Stat 442 Lecture Notes

Spring Semester, 2025

Last Updated: February 4, 2025

1 Wednesday 2/5: Welcome, what is statistical learning, basics of supervised learning

1.1 What is statistical learning?

Classical statistics is built on the scientific method. It assumes that we start with a hypothesis, and then we go out and we collect data that can be used to test this hypothesis. Importantly, the statistical methods that we will use to test this hypothesis are pre-specified; we choose them before we ever look at our data.

For example, maybe we come up with the hypothesis that college students who sleep more than 7 hours per night have higher average GPAs than those who sleep less than 7 hours per night. We then go out and take a random sample of students, and ask them all if they sleep more than 7 hours per night, and also ask for their GPA. We then use a t-test to test if there is a statistically significant different in GPA between students who sleep more or less than 7 hours per night.

Statistical learning is different. While there is no single, satisfying definition of statistical learning, it refers generally to a collection of tools used to make sense of complex data. Importantly, we don't always start with a pre-specified hypothesis. In this "big data" era that we are living in, statistical learning has become very important.

In the context of our motivating example, suppose that we collect a large survey of students at a college. We are generally interested in what factors impact the GPA of a college student. We search through the data, try different models, and end up realizing that "hours of sleep per night" seems to be an important factor in determining GPA: more important than, say, major, whether or not you are a varsity athlete, etc. However, we also realize that the relationship between hours of sleep per night and GPA does not look linear. We decide to turn "hours of sleep per night" into a binary variable: yes/no do you sleep more than 7 hours per night. We have ended up with the same "model" as before, but this was a much more exploratory process. It started with a general task of *prediction* (as opposed to a specific model or hypothesis), and used *variable selection* and *feature engineering* to settle on a final model. I would consider this to be an example of statistical learning.

In recent decades, as datasets have become larger and computers have become more powerful, there have been a proliferation of complex methods for analyzing data that are often labeled as machine learning methods. In general, however, there is no clear dividing line between statistics and machine learning. Linear regression and logistic regression, which you all studied in detail in Stat 346, are certainly examples of machine learning methods: they help us learn from data. In fact, they are both methods for *supervised learning*, which is one of two primary types of statistical learning.[1]

- In *supervised learning*, we start with a response variable of interest and a set of potential predictor variables (also known as explanatory variables). Our general goal is to use the predictor variables to explain or predict the response variable.
- In *unsupervised learning*, we just start with a collection of variables: there is no specific response variable. Our goal is to find structure or patterns in the data.

In this course, we will spend around 9 weeks on supervised learning, and only 2 weeks at the end on unsupervised learning. However, I do not want to give you the impression that unsupervised learning is less important! Unsupervised learning is extremely important, and is in fact the back-bone of recent generative AI technology. We start with supervised learning because it has a more natural connection to classical statistics and builds more naturally on your previous work. Hopefully, some of the concepts that you learn during our supervised learning unit will help you be equipped to delve more deeply into unsupervised learning in the future if you choose.

There is one more type of learning that has become really important in recent years, partly due to the proliferation of generative AI. This is the field of *semi-supervised learning*, in which we have a (potentially small) set of *labeled data* (data with predictors + response and a (potentially much larger) set of *unlabeled data* (data that is missing the response). While we will not cover this in lecture, it would make a great *final project topic*. Throughout the semester, I will try to flag these topics as they come up!

[1] I would be remiss if I did not mention *reinforcement learning*, which is a very important topic in machine learning but does not relate as directly to drawing insights from a given dataset.

1.2 Supervised learning

1.2.1 Introduction

As mentioned, you all already know at least two methods for supervised learning: linear regression and logistic regression. In general, these represent one from each of two classes of supervised learning methods.

- Regression: we use this term for any method that is used to predict a quantitative response variable.
- Classification: we use this term for any method that is used to predict a categorical response variable.

As a reminder, a quantitative variable can be either continuous or discrete, but it must be a number. A categorical variable can be either ordinal (the categories have orders) or unordered. In this class, we will not cover any methods for considering ordinal response variables: if this topic interests you, it is another *potential final project topic*.

Linear regression and logistic regression are basic examples of supervised learning methods. In the past few decades, much more complex algorithms have been developed, and have exploded in popularity. There are a few reasons for this proliferation in complex methods.

- Datasets have gotten bigger. As we will learn in this class, complex models are often only appropriate when applied to enough data. Thus, it is natural that the "big data" era has gone hand-in-hand with the development of complex statistical learning methods.
- Computers have gotten better, faster, and cheaper, which has enabled the fitting of complex models.
- Free, open-source software programs like R and python have allowed researchers to develop algorithms and then easily share them with non-experts. This has allowed many methods to become popular.

Given the huge number of statistical learning methods, we could spend each class this semester learning a new method, such that at the end you are left with a huge cookbook of methods to choose from for your future data analysis needs. In fact, this class will be a little bit like this. But, there is a problem with this approach of treating statistical learning like a cookbook. New machine learning methods get published every day, and the state-of-the-art is constantly changing. If your goal is to learn a list of methods or algorithms, you will find yourself continually behind and out-of-date! Thus, the goal of this class is not really to teach you a list of methods. Instead, the goal is to teach you the fundamental concepts and overarching themes that go into the development of the methods, such that you can compare methods, recognize their limitations, and learn new methods on your own in the future. This goal helps explain two features of this course:

- We will spend a fair amount of time on older, less state-of-the-art methods. We will use these methods to illustrate general principles and themes, even if they may not be the methods that you will use in practice in your future.
- You will all be doing a *teaching presentation* this semester. You will be responsible for doing independent reading to learn about a method or a concept. You will need to synthesize what you learn, and give a 20 minute presentation to the class explaining the method. This skill mirrors the way you will all interact with machine learning in the future: you will be continually learning and synthesizing new methods.

1.2.2 Notation for datasets and random variables

The backbone of statistical learning is a matrix of predictors $\mathbf{X} \in \mathbb{R}^{n \times p}$ and a vector of length n that stores the responses $\mathbf{y}$. We let n denote the number of *observations* (rows of our dataset) and let p be the number of *variables* (columns of our dataset, usually not including the response): this notation is quite standard in statistics, but isn't always particularly precise.

Your textbook (ISL) denotes pieces of the dataset $\mathbf{X}$ and $\mathbf{y}$ as follows. We observe x_i for $i = 1, \ldots, n$, where $x_i = (x_{i1}, \ldots, x_{ip})^\top$ is a p-vector for each observation. We use $\mathbf{x}_1, \ldots, \mathbf{x}_p$ to denote the columns of $\mathbf{X}$; each is an n-vector representing its own variable. Your textbook says that it will use bold lowercase letters anytime the vector has length n and use unbold lowercase letters otherwise: I am sure that I will start messing this up soon, but I will try to be consistent.

The other important distinction is between a random variable and its observed realization. We will use capital, unbold letters to denote random variables. In our example from Section 1.1, we might let X_1 denote the number of hours that a randomly selected student sleeps per night and let Y denote their GPA. Once we observe a particular n students for our dataset, we let their sleep values be $\mathbf{x}_1 = (x_{11}, x_{12}, \ldots, x_{1n})^\top$ and we let their GPA values be $\mathbf{y} = (y_1, \ldots, y_n)^\top$.

Consider a dataset that contains information on n pets. The response variable is the weight of the pet $\mathbf{y}$, and we have one single predictor variable: the type of pet. This is a categorical variable that takes on values {cat, dog, hamster, rabbit}. I could represent $\mathbf{X}$ in this case as a matrix with one column; where the column stores the actual values {dog, dog,

cat, dog, hamster, cat, dog, ...}. But, as you all know from Stat 346, it is likely going to be convenient for fitting to instead let $\mathbf{X}$ be a matrix with four columns, that store binary variables indicating "is this a dog", "is this a cat", etc. This simple example shows us how notation can get complicated: do we say that $p = 4$ or $p = 1$ for this dataset? Practically speaking, the $p = 4$ is likely more relevant! But it can depend!

We can also get into problems when counting n sometimes. Suppose we take measurements on 920 students, who are spread out between 8 schools. The schools are randomly assigned to either have required yoga PE class, or to have traditional PE class. While we have 920 students, the students within a given school are not *independent* of one another: there may be factors other than the yoga class that are making their test scores more similar to one another. On the other hand, the schools underwent random assignment and can be safely assumed to be independent of one another. Thus, is the sample size n the 920 students or the 8 schools? It turns out that for this *clustered* or *hierarchical* data, we have a notion of *effective sample size* that is somewhere between 920 and 8, and depends on how correlated the students are within the schools.

We should also mention that data does not always come to you in a form where it looks like $\mathbf{X} \in \mathbb{R}^{n \times p}$ and $\mathbf{y} \in \mathbb{R}^n$. Consider the task of image classification. You get a set of 500 images, which are stored as 400×400 pixel images, where each pixel records a numerical value between 0 and 255 for red, green, and blue. Each image is associated with a label in $\{\text{cat}, \text{dog}\}$, and you observe $y_1, \ldots, y_{500}$. In this case, the RGB values in the pixels in the images are your predictor values, but they don't come in a simple $n \times p$ matrix. We need to reshape the data so that each image i has an associated vector $x_i \in \mathbb{R}^{400 \times 400 \times 3}$ that stores all three color values for all 400×400 pixels. It will also likely be convenient to code y_i such that a 1 denotes a dog and a 0 denotes a cat, such that y becomes numerical. When there are more than 2 classes, such as $\{\text{cat}, \text{dog}, \text{fish}\}$, we will need more than one column to store our response variables numerically.

Who knew that counting n and p could be so complicated! In this class, just think of it as the number of rows and columns (minus the response columns) of the data, and remember these subtleties in case they ever come up.

In our supervised learning unit, we will start with *regression*. Thus, for the rest of today, we assume that Y is numerical.

1.3 Regression

In a regression problem, we assume that our response variable Y is related to a set of predictors $X = (X_1, \ldots, X_p)$ via some model that can be written as

$$Y = f(X) + \epsilon. \tag{1}$$

The function $f()$ is a fixed but unknown function that represents the systematic information that X provides about Y, and ϵ is a random error term, which should satisfy the condition that $E[\epsilon] = 0$ and that ϵ is independent of X. These conditions ensure that $f()$ is capturing all parts of Y that can be explained by X, and ϵ captures the parts of Y that cannot be explained by X. The task in regression is to come up with a good approximation for the unknown function $f()$. More specifically, we want to find a function $\hat{f}()$ such that, $Y \approx \hat{f}(X)$.

1.3.1 Two possible goals

There are two reasons why we might want to come up with a function $\hat{f}()$ such that, $Y \approx \hat{f}(X)$.

1. We might want to be able to come up with predictions $\hat{Y} = \hat{f}(X)$ for future realizations of data where Y itself is not observed.
2. We might want to understand which predictors in X are most associated with Y in order to draw a scientific conclusion, or at least take a step towards drawing a scientific conclusion.

When beginning a regression problem, it is important to clearly define which of these two reasons is more important to you. This will aid you in choosing the best strategy for approaching the problem!

As we will see throughout the semester, these two goals can sometimes be at odds with one another! If we only care about prediction, we do not need our function $\hat{f}()$ to be interpretable: we can use an extremely complex model (known as a "black box"), and we will be happy as long as $\hat{Y} \approx Y$. On the other hand, if we want to draw scientific conclusions, then we might wish to use a simple, interpretable model for $\hat{f}()$ that lends itself to rigorous inference. We should also note that it is not always an either-or situation. Sometimes, we want to make good predictions and also understand which variables are contributing significantly to the predictions. Managing the tradeoff, or lack thereof, between these two goals will be one of our fundamental themes for the semester.

1.3.2 Measuring accuracy

Let's assume for a few minutes that our goal is to make good predictions. In this case, we will be happy with any model that has low prediction error, meaning that $Y \approx \hat{Y}$.

Formally, let's measure error using squared error loss. You are familiar with this from least squares regression, and you know that it has some nice properties but also some downsides. Specifically, we would like to know about the *expected prediction error* of our model, which we can write as:

$$E[(Y - \hat{Y})^2] = E\left[\left(f(X) + \epsilon - \hat{f}(X)\right)^2\right]. \tag{2}$$

We can see from (2) that, even if we manage to estimate f perfectly, our expected prediction error is not 0, because we always have the contribution of our random error ϵ. As statisticians, we always think that there is random noise ϵ; we do not expect to ever make perfect predictions![2]

It turns out that, under the assumptions stated below (1), we can rewrite (2) as:

$$E\left[\left(f(X) + \epsilon - \hat{f}(X)\right)^2\right] = E\left[\left(f(X) - \hat{f}(X)\right)^2\right] + \text{Var}\left(\epsilon\right). \tag{3}$$

We call the term $E\left[\left(f(X) - \hat{f}(X)\right)^2\right]$ the *reducible error*: we can make it smaller by using a better statistical learning tool to estimate $f()$. We call $\text{Var}\left(\epsilon\right)$ the *irreducible error*: no matter how good our estimate of $f()$ is, this is the portion of Y that simply cannot be explained using our predictors X. The irreducible error may stem from missing variables, unmeasurable variables, measurement error, or another cause. You will derive this equality in (3) in HW1!

These error quantities are important even if our primary goal is interpretation rather than prediction. We typically do not want to spend too much time interpreting the findings from a model where $Y \not\approx \hat{Y}$!

1.3.3 The best possible model?

What is the function $\hat{f}$ that minimizes the expected prediction error? The expectation in (2) is taken over the joint distribution of Y and X. Using the law of total expectation, we can rewrite this as:

$$E_X\left[E_{Y|X}\left[(Y - \hat{f}(X))^2 \mid X\right]\right].$$

In the inner expected value, X is no longer random. It turns out that we know what the point-wise minimum is of the inner expected value:

$$\underset{c}{\text{argmin}}\ \text{E}_{Y|X=x}\left[(Y-c)^2 \mid X = x\right] = E[Y \mid X = x].$$

If $E[Y \mid X = x]$ minimizes the point-wise expected prediction error at every x, then the function $\hat{f}(x) = E[Y \mid X = x]$ is the function that minimizes the overall expected prediction error. Of course, we do not know the joint distribution of X and Y, and so we cannot simply set $\hat{f}(x)$ to be this conditional expectation. But we can develop methods that attempt to approximate $\hat{f}(x) = E[Y \mid X = x]$ under various sets of assumptions!

Since we cannot optimize over the set of ALL functions, we are typically going to restrict our attention to functions $\hat{f}$ that fall into some specified model class $\mathcal{F}$. We will then come up with the function $\hat{f} \in \mathcal{F}$ that appears to perform the best according to our training dataset. We will see some examples below.

1.4 A case study in regression

Let's review a really simple case. We have a single numerical response variable Y and a single numerical predictor variable X. We observe 100 datapoints, so $n = 100$ and $p = 1$. Our goal is to use our dataset $\mathbf{x} = (x_1, \ldots, x_n)$, $\mathbf{y} = (y_1, \ldots, y_n)$ to come up with a function $\hat{f}$ such that, on new, unseen data, $Y \approx \hat{f}(X)$. We will consider two methods for coming up with $\hat{f}$; one of which is likely familiar and the other is likely unfamiliar.

[2] Introductory Machine Learning courses taught in CS departments might derive everything starting from the "no noise" setting where things can be perfectly predicted! This is a different perspective that may be useful sometimes! But as a statistician, I believe that we always have random noise!

1.4.1 Simple linear regression

We let $\mathcal{F} = \{f : f(x) = b_0 + b_1 x, b_0 \in \mathbb{R}, b_1 \in \mathbb{R}\}$. Based on our training data, we let:

$$\hat{\beta}_0, \hat{\beta}_1 = \underset{b_0, b_1}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2. \tag{4}$$

We then lat $\hat{f}(X) = \hat{\beta}_0 + \hat{\beta}_1 X$. Under the assumption that $E[Y] = \beta_0 + \beta_1 X$ for some true, unknown β_0 and β_1, this solution directly approximates the best possible function $E[Y \mid X]$. However, if this assumption does *not* hold, then the linear model is quite rigid, and might not do a good job approximating $E[Y \mid X = x]$ for all x.

You are all experts in linear regression! So we will not spend too much more space in the notes on it.

1.4.2 K-nearest neighbors

KNN is at the opposite end of the spectrum from linear regression. The premise is quite simple. KNN lets

$$\hat{f}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i, \tag{5}$$

where $N_k(x)$ is a function that returns the k points in the training set that are closest to the input point x according to some distance metric (for us, Euclidean distance). So, avoiding equations: KNN makes predictions by finding the k training observations with x_i closest to x, and predicts that the response for x will be the average of the responses for those k points.

The hyper-parameter k is chosen by the user. When $k = n$, KNN just predicts $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$ for all observations, regardless of x. As k decreases, the functions returned by KNN get increasingly wiggly and flexible. When $k = 1$, the KNN function is a step function that can be arbitrarily wiggly, and that always has 0 training error.

Note that KNN directly approximates $E[Y \mid X = x]$ *locally*, making no assumptions on the structure of $E[Y \mid X]$. When k is small, the estimation of $E[Y \mid X = x]$ uses only data that are really close to x, which is desirable if we believe a very local approximation is necessary. However, when k is small, each prediction is made using only a few datapoints, which makes the predictions less stable and more variable. We will explore this tradeoff thoroughly next class!

Linear regression is parametric; we can write down the function $\hat{f}$ by writing down the estimated values of two parameters. KNN is non-parametric; we cannot write down the function $\hat{f}$ without storing the entire dataset. T

Linear regression in theory takes some time to train, but it is nearly instantaneous to make new predictions. KNN involves no training step, but it could be computationally expensive to obtain new predictions.

1.4.3 Comparison

If KNN is so flexible, why wouldn't we always prefer KNN to linear regression? This is the topic of our next class!