to \infty$ the ridge regression model approaches a constant model. The ridge regression model approaches standard least-squares regression as $\lambda \to 0$.

Variable Selection and Dimension Reduction (10 points) Suppose I have a multiple linear regression model, and I want to build a simpler model using fewer variables. I want to drop variables in a way that best maintains the predictive power of my model.

1. (4 points) Suppose I decided to pick the best model over all possible subsets of variables by determining which subset gives me the highest R^2 value. Would this be a good idea? If so, why? If not, why not?

   It would not be a good idea. Adding more variables will always increase the R^2 value for the training data, so this approach is guaranteed to pick the subset with the most variables.

2. (3 points) What would be the advantage of using forward stepwise selection to choose my variables as compared to best subset selection?

   The advantage to using forward stepwise selection, or backward stepwise selection, is it doesn't check every possible combination of variables like best subset. The number of possible combinations increases exponentially with the number of variables, so if the number of variables is reasonably large, best subset might be infeasible.

3. (3 points) If I wanted to completely eliminate some of my input variables, would I likely want to use lasso or ridge regression? Why would I prefer one to the other?

   You'd want to use lasso regression. Lasso regression tends to set the coefficients of unimportant variables to 0, while ridge regression tends to make them small but still positive.

Bootstrapping (5 points) If I have a sample of 500 observations, and I use the bootstrapping approach to create another sample of 500 observations from the original sample, why isn't the other sample guaranteed to be exactly the same as the original? Could it be exactly the same as my original?

The sample created almost certainly won't be the same as the original sample because bootstrapping is done by sampling with replacement, and so its likely that some observations will be repeated, while others won't occur at all, within the second sample.

It is possible that the sample created is exactly the same as the original, but it's very, very, very unlikely. The probability of it happening is $500!/500^{500}$, which is an incredibly small number.

Logistic Regression (20 points) Please note that for all these questions by *logistic regression* we mean *binary logistic regression*. With linear regression we use a predictive model of the form:

$$\hat{y} = c_1X_2 + c_2X_2 + \cdots + c_nX_n.$$

With logistic regression we use a predictive model of the form:

$$\hat{y} = \frac{1}{1+e^{-(c_1X_1+c_2X_2+\cdots+c_nX_n)}}.$$

1. (3 points) Why can the logistic regression model be interpreted as a probability while the linear regression model cannot?

   Because the logistic regression model returns a value between 0 and 1, while the linear regression model has no such constraints.

2. (4 points) With linear regression we try to minimize the residual sum of squares (RSS). What do we try to optimize with logistic regression?

   With logistic regression we try to find the maximum likelihood, which is the value of the coefficients that makes the probability of the observations the greatest. In practice, we usually try to minimize the negative of the logarithm of the likelihood a.k.a. the log-likelihood.

3. (4 points) If I flip a coin 10 times and 4 of those flips come up heads, derive the maximum likelihood estimate for p, the probability that the coin comes up heads on a given flip.

   The probability of 4 heads is $p^4(1-p)^6$. To find where this is maximized, we can take the derivative and set it to 0:

   $$4p^3(1-p)^6 - 6p^4(1-p)^5 = 0 \Rightarrow 4(1-p) = 6p \Rightarrow 4 = 10p \Rightarrow \frac{4}{10} = p.$$

   So, the probability that maximizes the likelihood of this result is $p = .4$.

4. (4 points) With logistic regression, unlike linear regression, there's no closed-form solution to optimize the model coefficients. Instead, what method is typically used to search for optimal coefficients?

   Gradient descent is typically used to attempt to find the optimal value of the model coefficients.

5. (5 points) Describe stochastic gradient descent, and how it differs from standard gradient descent.

   With standard gradient descent, we look at *all* of our data, and move in the direction that minimizes the error over all of it. With stochastic gradient descent, we look at the data one observation at a time, and each step we move in the direction that minimizes the error for the current observation.
   Stochastic gradient descent requires significantly less computation time than standard gradient descent, but it can introduce bias depending on the order in which the observations are processed.