Homework 1

Brandon Amaral, Monte Davityan, Nicholas Lombardo, Hongkai Lu

September 1, 2022

ISLR Chapter 2

What is statistical learning?

We make use of statistical learning to investigate and predict association between data. We feed our process inputs, also known as predictors, independent variables, or sometimes just "variables," typically denoted "X." In return, we get output variables, also called responses or dependent variables, typically denoted "Y". For example, in searching for insight about how to increase sales in a given market, a business is seeking to maximize "sales" as its output variable. The business can consider an array of known inputs such as advertising spent separately on TV, radio, and newspaper in any given market. Statistical learning seeks to estimate a function "f" that, when given an array of inputs, either accurately predicts an output, or provides insight on the effect of the inputs on the output.

Quoting James et al., "More generally, suppose that we observe a quantitative response Y and p different predictors, $X_1, X_2, ..., X_p$. We assume that there is some relationship between Y and $X = (X_1, X_2, ..., X_p)$, which can be written in the very general form

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

Here f is some fixed by unknown function of $X_1, ...X_p$, and ϵ is a random **error term**, which is **independent of X** and has **mean zero**. In this formulation, f represents the **systematic** information that X provides about Y."

Usually, the function f that connects each input variable to its corresponding output variable is **unknown**. Thus, we must usually use statistical methods to estimate the relationship f based on the observed points. Generally, the function f may involve more than one input variable.

The two main reasons we estimate f are **prediction** and **inference**. With prediction, we are generally not concerned with the form or workings of our predictor functions as long as it yields accurate results for Y. Since error terms average to zero, we predict Y using

$$\hat{Y} = \hat{f}(X)$$

where $\hat{f}$ represents our estimate for f, and $\hat{Y}$ represents the resulting prediction for Y. Again, since we don't necessarily need to understand how $\hat{f}$ works, we can treat it as a **black box**.

The two quantities that determine the accuracy of $\hat{Y}$ as a predictor for Y are **reducible error** and **irreducible error**. "In general, $\hat{f}$ will not be a perfect estimate of f, and this inaccuracy will introduce some error. This error is **reducible** because we can potentially improve the accuracy of $\hat{f}$ by using the most appropriate statistical learning technique to estimate f." Even if we could produce a perfect estimate for f, the prediction would still have some quantity of error. "This is because Y is also a function of ϵ, which, by definition, cannot be predicted using X. Therefore, variability associated with ϵ also affects the accuracy of our predictions." We call this **irreducible error** because we cannot perfectly estimate f no matter how well we try. "The quantity ϵ may contain unmeasured variables that are useful in predicting Y: since we don't measure them, f cannot use them for its prediction." Quantity ϵ may also contain unmeasured variation due to random chance, or noise.

On the other hand, we often seek to understand the association between predictors $X_1, ..., X_p$ and Y. Here, we do wish to estimate f, but we are not necessarily interested in making predictions for Y. Now $\hat{f}$

cannot be treated as a black box because we **need to know its exact form**. We may be interested in questions such as:

- "Which predictors are associated with the response?" We may seek to eliminate predictors from our consideration which do not have a significant predicting effect.
- "What is the relationship between the response and each predictor?" Some predictors may have a positive relationship with Y, while others have the opposite. Sometimes, the relationship between predictors themselves is predictive of Y.
- "Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?"

An example **inference** question could be, "How much extra will a house be worth if we add an extra bedroom?" An example **prediction** question could be, "Is this house under- or over-valued?" It is important to determine the goal type of our analysis so that we may choose an appropriate method for estimating f. Simpler models, such as linear regressions, are more interpretable.

In all methods of estimating f, we will assume we have observed a set of n different data points. These observations are called the **training data** because we use them to train our method how to estimate f. We do this by applying a statistical learning method to find a function $\hat{f}$ such that $Y \approx \hat{f}(X)$.

Parametric methods involve a two-step model-based approach:

1. First, make an assumption about the functional form (shape) of f, such as linear in X, where

   $$f(X) = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p.$$

   This linear model greatly simplifies the problem of estimating f because we only need to estimate the $p + 1$ coefficients $\beta_0, ..., \beta_p$.

2. Next, we need a procedure that uses the training data to **fit** or **train** the model. In the linear example, we need to find $\beta_0, ..., \beta_p$ such that

   $$Y \approx \beta_0 + \beta_1 X_1 + ... + \beta_p X_p.$$

   The most common approach for fitting the model is the **least squares** method, though there are many more to learn in the future.

Notice that once we made an assumption, all we worked to find were the parameters of the model, thus why we call this simple approach **parametric**. Although this method is simple, it suffers a potential disadvantage: the model we choose may not match f, and no matter how good the parameters are, the fit may still be poor if the model is too far from the true f. We can address this disadvantage by choosing a more **flexible** model, such as one with more degrees of freedom. Flexible models can fit many different possible functional forms for f, though they generally require estimating a greater number of parameters. The disadvantage of flexible models is they may **overfit** the data, meaning the follow errors, or noise, too closely.

"**Non-parametric** methods do not make explicit assumptions about the functional form of f. Instead they seek an estimate of f that gets as close to the data points as possible without being too rough or wiggly." Their major advantage is avoiding the assumption of a particular functional form for f. However, non-parametric approaches suffer from a major disadvantage: they require a very large number of observations (far more than is typically needed for a parametric approach) in order to obtain an accurate estimate for f.

Here we answer a reasonable starting question, "Why would we ever choose to use a more restrictive method instead of a very flexible method?" First, we if we mainly interested in **inference** (over prediction), then restrictive models are much more interpretable. Linear models are very restrictive, but they are a good choice for inference because they are easy to understand. "As X increases, Y increases/decreases." On the other hand, very flexible approaches, like splines and boosting, lead to complicated estimates for f, making it difficult to understand how any individual predictor is associated with the response. In the case of **prediction**, since interpretability is not a concern, we may use more flexible models which may provide more accurate predictive power. Although we might think that the most flexible methods for fitting the training data will have the greatest accuracy in predicting test outputs, this is surprisingly not always the case. **We often obtain more accurate predictions using a less flexible method.** This phenomenon is due to the potential for overfitting the test data, where our model follows errors too closely.

Most statistical learning problems fall into one of two categories: **supervised** or **unsupervised**. In the supervised domain, for each observation of the predictor measurement(s) $X_i, i = 1, ..., n$ there is an associated response measurement y_i. In the unsupervised domain, we have a more challenging situation where for every observation $i = 1, ..., n$, we observe a vector of measurements x_i but **no associated response** y_i. It is impossible to fit a linear regression model since there is no response variable to predict. Here, "we can seek to understand the relationships between the variables or between the observations." For example, in cluster analysis, or clustering, we seek to ascertain whether the observations fall into relatively distinct groups. There is often, but not always, a clear distinction between whether a problem belongs in the supervised or unsupervised domain, such as with incomplete data sets.

Quantitative variables take on numerical values. We tend to refer to problems with a quantitative response as **regression** problems. **Qualitative** (or categorical) variables take on value in one of K different **classes**, or categories. We tend to refer to problems with qualitative responses as **classification** problems. However, while we typically use **linear regression** for quantitative responses, we typically use **logistic regression** for qualitative (two-class, or **binary**) response. "Thus, despite its name, logistic regression is a classification method. But since it estimates class probabilities, it can be thought of a regression method as well."

Assessing Model Accuracy

"Why is it necessary to introduce so many different statistical learning approaches, rather than just a single best method? There is no free lunch in statistics: no one method dominates all others over all possible data sets." To evaluate the performance of a statistical learning method, we measure how well its predictions actually match observed data. "In the regression setting, the most commonly-used measure is the **mean squared error (MSE)**, given by

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

where $\hat{f}(x_i)$ is the prediction that f gives for the ith observation." A small MSE indicates the predicted responses are very close to the true responses.

In general, "we do not really care how well the method works on the training data. Rather, we are interested in the accuracy of the predictions that we obtain we apply our method to previously unseen **test data**." "We want to know whether $\hat{f}(x_0)$ is approximately equal to y_0 where (x_0, y_0) is a previously unseen test observation not used to train the statistical learning method." We want to minimize **test MSE**, not **training MSE**. If we have a large number of test observations, we would minimize

$$Ave(y_0 - \hat{f}(x_0))^2.$$

However, if no test data are available, we must be cautious. We should not assume that minimizing training MSE will also minimize test MSE. Typically, as **flexibility** increases, the curves of $\hat{f}$ more closely fit the **observed** data, so we may be tempted to increase flexibility to very high degree if we don't have any test data to calculate a test MSE, because it is true that, as with training MSE, "test MSE initially declines as the level of flexibility increases. However, at some point, the test MSE levels off and then starts to increase a 'gain'." "As the flexibility of the statistical learning method increases, we observe a monotone decrease in the training MSE and a **U-shape** in the test MSE. This is a fundamental property of statistical learning that holds regardless of the particular data set at hand and regardless of the statistical method being used. As model flexibility increases, training MSE will decrease, but the test MSE may not. When a given method yields a small training MSE but a large test MSE, we are **overfitting** the data." "Note that regardless of whether or not overfitting has occurred, we almost always expect the training MSE to be smaller than the test MSE because most statistical learning methods either directly or indirectly seek to minimize the training MSE. **Overfitting** refers specifically to the case in which a less flexible model would have yielded a smaller test MSE."

Expected test MSE can be shown to be

$$E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + [Bias(\hat{f}(x_0))]^2 + Var(\epsilon).$$

Thus, to minimize MSE, we need to select a statistical learning method that simultaneously achieves low **variance** and low **bias**. Note that both variance and squared bias are non-negative, so expected test MSE can never be less than $Var(\epsilon)$, the **irreducible error**. "**Variance** refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. On the other hand, **bias** refers to the error that is introduced by approximating a real-life problem (potentially extremely complicated) by a much simpler model." "As a general rule, increasing a models flexibility increases variance and decreases bias. As we increase the flexibility of a class of methods, the bias tends to initially decrease faster than variance increases," so test MSE initially declines too. "However, at some point, the increasing flexibility has little impact on the bias but starts to significantly increase the variance," so test MSE rises. This results in the "U-shape" of test MSE as a function of model flexibility. We call this the **bias-variance trade-off**.

Now consider the setting of **classification** (instead of regression). "The most common approach for quantifying the accuracy of our estimate $\hat{f}$ is the training **error rate**, the proportion of mistakes that are made if we apply our estimate $\hat{f}$ to the training observations:

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

Where $\hat{y_i}$ is the predicted class label for the ith observation using $\hat{f}$, and $I(y_i \neq \hat{y_i})$ is an **indicator variable**. The **test error** rate associated with a set of test observations of the form (x_0, y_0) is given by

$$Ave(I(y_0 \neq \hat{y_0}))$$

where $\hat{y_0}$ is the predicted class label resulting from applying the classifier to the test observation with predictor x_0." "A good classifier is one for which the test error is smallest."

"The test error rate is minimized, on average, by a very simple classifer that assigns each observation to the most likely class, given its predictor values." The **Bayes classifier** is where we seek to maximize the conditional probability,

$$Pr(Y = j|X = x_0).$$

The Bayes classifier's predictions are determined by the **Bayes decision boundary**; an observation that falls one side of the boundary will be assigned to a particular class, while an observation on the other side will belong to the other class. The **Bayes error rate**, this method's test error rate, will be $1 - max_j Pr(y = j|x = x_0)$ at $X = x_0$. In general, the overall Bayes error rate is given by

$$1 - E(max_j Pr(Y = j|X)).$$

The Bayes error rate is analogous to **irreducible error**. "For real data, we do not know the true conditional distribution of Y given X, and so computing the Bayes classifier is impossible. Therefore, the Bayes classifier serves as an unattainable gold standard against which to compare other [classification] models."

K-nearest neighbors (KNN) attempts to estimate the conditional distribution of Y given X, and then classify a given observation to the class with the highest estimated probability. "Given a positive integer K and a test observation x_0, the KNN classifier first identifies the K points in the training data that are closest to x_0, represented by N_0. It then estimates the conditional probability for class j as a fraction of the points in N_0 whose response values equal j:

$$Pr(Y = j|X = x_0) = \frac{1}{K}\sum_{i \in N_0} I(y_i = j).$$

Finally, KNN classifies the test observation x_0 to the class with the largest probability from the above equation.

Our choice of K has a drastic effect on the KNN classifier obtained, similar to choosing degrees of freedom for a model. "When $K = 1$, the decision boundary is overly flexible and finds patterns in the data that don't correspond to the Bayes decision boundary. This corresponds to a classifier that has low **bias** but high **variance**. As K grows large, KNN becomes less flexible and produces a decision boundary that is close to linear, corresponding to a high-bias but low-variance classifier. Typically, neither a very low nor a very high K values will give good predictions.

"In both the regression and classification settings, choosing the correct level of flexibility is critical to the success of any statistical learning method. The **bias-variance tradeoff**, and the resulting U-shape in the test error, can make this a difficult task."

CASI Chapters 1 & 2

Chapter 1

The broad definition of statistics provided by the book suggests that statistics is the science of learning from experience. However, that experience need not be provided all at the same time, but can be accumulated over time.

There are two primary, leading statistical theories: Frequentist and Bayesian.

The book makes a case of differentiating between algorithms and inferences of statistics. The example provided is that the average (mean) provides a summary of the data compressed into a single number. The mean is a representation in this case of an algorithm.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

In order to properly determine the accuracy and relevance of that algorithm, an inferential technique can be used, in this case being the standard error.

$$\hat{se} = [\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n(n-1)}]^{\frac{1}{2}}$$

An important note the book makes is that **the data that you use in an algorithm to make an estimate can also be used to determine that algorithms/ estimates accuracy**. Also, the standard error ($\hat{se}$) above is also an algorithm and inferential techniques could be used to determine its accuracy.

An example of kidney quality as a function of age is provided with a linear regression and lowess (a seemingly non- parametric regression curve algorithm). The example showcases that linear regression has statistical properties useful in the inference of the model when determining standard errors; whereas, lowess being non- parametric does not have nice inferential properties but that these properties are not required because methods such as bootstrapping exist. It also showcases that although lowess is more flexible (with very little to no model assumption), the standard errors are higher due to this increased flexibility.

Another example of differences in leukemia patients showcases the task of hypothesis testing in comparison to the prior example of estimation. The example reminds us that a low p-value represents a surprise difference in the expectations of the real world. However, the problem with hypothesis testing is with large sample size, a large t value (or low p-value) will be obtained even with no significant difference between the two samples. However, methods and techniques exist such as the False Discovery Rate to adjust for this large sample size.

In general, the process of algorithm and technique development often occur with the motivations of a specific real world problem; then, statisticians frame these new algorithms and techniques into statistical theory.

Chapter 2

This chapter focuses mostly on inference using the frequentist approach. There is usually an assumption in statistical inference that some probability model has created the observed data. The formula for bias is:

$$bias = E(\theta) - \theta$$

Frequentism is defined as "the probabilistic properties of a procedure of interest are derived and then applied verbatim to the procedure's output for the observed data." However, this requires the properties of an unknown distribution. In practice, different techniques can be used for this problem:

- Plug in Principle: For example $se(x) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})}{n}}$ is a biased estimate of the standard error but altering the formula to $se(x) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})}{n-1}}$
- Taylor Series Approximation: Approximations can be used for example $se(\bar{x}^2) = 2|\bar{x}|\hat{se}$
- Parametric Families and Maximum Likelihood
- Simulation and Bootstrap
- Pivotal Statistics: Is a distribution who doesn't depend on the underlying distribution. The Students t- sample t test is an example

Another downside of frequentist inference is while it is flexible, it is not obvious which algorithm should be chosen. Frequentist optimality is the study of choosing the best distribution/ algorithm. Some techniques used for frequentist optimality include:

- Fisher's theory of maximum likelihood estimation
- Fisher information bound
- Neyman–Pearson lemma

PML Chapter 1

A popular definition of machine learning, due to Tom Mitchell is

> A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Supervised Learning

Supervised learning is the most common form of machine learning, where the task T is to learn a mapping f from inputs $x \in \mathcal{X}$ (features, covariates, or predictors) to outputs $y \in \mathcal{Y}$ (label, target, or response). In this case, $\mathcal{X} = \mathbb{R}^D$, where D is the dimensionality of the vector. The experience E is a set of N input-output pairs $\{(\boldsymbol{x_n}, \boldsymbol{y_n}\}_{n=1}^{N}$, known as the training set. The performance measure P depends on the type of output, classification or regression.

Classification

The output space of a classification problem is a set of unordered and mutually exclusive labels known as classes. It is common to store small datasets of features as an $N \times D$ matrix, where each row is an example and each column is a feature. This is known as a design matrix.

Exploratory Data Analysis

For tabular data with a small number of features, a pair plot is common to use, in which each panel shows a scatter plot of each of the variables against each other, with the diagonal entries showing the marginal density of the i^{th} variable. For higher-dimensional data, it is common to perform dimensionality reduction, and then visualize the data.

Learning a classifier

A simple example of a classifier is a one-dimensional decision boundary $x_i < c$ for some constant c. The input space is thus partitioned into two regions, i.e. two classifications. To improve on this, we can recursively partition the space, splitting regions in which the classifier makes errors. These nested rules can be arranged into a tree structure, called a decision tree.

Empirical Risk Minimization

Supervised learning attempts to create classification models to reliably predict labels for any given input. To measure performance, we can use the misclassification rate on the training set:

$$\mathcal{L}(\boldsymbol{\theta}) \triangleq \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}(y_n \neq f(\boldsymbol{x_n};\boldsymbol{\theta})),$$

where $\mathbb{I}(e)$ is the indicator function, i.e.

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{if } e \text{ is false} \end{cases}.$$

It may be the case however, that some errors are more costly than others, so instead of using a function that assumes all errors are equal, we might use an asymmetric loss function $\ell(y, \hat{y})$. Then we define the empirical risk to be the average loss of the predictor on the training set:

$$\mathcal{L}(\boldsymbol{\theta}) \triangleq \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x_n};\boldsymbol{\theta})).$$

One way to define model fitting or training, is to find parameters that minimize the empirical risk on the training set:

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x_n};\boldsymbol{\theta}))$$

This is empirical risk minimization. The general goal, however, is to generalize this notion to the expected loss on future data that has not yet been seen.

Uncertainty

Epistemic uncertainty, or model uncertainty, is the idea that we will not be able to perfectly predict the exact output given an input, due to lack of knowledge about the input-output mapping. This may also be due to aleatoric uncertainty, or data uncertainty, which is an irreducible stochasticity in the mapping.

To reduce uncertainty, one may be able to perform an information gathering action. We can capture uncertainty with the conditional probability distribution:

$$p(y = c|\boldsymbol{x};\boldsymbol{\theta}) = f_c(\boldsymbol{x};\boldsymbol{\theta}),$$

where $f : \mathcal{X} \to [0,1]^C$ maps inputs to a probability distribution over the C possible output labels. Note that we then require $0 \leq f_c \leq 1$ for each c, and $\sum_{c=1}^{C} f_c = 1$. It is common to return unnormalized log-probabilities instead, which can be converted to probabilities using the softmax function:

$$\text{softmax}(\boldsymbol{\alpha}) \triangleq \left[\frac{e^{\alpha_1}}{\sum_{c'=1}^{C} e^{\alpha_{c'}}}, \ldots, \frac{e^{\alpha_C}}{\sum_{c'=1}^{C} e^{\alpha_{c'}}}\right],$$

which satisfies softmax : $\mathbb{R}^C \to [0,1]^C$, $0 \leq \text{softmax}(\boldsymbol{\alpha}) \leq 1$, and
$\sum_{c=1}^{C} \text{softmax}(\boldsymbol{\alpha})_c = 1$. The inputs $\boldsymbol{\alpha}$ are called logits. Thus, the overall model is

$$p(y = c|\boldsymbol{x};\boldsymbol{\theta}) = \text{softmax}_c(f(\boldsymbol{x};\boldsymbol{\theta}))$$

If f is an affine function of the form

$$f(\boldsymbol{x};\boldsymbol{\theta}) = b + \boldsymbol{w^T x} = b + w_1x_1 + \cdots + w_Dx_D,$$

where $\boldsymbol{\theta} = (b, \boldsymbol{w})$ are the parameters of the model, this is called logistic regression. The $\boldsymbol{w}$ parameters are called regression coefficients, with b being the intercept. They may also be called weights and bias respectively. It is common to absorb the bias term into the weights by defining $\tilde{\boldsymbol{w}} = [b, w_1, \dots, w_D]$ and $\tilde{\boldsymbol{x}} = [1, x_1, \dots, x_D]$ so that $\tilde{\boldsymbol{w}}^{\boldsymbol{T}}\tilde{\boldsymbol{x}} = b + \boldsymbol{w^T x}$. Thus, the affine function is now assumed to be a linear function and can write the prediction function as $f(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w^T x}$.

Maximum Likelihood Estimation

It's common to use the negative log probability as the loss function:

$$\ell(y, f(\boldsymbol{x};\boldsymbol{\theta}) = -\log p(y|f(\boldsymbol{x};\boldsymbol{\theta})).$$

The intuition is that a good model is one that assigns high probability to the true output y for each corresponding input $\boldsymbol{x}$. The negative log likelihood is given by

$$\text{NLL}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{n=1}^{N} \log p(y_n|f(\boldsymbol{x_n};\boldsymbol{\theta})),$$

and hence the maximum likelihood estimate for $\boldsymbol{\theta}$ is

$$\hat{\boldsymbol{\theta}}_{mle} = \underset{\boldsymbol{\theta}}{\text{argmin}}\, \text{NLL}(\boldsymbol{\theta}).$$

Regression

Predicting a real-valued quantity $y \in \mathbb{R}$ is known as regression. Since the output is real-valued, a common choice for the loss function is to use quadratic loss or ℓ_2 loss:

$$\ell_2(y, \hat{y}) = (y - \hat{y})^2,$$

which notably penalizes large residuals $y - \hat{y}$ more than small ones. The empirical risk for this loss function is the mean squared error:

$$\text{MSE}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}(y_n - f(\boldsymbol{x_n};\boldsymbol{\theta}))^2.$$

In regression problems, it's common to assume the output distribution is Gaussian or normal, defined by

$$\mathcal{N}(y|\mu, \sigma^2) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-\mu)^2},$$

where μ is the mean, σ^2 is the variance, and $\sqrt{2\pi\sigma^2}$ is the normalization constant for the density. Defining $\mu = f(\boldsymbol{x_n};\boldsymbol{\theta})$, we have

$$p(y|\boldsymbol{x};\boldsymbol{\theta}) = \mathcal{N}(y|f(\boldsymbol{x};\boldsymbol{\theta}), \sigma^2).$$

Assuming constant variance, we have

$$\begin{aligned}\text{NLL}(\boldsymbol{\theta}) &= -\frac{1}{N}\sum_{n=1}^{N_D} \log\left[\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} \exp\left(-\frac{1}{2\sigma^2}(y_n - f(\boldsymbol{x_n};\boldsymbol{\theta}))^2\right)\right] \\ &= \frac{1}{2\sigma^2}\text{MSE}(\boldsymbol{\theta}) + \text{ const.}\end{aligned}$$

Note that the NLL is proportional to the MSE, so computing the MLE of the parameters results in minimizing the squared error.

Linear Regression

A simple linear regression model is of the form

$$f(x;\boldsymbol{\theta}) = b + wx,$$

where w is the slope, b is the offset, and $\boldsymbol{\theta} = (w, b)$ are the parameters of the model. Adjusting $\boldsymbol{\theta}$, we can minimize the sum of squared errors to find the least squares solution

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}).$$

If there are multiple input features, multiple linear regression has the form

$$f(\boldsymbol{x};\boldsymbol{\theta}) = b + w_1 x_1 + \cdots + w_D x_D = b + \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x},$$

where $\boldsymbol{\theta} = (\boldsymbol{w}, b)$.

Polynomial Regression

A polynomial regression model of degree D has the form $f(x;\boldsymbol{w}) = \boldsymbol{w}^{\boldsymbol{T}}\phi(x)$, where $\phi(x)$ is a feature vector derived from the input of the form

$$\phi(x) = [1, x, x^2, \ldots, x^D].$$

D can be increased until $D = N - 1$, i.e. there is one parameter per data point, and the data is perfectly interpolated. This model has 0 MSE, but will not be a good predictor for future inputs.
Polynomial regression can also be applied to multi-dimensional inputs, e.g.

$$f(\boldsymbol{x};\boldsymbol{w}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2.$$

Note that the prediction function is still a linear function of the parameters $\boldsymbol{w}$, even though it is nonlinear in the input $\boldsymbol{x}$. A linear model induces an MSE loss function with a unique global optimum.

Deep Neural Networks

We can create more powerful models by learning to do nonlinear feature extraction automatically. If we let $\phi(x)$ have its own parameters $\boldsymbol{V}$, then the model has the form

$$f(\boldsymbol{x};\boldsymbol{w},\boldsymbol{V}) = \boldsymbol{w}^{\boldsymbol{T}}\phi(\boldsymbol{x};\boldsymbol{V}).$$

Further decomposing ϕ, the model becomes L nested functions

$$f(\boldsymbol{x};\boldsymbol{\theta}) = f_L(f_{L-1}(\cdots(f_1(\boldsymbol{x})\cdots)),$$

where $f_\ell(\boldsymbol{x}) = f(\boldsymbol{x};\boldsymbol{\theta}_\ell$ is the function at layer ℓ. The final layer is linear and has the form $f_L(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}} f_{1:L-1}(\boldsymbol{x})$, where $f_{1:L-1}(\boldsymbol{x})$ is the learned feature extractor. This is the key idea for deep neural networks.

Overfitting and Generalization

We can rewrite the epirical risk as

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{D}_{\text{train}}) = \frac{1}{|\mathcal{D}_{\text{train}}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}_{\text{train}}} \ell(\boldsymbol{y}, f(\boldsymbol{x};\boldsymbol{\theta})),$$

where $|\mathcal{D}_{\text{train}}|$ is the size of the training set $\mathcal{D}_{\text{train}}$.

We can drive the training loss to zero by memorizing the correct output for each input, but this may not be useful for prediction accuracy. A model that perfectly fits the training data, but is too complex is overfitting.

To detect model overfitting, we can compute the theoretical expected loss or population risk:

$$\mathcal{L}(\boldsymbol{\theta}; p^*) \triangleq \mathbb{E}_{p^*(\boldsymbol{x},\boldsymbol{y})}[\ell(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta}))],$$

where p^* is the true distribution used to generate the training set. The difference $\mathcal{L}(\boldsymbol{\theta}; p^*) - \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}})$ is the generalization gap. A large generalization gap is a sign of overfitting.

Since p^* is unknown, we partition the data into training and test sets to approximate the population risk using

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{test}}) \triangleq \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}_{\text{test}}} \ell(y, f(\boldsymbol{x}; \boldsymbol{\theta})).$$

To pick a model of the right complexity, we should pick the model with the minimum test loss. In practice, we partition data into training, test, and validation sets, the latter of which is used for model selection.

No Free Lunch Theorem

There is no single best model that works optimally for all problems. The best way to pick a model is based on domain knowledge and/or trial and error.

Unsupervised Learning

Unsupervised learning attempts to understand observed inputs $\mathcal{D} = \{\boldsymbol{x_n} : n = 1 : N\}$ without any corresponding outputs $\boldsymbol{y_n}$. From a probabilistic perspective, the difference between unsupervised and supervised learning is fitting an unconditional model $p(\boldsymbol{x})$, which can generate new data $\boldsymbol{x}$, versus fitting a conditional model $p(\boldsymbol{y}|\boldsymbol{x})$, which specifies a distribution of outputs given the inputs.

Unsupervised learning avoids the need for large labeled datasets, avoids the need for learning to partition the data into arbitrary categories, and forces the model to explain the high-dimensional inputs, rather than the low-dimensional outputs.

Clustering

The goal of clustering is to partition the input into regions that contain similar points. There may not be a correct number of clusters. Instead, we must consider the tradeoff between model complexity and fit to the data.

Discovering Latent "Factors of Variation"

It's often useful to reduce dimensionality by projecting onto a lower dimensional subspace which captures the essence of the data. One approach is to assume there are hidden or unobserved low-dimensional latent factors $\boldsymbol{z_n} \in \mathbb{R}^K$ which generate the high dimensional output $\boldsymbol{x_n} \in \mathbb{R}^D$. We can represent this as $\boldsymbol{z_n} \to \boldsymbol{x_n}$, where the arrow represents causation. We often assume a simple prior probability model for $p(\boldsymbol{z_n})$, and if the data is real-valued, we can use a Gaussian likelihood as well.

A simple example is a linear model $p(\boldsymbol{x_n}|\boldsymbol{z_n}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x_n}|\boldsymbol{W}\boldsymbol{z_n} + \boldsymbol{\mu}, \boldsymbol{\Sigma})$. The resulting model is called factor analysis. In the special case $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$, this is called probabilistic principal components analysis.

Self-supervised Learning

We create proxy supervised tasks from unlabeled data. The idea is that the resulting predictor $\hat{\boldsymbol{x}}_{\boldsymbol{1}} = f(\boldsymbol{x}_2; \boldsymbol{\theta})$, where $\boldsymbol{x}_2$ is the observed input and $\hat{\boldsymbol{x}}_{\boldsymbol{1}}$ is the predicted output, will learn useful features from the data that can then be used in standard, downstream supervised tasks.

Evaluating Unsupervised Learning

A common method for evaluating unsupervised models is to measure the probability assigned by the model to unseen test examples by computing the negative log likelihood of the data:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}_{\text{test}}} \log p(\boldsymbol{x}|\boldsymbol{\theta}).$$

This treats the problem as density estimation. The model then learns to capture typical patterns in the data.

An alternative evaluation metric is to use the learned unsupervised representation as features or input to a downstream supervised learning method. If this works well, it should be possible to perform supervised learning using much less labeled data than when working with the original features. We can increase the sample efficiency of learning (i.e. reduce the number of labeled examples needed to get good performance) by first learning a good representation.

The goal of unsupervised learning is to gain understanding, not necessarily to improve performance on a prediction task. This requires interpretable models that can also explain most of the observed patterns in the data.

Reinforcement Learning

The system or agent has to learn how to interact with its environment. This can be encoded by means of a policy $\boldsymbol{\alpha} = \pi(\boldsymbol{x})$, which specifies which action to take in response to each possible input $\boldsymbol{x}$. The difference from supervised learning is that the system is not told which action is the best one to take. Instead, the system receives an occasional reward or punishment signal in response to actions. A key difficulty with RL is that the reward signal may only be given occasionally, and it may be unclear which of the actions were responsible for the getting the reward.

It's common to use other information sources, such as expert demonstrations in a supervised setting, or unlabeled data in an unsupervised setting to discover the underlying structure of the environment.

Preprocessing Discrete Input Data

Sometimes the input may have discrete features, such as categorical variables.

One-hot Encoding

For categorical features, we need to convert them to numerical scale, so that computing weighted combinations of the inputs makes sense. If a variable x has K values, one-hot encoding is one-hot$(x) = [\mathbb{I}(x = 1), \ldots, \mathbb{I}(x = K)]$.

Feature Crosses

A linear model using one-hot encoding for categorical variables, can capture the main effects of each variable, but can't capture interaction effects between them. We can fix this by computing explicit feature crosses, e.g.

$$f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{\phi}(\boldsymbol{x}) = w_0 + w_1 \mathbb{I}(x_1 = A) + w_2 \mathbb{I}(x_2 = B) + w_3 \mathbb{I}(x_1 = A, x_2 = B),$$

which converts the original dataset into a wide format with more columns.

Handling Missing Data

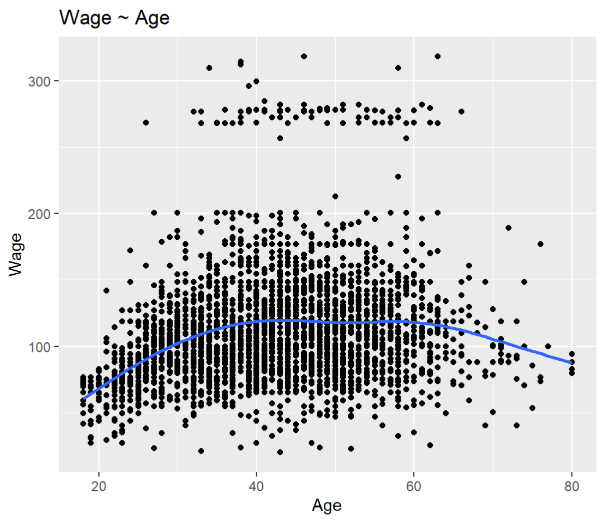
Parts of the input $\boldsymbol{x}$ or output y may be unknown. To model the case where some input features may be missing, let $\boldsymbol{M}$ be an $N \times D$ matrix of binary variables, where $M_{nd} = 1$ if feature d in example n is missing and 0 otherwise. Let $\boldsymbol{X_v}$ be the visible parts of the ipnut feature matrix, corresponding to $M_{nd} = 0$, and $\boldsymbol{X_h}$ be the missing parts, corresponding to $M_{nd} = 1$. Let $\boldsymbol{Y}$ be the output label matrix, which is assumed to be fully observed. If we assume $p(\boldsymbol{M}|\boldsymbol{X_v}, \boldsymbol{X_h}, \boldsymbol{Y}) = p(\boldsymbol{M})$, we say the data is missing completely at random, since the missingness doesn't depend on hidden or observed features. If we assume $p(\boldsymbol{M}|\boldsymbol{X_v}, \boldsymbol{X_h}, \boldsymbol{Y}) = p(\boldsymbol{M}|\boldsymbol{X_v}, \boldsymbol{Y})$, we say the data is missing at random, since the missingness does not depend on the hidden features, but may depend on visible features.

In the not missing at random case, we need to model the missing data mechanism, since the lack of information may be informative.

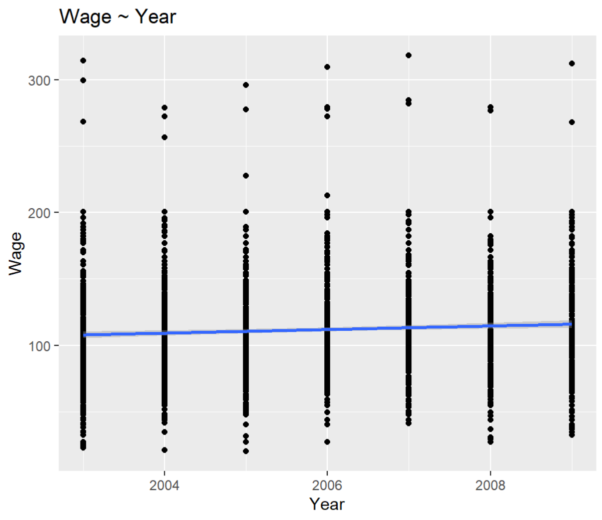
A common heuristic is called mean value imputation, in which missing values are replaced by their empirical mean. We can also fit a generative model to the input and use that to fill in the missing values.

Problem 2

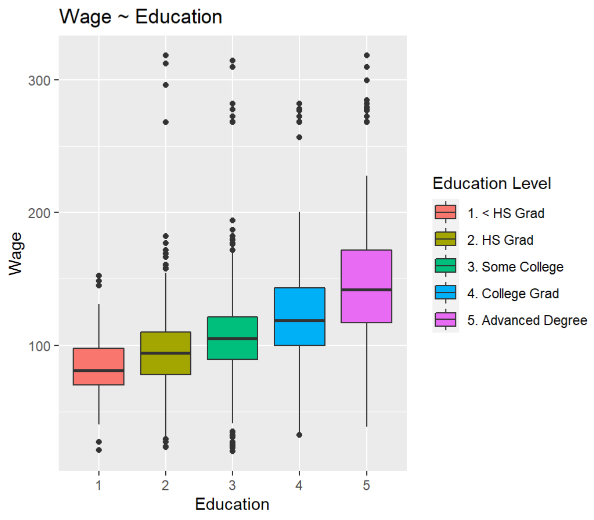
```
library(ggplot2)
library(ISLR)
library(tidyverse)

# wage as a function of age

ggplot(data = Wage, mapping = aes(x = age, y = wage)) +
  geom_point() +
  geom_smooth(se = FALSE) +
  ggtitle("Wage ~ Age") +
  xlab("Age") +
  ylab("Wage")
```



```
# wage as a function of year

ggplot(data = Wage, mapping = aes(x = as.integer(year), y = wage)) +
  geom_point() +
  geom_smooth(method = "lm") +
  ggtitle("Wage ~ Year") +
  xlab("Year") +
  ylab("Wage")
```



```
# wage as a function of education

Wage %>%
  # To replicate the graph we want to remove the extra details for what
  # education actually is and only focus on the id
  mutate(education_id = substr(education, 1, 1)) %>%
  mutate(education_id = as.factor(education_id)) %>%
  ggplot(mapping = aes(x = education_id, y = wage, fill = education)) +
    geom_boxplot() +
    ggtitle("Wage ~ Education") +
    xlab("Education") +
    ylab("Wage") +
    labs(fill = "Education Level")
```


Wage ~ Education
Wage
Education
Education Level
1. < HS Grad
2. HS Grad
3. Some College
4. College Grad
5. Advanced Degree