Handout 01: Statistical Learning

Introduction

I am planning on teaching this course quite a bit differently than when it was offered in Fall 2017. With the addition of a data science course, this frees us up to cover a more theoretical treatment of statistical learning. Of course, applications will still be discussed and our work will be very applicable to real-world applications. It is just that, at least at first, we will do a bit more with theoretical derivations. These will be most present early in the semester, in part because the theory for later material requires more advanced mathematics (such as functional analysis). Also, once we get to neural networks, most of the theory has not even been written yet.

The notes you have here, and many of those used this semester, come from a textbook I wrote with Michael Kane (Yale Biostatistics) and Bryan Lewis. As it comes out in April 2019, we will not be able to use it directly for the course. I will take notes liberally from the book and place them online here. Note that you may catch some strange language (such as "'in the next chapter...") that I simply missed when converting the notes. Hopefully these do not detract too much from the core material.

Our book *A Computational Approach to Statsitical Learning* was written to supplement the existing literature in statistical learning and predictive modeling. It provides a novel treatment of the computational details underlying the application of predictive models to modern datasets. It grew out of lecture notes from several courses we have taught at the undergraduate and graduate level on linear models, convex optimization, statistical computing, and supervised learning.

The major distinguishing feature of the text is the inclusion of code snippets that give working implementations of common algorithms for estimating predictive models. These implementations are written in the R programming language using basic vector and matrix algebra routines. The goal is to demystify links between the formal specification of an estimator and its application to a specific set of data. Seeing the exact algorithm used makes it possible to play around with methods in an understandable way and experiment with how algorithms perform on simulated and real-world datasets. This *try and see* approach fits a common paradigm for learning programming concepts. The reference implementations also illustrate the run-time, degree of manual tuning, and memory requirements of each method. These factors are paramount in selecting the best methods in most data analysis applications.

In order to focus on computational aspects of statistical learning, we highlight models that can be understood as extensions of multivariate linear regression. Within this framework, we show how penalized regression, additive models, spectral clustering, and neural networks fit into a cohesive set of methods for the construction of predictive models built on core concepts from linear algebra. The general structure of follows that of the two popular texts *An Introduction to Statistical Learning* (ISL) [2] and *The Elements of Statistical Learning* (ESL) [1].

Statistical learning

Statistical learning is the process of teaching computers to “learn” by automatically extracting knowledge from available data. It is closely associated with, if not outright synonymous to, the fields of pattern recognition and machine learning. Learning occupies a prominent place within artificial intelligence, which broadly encompasses all forms of computer intelligence, both hand coded and automatically adapted through observed data.

We will focus on the subfield of *supervised learning*. The goal is to find patterns in available inputs in order to make accurate predictions on new, unseen data. For this reason models used in supervised learning are often called *predictive models*. Take the task of building an automatic spam filter. As a starting point, we could label a small dataset of messages by hand. Then, a statistical learning model is built that discovers what features in the messages are indicative of the message being labeled as spam. The model can be used to automatically classify new messages without manual intervention by the user. Many applications can be written as supervised learning tasks:

- Estimate how much a house will sell for based on properties such as the number of bedrooms, location, and its size.
- Determine whether a mass detected in an MRI scan is malignant or benign.
- Predict whether a flight will be delayed given the carrier, scheduled departure time, the departure airport, and the arrival airport.
- Estimate the number of page views a website will need to handle next month.
- Given a picture of a person, predict their age and mood.
- Determine the correct department to forward an online form request sent to a company’s help desk.
- Predict the increased sales resulting from a new advertising campaign.

Domain-specific expertise is essential to the successful construction and deployment of statistical learning algorithms. However, most methods used for learning a predictive model from preprocessed data consist of applying general purpose training algorithms. The algorithms we cover can be applied to a wide range of tasks, including all of the above examples, given the availability of sufficiently representative data for training.

Example

It is useful to look at a concrete example of a statistical learning task. Consider recording the number of capital letters used in the content of 18 text messages and labeling whether the message is spam or “ham” (non-spam). For example, assume we observed the following dataset listing the number of capital letters followed by whether this is a spam message or not:

```
(0, ham)  (0, ham)   (0, ham)  (1, ham)   (1, ham)  (1, spam)
(2, ham)  (2, spam)  (2, ham)  (2, spam)  (2, ham)  (4, ham)
(5, spam) (5, spam)  (6, spam) (6, ham)   (8, spam) (8, spam)
```

As one might expect, messages with a large number of capital letters are more likely to be spam. Now, we will use this data to predict whether a new text message is spam based on the number of capital letters that are used. A straightforward model for this prediction task would be to select a cutoff value N, and classify new messages with N or more capital letters as spam. We can apply supervised statistical learning to select a value of N by making use of the observed data.

What is a good value for the parameter N based on the observed text data? If we set N equal to 2, the first row of data will be categorized as ham and the second two will be labeled as spam. This leads to one mistake in the first row and five in the second two rows, for a total of six mistakes. Setting it to 5, where only the last row is labeled as spam, yields a better error rate with only four mistakes. Looking exhaustively through all possible splits illustrates that the choice of 5 leads to the fewest errors, and therefore would be a good choice for N. To test how well this model works, finally, we could acquire an additional dataset of text messages and compute the new error rate for our chosen split value. We will discuss all of these details, with fully worked out examples, throughout the text.

Continuing with our example we illustrate the concept of a reference implementation by writing a best split algorithm for classification. Our function takes two vectors; one gives a numeric set of numbers on which to classify and the second defines the two classes we want to distinguish. It works by exhaustively testing each possible value in the first vector as a possible split point, and returns the best possible split value. If there is a tie, all possible best splits are returned as values.

```r
# Compute most predictive split of the training data.
#
# Args:
#     x: Numeric vector with which to classify the data.
#     y: Vector of responses coded as 0s and 1s.
#
# Returns:
#     The best split value(s).
casl_utils_best_split <-
```

```
function(x, y)
{
  unique_values <- unique(x)
  class_rate <- rep(0, length(unique_values))
  for (i in seq_along(unique_values))
  {
    class_rate[i] <- sum(y == (x >= unique_values[i]))
  }
  unique_values[class_rate == max(class_rate)]
}
```

The reference implementation uses common functions available in the base version of R. It should be readable, though perhaps not reproducible from scratch, by anyone familiar with scientific programming. Representing the data displayed in Section as objects in R, we can test the implementation to make sure that it returns the same best fit value we arrived at by manually checking each split value.

```
x <- c(0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2, 4, 5, 5, 6, 6, 8, 8)
y <- c(0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1)

casl_utils_best_split(x, y)
```

```
[1] 5
```

We see that it does return the correct value of 5. Exercises at the end of this chapter provide an opportunity to further test and tweak the `casl_utils_best_split` function.

Formalisms and terminology

We conclude this introduction by mathematically formalizing the central elements of supervised learning. The resulting language and terminology will be useful as a reference throughout the course.

Assume that there exists some unknown function f that maps elements from a set Ω into a set $\mathcal{Y}$,

$$f : \Omega \rightarrow \mathcal{Y}, \tag{1.1}$$

and consider observing tuples, known as *training data*,

$$\left\{ \omega_i, y_i = f(\omega_i) \right\}_i, \quad \omega_i \in \Omega,\, y_i \in \mathcal{Y}, \quad i = 1, \ldots n. \tag{1.2}$$

We will avoid writing out a formal definition of the (possibly) random nature of f and the (possibly) random process that generates each ω_i. Doing so here would overly divert from the main discussion; we will define the random nature of the data generation process whenever necessary.

A supervised learning algorithm constructs an estimate $\widehat{f}$ of the function f using the training data. The goal is to minimize errors in estimating f on new data for some loss function $\mathcal{L}$. When predicting a continuous response variable, such as an expected price, common loss functions include the squared or absolute difference between the observed and predicted values,

$$\mathcal{L}(\widehat{f}(\omega_{new}), f(\omega_{new})) = \left|\widehat{f}(\omega_{new}) - f(\omega_{new})\right| \tag{1.3}$$

When the set of responses $\mathcal{Y}$ is finite, as it would be when building a spam prediction algorithm, a common choice of $\mathcal{L}$ is to measure the proportion of incorrectly labeled new observations,

$$\mathcal{L}(\widehat{f}(\omega_{new}), f(\omega_{new})) = \begin{cases} 0, & \widehat{f}(\omega_{new}) = f(\omega_{new}) \\ 1, & \text{otherwise} \end{cases} \tag{1.4}$$

Depending on the application, more complex metrics can be used. When evaluating a medical diagnostic algorithm, for instance, it may make more sense to weight incorrectly missing a serious condition more heavily than incorrectly diagnosing a serious condition.

In nearly all supervised learning tasks, the training data will either be given as, or coercible to, a vector of real values. In other words, we can write the set Ω in Equation 1.1 as $\mathbb{R}^p$ for some number p. Similarly, the prediction task can usually be re-written such that $\mathcal{Y}$ is equal to $\mathbb{R}$—in the case of a discrete set, this can be done by associating each category with an integer-based index. Then, by stacking the n training inputs together, we have

$$X = \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_n \end{pmatrix} \in \mathbb{R}^{n \times p}, \quad y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \in \mathbb{R}^n, \quad y = f(X). \tag{1.5}$$

The matrix X is known as the *feature matrix*. This simplification allows for us to draw on techniques from numerical analysis, functional analysis, and statistics in the pursuit of predictive models. A central theme of this course will be building and evaluating supervised learning algorithms motivated by the study of properties of the matrix X and assumptions made regarding the function f.

If the success of a supervised learning algorithm is defined on data that is, by definition, unavailable, how will it be possible to determine how well a predictive model is able to estimate the function f? One approach is to take the observed tuples of data available for building predictive models and partition them into two subsets. Only one of these partitions is used as the *training set* to produce the estimate $\widehat{f}$. The remaining partition, the *testing set*, is used to evaluate how well the estimate can make predictions on new data. More complex schemes operate similarly by splitting the data multiple times (e.g., *cross-validation*) or include a third partition to allow for tuning hyperparameters in the model estimation algorithm.

When considering the construction of a predictive model, a key concern is the desired capacity, or complexity, of a model building algorithm. A formal definition of a training algorithm's complexity is given by its Vapnik–Chervonenkis (VC) dimension [3], though an informal understanding of complexity will suffice here. A model that is overly complex will *overfit* to the training data; that is, $\widehat{f}$ will fit the training data very closely but not be able to generalize well to the testing data. Conversely, a model with low complexity may be too constrained to provide a good approximation to f. In a probabilistic framework, this can be seen as equivalent to a trade-off between *bias* and *variance*. A model building algorithm with high complexity will have a large variance (it may overfit, and is therefore highly sensitive to the training data); if it has a complexity that is too low to approximate f, it will provide systematically biased results for some inputs. The study and control of model complexity, in both its numerical and probabilistic forms, is a guiding theme throughout this text.

References

[1] Friedman, J., Hastie, T., and Tibshirani, R. *The Elements of Statistical Learning*. Springer, New York, NY, 2001.

[2] James, G., Witten, D., Hastie, T., and Tibshirani, R. *An Introduction to Statistical Learning*, vol. 112. Springer, New York, NY, 2013.

[3] Vapnik, V., and Chervonenkis, A. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications 16*, 2 (1971), 264.

LAB QUESTIONS

1. Describe what the spaces Ω and $\mathcal{Y}$ would be for a predictive modeling task that estimates whether an email message is either spam or not spam.

2. Describe what might be some possible columns of the dataset X in a task trying to predict the sale price for a used car.

3. Assume that the feature matrix X has only one column, with values evenly distributed between 0 and 1, and the true function f is deterministic and defined such that to $y_i = x_i^2$. If our modeling algorithm is only allowed to produce a $\widehat{f}$ equal to a linear function of x_i, illustrate using a hand drawn plot how (in this case) the low model complexity directly leads to model bias.

4. Assume that the feature matrix X has only one column, with values evenly distributed between 0 and 1, and the true function f is deterministic and defined such that to $y_i = x_i^2$. If our modeling algorithm is only allowed to produce a $\widehat{f}$ equal to a linear function of x_i, illustrate using a hand drawn plot how (in this case) the low model complexity directly leads to model bias.

5. Using the same setup as in the previous question, but allowing $\widehat{f}$ to take on any form, illustrate an example of a curve that has been overfit to the data.

6. Consider the following small dataset:

x	y
1	0
2	1
3	0
4	0
5	1
6	1
7	0
8	1
9	1
10	1

   Answer the following questions (you should be able to do this by hand):

   - What is the best split value derived from this data? What value does your model predict will be the value of y if $x = 1$? How about $x = 10$ and $x = 5$?
   - Using mis-classification loss, what is the loss of your model on this dataset?

7. Take the following data as a *test set* for the model you built in the prior question:

x	y
4	0
4	1
4	1
8	0
8	1
8	0
8	1

What is the misclassification rate on the test data? Is this value better or worse than the loss function applied to the training data?

8. If you have not yet installed R, do that now. Using code similar to that in the notes, redo the best split prediction using R. Does it give the same output that you had when computing it by hand?
9. (optional) Modify the function `cnlp_utils_best_split` so that it also checks whether the best split would classify every value as a `1` if it is less than some cutoff value. Make sure that the output makes it clear whether the cutoff is a less than or greater than value.