Statistical Learning

What Is Statistical Learning?

We've previously looked at linear regression as a method of using sample data to estimate a model for a relationship between an input variable and an output variable. This is an example of a broader concept of using sample data to estimate some sort of model to describe one or more variables. This process is known as **statistical learning**, or **machine learning**, or **statistical modeling**.

Supervised and Unsupervised Learning

In regression, we are specifically looking at a model that relates one or more **input** variables to an **output** variable. These input variables might also be referred to as **predictors**, or **independent variables**, or **features**, or sometimes simply **variables**. The output variable might also be referred to as a **response** or a **dependent variable**.

We will study a number of additional methods beyond linear regression which similarly create a model that relates one or more input variables to an output variable. We refer to these methodologies collectively as **supervised learning**, because, when fitting the model, we can supervise how well the model is achieving its objective of predicting the output variable, and use that information to find an optimal model.

In general, if we have some output variable Y, and some set of input variables X_1, X_2, ..., X_p (and we will then use X to describe the total data set of all of the X_i input variables), then we are trying to estimate some function

$$Y = f(X) + \epsilon$$

In a linear regression model, f(X) is the regression equation, and ϵ is the residuals. More generally, f(X) is the systematic information that X provides about Y, and ϵ is the random error.

In other situations, we may not have an output variable to try to predict. We may instead just be interested in describing the structure and patterns amongst a group of variables. Methods that do not model an output variable, but instead describe the structure and patterns amongst one or more variables are known as **unsupervised learning**. For example, a company may have data on customers, and want to identify whether there are subsets of customers who are similar to one another. The most common type of unsupervised learning is **cluster analysis** or **clustering**, where data is divided into a set of clusters based on which data points share common characteristics.

Most of the focus in DATA 5321 and 5322 will be on supervised learning methods. In most areas of application currently, supervised methods are much more commonly used. DATA 5322 will include a section on unsupervised learning methods as well.

Prediction vs. Inference

In supervised learning, there are two different goals that we might be interested in: prediction or inference. Different methods will work better for one or the other, so understanding the relative importance of these two goals in a given problem is essential in choosing the best approach to use.

The goal of prediction refers to the quality of the predicted values our model can create for Y. For example, I have a colleague who used to work for a cell phone company, developing models for their help center to predict the length of time that a customer would spend on a technical support call. If the company is focused on properly allocating staff for the help center, then they are interested in making as accurate of predictions for support call time as possible.

When assessing prediction, we focus on the prediction errors. We do not know the true best model for the population, but we can estimate the best model based on sample data. We will refer to this estimated model as $\hat{f}\,(X)$. Our predicted value would be $\hat{Y} = \hat{f}\,(X)$. Our prediction error would be the difference between what is observed and what is predicted, $Y - \hat{Y}$.

That prediction error will have two components. First we consider how far off our estimated model is from the true best model for the population. That is, how far off is $\hat{f}\,(X)$ from $f\,(X)$? In theory, this component of the error could be reduced improving our model, either by obtaining additional data, or by using a model that better fits our data (maybe we should be fitting a quadratic model rather than a linear model, for example). We refer to this as the **reducible error**.

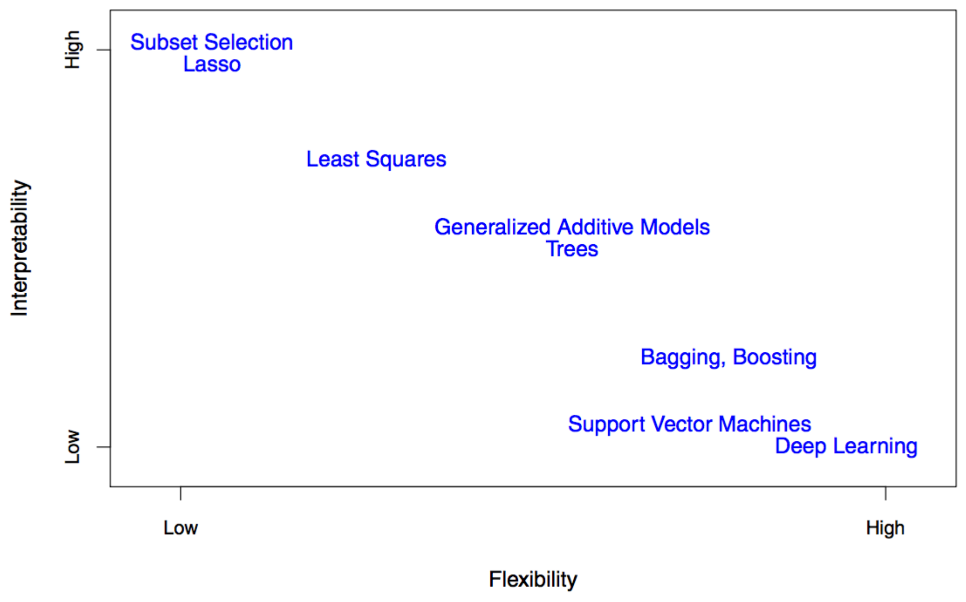
There is also a component of the prediction error that is outside of our control. This is the variability in Y that even the true best model would not be able to account for, ϵ. This could be a combination of uncertainty due to additional variables that we do not have access to, and uncertainty due to inherent randomness in Y. We refer to this as **irreducible error**.

When our goal is prediction, then our objective is to minimize the reducible error. We cannot reduce the irreducible error, but we can quantify it to describe our uncertainty in our predictions.

On the other hand, suppose the company wanted to understand what factors influence call time lengths, so that they could try to develop new procedures to try to reduce call time lengths. Now they are no longer primarily interested in the quality of the predictions from their model, but rather in the information their model can give them about how their inputs relate to their outputs. Their goal is now inference rather than prediction.

In these situations, our objective is to maximize the interpretability of our model. We want a model that lets us say things like "If customers are calling about such-and-such problem, then their call times are cut in half if they are directed to representatives who have received training in topic W."

When selecting what type of model to use for a given problem, there is a trade-off between these two goals. In general, the more flexible a model is (in terms of the variety of forms that $f(X)$ could take on), the better it will be at prediction, but the harder it will be to interpret. On the other hand, the more restrictive a model is, prediction will be poorer, but interpretability will be greater. The figure below shows a number of common machine learning techniques, arranged in terms of their model flexibility and their interpretability:



Estimating f

We can broadly categorize the various methods of estimating a model into two types: **parametric** and **nonparametric**. We've seen these terms previously when discussing methods for constructing confidence intervals or hypothesis tests. In the context of statistical learning, a parametric model is one in which we assume a model (such as a straight line) for which we then need to estimate parameters (such as a slope and intercept). Nonparametric models do not assume a particular functional form for $f(X)$, instead tending to create some sort of smoothed function around the data (a very simple example of this would be a moving average, where, for a given value of x, you would predict the average of the Y values for a set of points whose X values were near the x value of interest).

We will see examples of both parametric and nonparametric models in this course. In general, with parametric models, you need to be careful that you have picked a model that is appropriate for the data (such as, with regression, making sure that there isn't any non-linear pattern in the data). With nonparametric models, this is not a concern. But nonparametric models, in avoiding assumptions about the overall form for $f(X)$, end up needing much larger amounts of data to develop a good model.

Regression vs. Classification Problems

Within supervised learning, we can divide our methods into two categories, depending on whether we are constructing a model with a numerical output variable or a categorical output variable.

Models with numerical output variables are often broadly referred to as **regression** models, and models with categorical output variables are often referred to as **classification** models.

Some types of models can be used for both numerical and categorical outputs, but many are specific to one situation or the other.

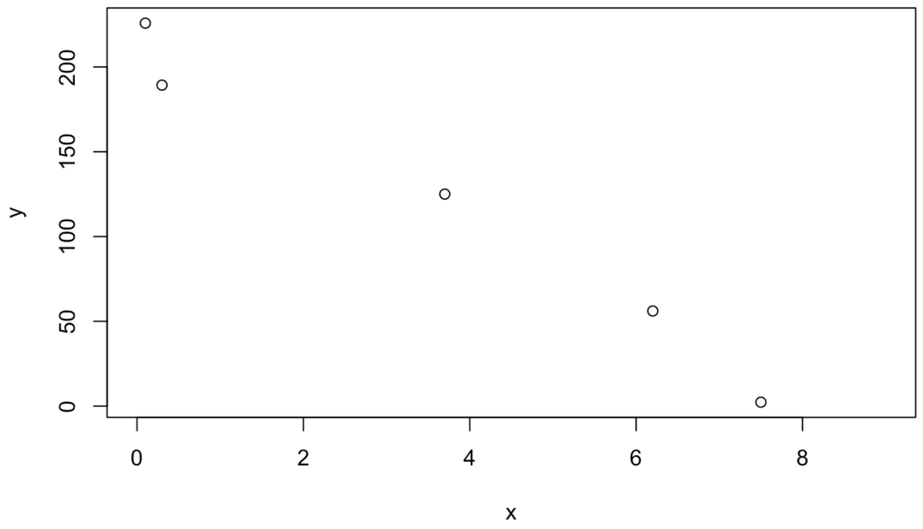
Measuring Quality of Fit

When we are modeling a numerical output, one common way to measure how well a model fits our data is to calculate the **mean squared error (MSE)**. This is the average of the squared differences between observed and predicted y values:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{f}\,(x_i) \right)^2$$

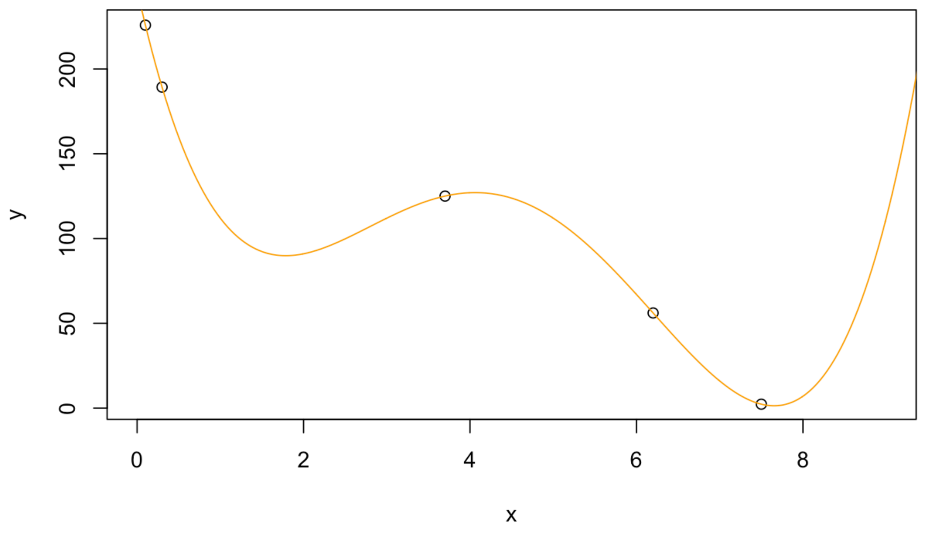
However, we want to be careful about what data we use to calculate this.

Consider the following set of points:



Suppose we fit a 4th order polynomial to model this data. We could find that this data is perfectly modeled by the equation

$$\hat{y} = x^4 - 18x^3 + 104x^2 - 222x + 247$$



For every x_i, y_i is exactly equal to $\hat{y}_i$. Thus, the MSE will be equal to 0. This seems like a perfect model, then - it has absolutely no prediction error.

But do we believe that this is the best model for predicting Y based on X? Look at the original set of data again. What sort of Y value would you expect when X = 2? What about when X = 9? Does the model we have constructed look like it would give us good predictions at those values?

This is an example of what we call **overfitting** - a model that is fit too closely to a particular set of data, such that it will probably not generalize well to new data.

To avoid this sort of issue, rather than assessing the model fit based on the original data we used to fit the model, we can assess the model fit based on new data.

When we do this, we refer to the data used to estimate the model as **training data** or the **training set**. The data we use to assess the model is called the **test data** or **test set**.

In general, the more flexible a model is, the smaller the MSE for the training set will be. But if a model is overfitting the data, it will result in a larger MSE for the test set. By selecting a model that gives a small test MSE, we can select a model that has the appropriate level of flexibility to accurately represent the relationship between our inputs and our output without being too flexible and overfitting.

We sometimes talk about assessing a model in terms of **bias** and **variance**.

In this context, bias refers to how far off the form of our fitted model is from the truth. We can think of it in terms of, if we had an infinite amount of data, how close would our estimated model be to the truth? Fitting a linear model to non-linear data, for example, would introduce bias because no matter how many data points we have, we would obtain a fitted model that accurately described the real relationship.

Variance, on the other hand, refers to how much a fitted model will change if we give it different sample data. How sensitive is the fitted model to the specific data used to train it? In our 4th order polynomial model, if we were to take a new sample of five data points, we would likely end up with a very different curve being fit. This model has high variance.

The more flexible a model is, the lower its bias will be, and the higher its variance will be. The simpler a model is, the higher its bias will be, and the lower its variance will be.

Both bias and variance contribute to the test MSE. Too flexible of a model can result in a high test MSE because of too much variance. Too simple of a model can result in a high test MSE because of too much bias.

In any given problem, there will be some optimal trade off between flexibility and simplicity that gives the best model performance in terms of the test MSE.

In chapter 5, we will examine a number of ways for assessing models based on this principle.

Assessing Classification Models

When assessing classification models rather than regression models, we cannot look at the numerical difference between each y_i and its corresponding $\hat{y}_i$, since these are no longer numerical measurements, but instead are categorical.

The most common measure for classification models is the **error rate**, the fraction of the data points for which the predicted category is incorrect:

$$\frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \hat{y}_i\right)$$

Here, I() is what is known as an indicator function, a function which is equal to 1 if the given statement is true, and 0 if it is false.

As with the MSE for regression models, when assessing the error rate for a classification model, we make a distinction between the training error rate and the test error rate, and we want a model that minimizes the test error rate.

Bayes Classifier

Suppose we know the conditional probability distribution of Y based on X. As a simple example, suppose we have three bowls. Let X = 1, 2, or 3 depending on which bowl we are referring to. And suppose that, in each bowl, there are 100 marbles. Let Y be the color of a marble selected at random from one of the bowls. Say that in bowl 1, there are 50 red marble, 30 green marbles, and 20 blue marbles. In bowl 2, there are 35 red marbles, 25 green marbles, and 40 blue marbles. And in bowl 3, there are 10 red marbles, 10 green marbles, and 80 blue marbles.

In this case, P(Y = red | X = 1) = 0.50. P(Y = green | X = 1) = .30. And P(Y = blue | X = 1) = .20.

We could similarly describe the conditional probability distribution for Y for X = 2 and for X = 3.

Now suppose we want to develop a classification model to predict Y based on X.

If X = 3, what would we predict for Y?

If X = 3, there is a 10% chance Y will be red, a 10% chance Y will be green, and an 80% chance Y will be blue. Of the three possibilities, blue is most likely, so that would be our best prediction.

In general, if we know the conditional probability distribution for Y given X, we could make predictions for each X value by choosing the Y value that has the highest conditional probability. This is known as the **Bayes classifier**.

K-Nearest Neighbors

Most of the time, we do not know the true conditional distribution of Y given X. The Bayes classifier represents an unknowable best possible classification model.

Instead, we often come up with a method to estimate a conditional distribution of Y given X. In DATA 5300 you saw one example of this: logistic regression.

Another common method is the **K-nearest neighbors (KNN) classifier**.

Using this approach, we estimate the conditional probability distribution for Y given X by looking at the K values in our data set which have x values closest to our X of interest, and then finding the fraction of each type of Y value for those K data points.

For example, suppose Y is a binary variable indicating whether or not a flight arrived late, and X is a numerical variable measuring the flight's departure delay. Suppose we want to use a KNN classifier with K = 100 to predict whether a flight will arrive late if its departure delay was 15 minutes. We could go to our flight data set and find the 100 flights whose departure delays were closest to 15 minutes. Then we look at whether each of those flights arrived late. Suppose we find that, out of those 100 flights, 83 arrived late, and 17 did not. That means our estimate is that P(Y = late | X = 15) = 0.83, and P(Y = not late | X = 15) = 0.17.

Then, using the same principle as the Bayes classifier, we could say that "late" is the most likely outcome for Y when X = 15, and so our prediction would be that a flight whose departure delay was 15 minutes would arrive late.

The bias - variance trade-off arises here as well. Smaller values of K will result in more flexible models, with high variance and low bias. Larger values of K will result in less flexible models, with low variance and high bias.

R Commands

At the end of chapter 2, our textbook introduces a number of common R commands. You have already learned many of these previously, but we will highlight some new R commands here.

We can create a matrix in R as follows:

```
x <- matrix(data = c(1, 2, 3, 4), nrow = 2, ncol = 2)
```

You have previously learned to graph two numerical variables. Sometimes we might be interested in graphing three numerical variables. One approach is to construct a contour plot. As an example, suppose we wanted to graph the function $\frac{cos(y)}{1+x^2}$ for values of x ranging from -π to π, and values of y ranging from -π to π.

The code below first uses the seq() function to create 50 evenly spaced x values ranging from -π to π. Then it copies that set list of values to y.

It then uses the function outer(), which tells R to take every possible pairing of x and y, and calculate the value of the specified function. This will create a 50x50 matrix of value for f.

Finally, the function contour() is used to generate a contour plot.

```
x <- seq(-pi, pi, length = 50)
y <- x
f <- outer(x, y, function(x, y) cos(y) / (1 + x^2))
contour(x, y, f)
```

We've seen some examples of how to index data in R before. When we have a matrix of values, we can give indices for both rows and columns.

Let's first create a 4x4 matrix A, with values of 1 through 16. Then we will display the entire matrix, and then the entry in the 2nd row and 3rd column:

```
A <- matrix(1:16, 4, 4)
A
A[2, 3]
```

In general, any indexing before the comma refers to the rows, and any indexing after the comma refers to columns. Suppose we wanted the first two rows, and all but the final column:

A[1:2, -4]

When looking at several numerical variables at once, we might be interested in creating a set of scatterplots for each pair of variables. We could construct these each individually, but R also has a function that can do this for us, pairs().

We will first load the package ISLR2, which contains data sets from our textbook. Then we will create scatterplots for one of the data sets, Auto.

```
library(ISLR2)
pairs(Auto)
```