PG Diploma in Data Analytics

Course: Predictive Analytics - II

Instructor: Prof. G. Srinivasaraghavan

PGDDA-Lecture Notes/PA-II/1

Model Selection

Contents

List of Figures

List of Tables

1 Learning Objectives

After this module you are expected to be familiar with some of the key concerns in selecting an appropriate model for a task after an objective evaluation.

1. Build the Conceptual Foundation for model selection, introducing in the process some frequently occurring jargon in Machine Learning:
   (a) Occam's Razor
   (b) Overfitting
   (c) Regularization
   (d) Bias-Variance Tradeoff
   (e) Model Complexity
2. Be familiar with some of the common model evaluation strategies, including a typical methodology for model evaluation and selection:
   (a) Cross Validation
   (b) Hold-Out Strategy
   (c) Model Evaluation Tools — Classification
      i. Models that output a Class Label
      ii. Models that output a probability distribution among class labels
   (d) Model Evaluation Tools — Regression

2 Introduction

The central issue in all of Machine Learning is "how do we extrapolate what has been learnt from a finite amount of data to all possible inputs 'of the same kind'?". We build models from some training data. However the training data is always finite. On the other hand the model is expected to have learnt 'enough' about the entire domain from where the data points can possibly come. Clearly in almost all realistic scenarios the domain is infinitely large. How do we ensure our model is as good as we think it is based on its performance on the training data, even when we apply it on the infinitely many data points that the model has never 'seen' (been trained on)? This module will introduce you to some of the perspectives on this question and its implications.

3 Occam's Razor

A predictive model has to be as simple as possible, but no simpler. Often referred to as the **Occam's Razor**, this is not just a convenience but a fundamental tenet of all of machine learning.

Before we explore this further, let's first get some intuitive understanding of what it means for a model to be 'simple'. To measure the simplicity we often use its complementary notion — that of the complexity of a model. More complex the model, less simple it is. There is no universal definition for the complexity of a model used in machine learning. However here are a few typical ways of looking the complexity of a model.

1. Number of parameters required to specify the model completely. For example in a simple linear regression for the response attribute y on the explanatory attributes x_1, x_2, x_3 the model $y = ax_1 + bx_2$ is 'simpler' than the model $y = ax_1 + bx_2 + cx_3$ — the latter requires 3 parameters compared to the 2 required for the first model.
2. The *degree* of the function, if it is a *polynomial*. Considering regression again, the model $y = ax_1^2 + bx_2^3$ would be a more complex model because it is a polynomial of degree 3.
3. Size of the best-possible representation of the model. For instance the number of bits in a binary encoding of the model. For instance more complex (messy, too many bits of precision, large numbers, etc.) the coefficients in the model, more complex it is. For example the expression $(0.552984567 * x^2 + 932.4710001276)$ could be considered to be more 'complex' than say $(2x + 3x^2 + 1)$, though the latter has more terms in it.
4. The depth or size of a decision tree.

Intuitively more complex the model, more 'assumptions' it entails. Occam's Razor is therefore a simple thumbrule — given two models that show similar 'performance' in the finite training or test data, we should pick the one that makes fewer assumptions about the data that is yet to be seen. That essentially means we need to pick the 'simpler' of the two models. In general among the 'best performing' models on the available data, we pick the one that makes fewest assumptions, equivalently the simplest among them. There is a rather deep relationship between the complexity of a model and its usefulness in a learning context. We elaborate on this relationship below.

1. **Simpler models are usually more 'generic'** and are more widely applicable (are generalizable).
   - One who understands a few basic principles of a subject (simple model) well, is better equipped to solve any new unfamiliar problem than someone who has memorized an entire 'guidebook' with a number of solved examples (complex model). The latter student may be able to solve any problem extremely quickly as long as it looks similar to one of the solved problems in the guidebook. However given a new unfamiliar problem that doesn't fall neatly into any of the 'templates' in the guidebook, the second student would be hard pressed to solve it than the one who understands the basic concepts well and is able to work his/her way up from first principles.
   - A model that is able to accurately 'predict

Generalization Bounds

Statistical Learning Theory does establish bounds of the following kind for many of the models we will encounter, including regression: with probability at least $(1 - \delta)$

for any $\delta > 0$,

$$E_G(M) \leq E_T(M) + f\left(\frac{1}{n}, \frac{1}{\delta}, \mathscr{C}(M)\right)$$

Such bounds are known as *PAC* (**Probably Approximately Correct**) bounds. The basic idea is that data available for training is always finite while the domain on which the learnt model needs to be applied eventually is potentially infinite. So how does one know if what we have learnt from the finite set of training examples is good enough to be applicable in general across the domain from which the examples have been drawn? This question is not unlike any typical survey that involves the entire population (health surveys, election poll, etc.) — the survey samples just a small set of people but your predictions have to be for the entire population. A good model is one whose behaviour when applied to the training data is a reasonably accurate indicator of the expected behaviour of the model when used for prediction tasks outside the training set.

Here's a short explanation of the PAC bound that we stated above. The model M produced by the learning algorithm obviously depends on the training set given. Some training sets will result in a good model and others may be not so good. The general spirit of the PAC statement is that no matter what the training set is (remember the training set is really not in your control — you invariably have to work with what is available), under certain mild assumptions, the probability that the behaviour of the model on the training data is very different from the behaviour of the model when applied to the entire domain, is rather small. The 'behaviour' of the model is quantified by the *Empirical* or Training error $E_T(M)$ committed by the model on the training data, and the expected error $E_G(M)$ when it is applied across the domain. The statement above is a claim that with a high probability, $E_T(M)$ (what is observed) differs from $E_G(M)$ (what can be expected from the model used for predictions outside the training set) by at most a small quantity. The error bound $f\left(\frac{1}{n}, \frac{1}{\delta}, \mathscr{C}(M)\right)$ on the right hand side is a function that increases with (i) decrease in size of the training sample n, (ii) decrease in the confidence level δ, and (iii) increase in the complexity of the model $\mathscr{C}(M)$. So for achieving the same level of confidence (probability bound) one would require a higher number of training samples for more complex models. Conversely given a training set, the generalizability of the model is much more for simpler models than it is for the more complex ones. Note that the probability in the bound is taken over all possible training datasets of size n.

These bounds form the theoretical basis for the usefulness of many of the well known classical learning algorithms such as Logistic Regression, Linear Regression and SVM.

2. **Simpler models require fewer training samples** for effective training than the more complex ones and are consequently easier to train. In machine learning jargon, the **sample complexity** is lower for simpler models.

3. **Simpler models are more robust** — they are not as sensitive to the specifics of the training data set as their more complex counterparts are. Clearly we are learning a 'concept' using a model and not really the training data itself. So ideally the model must be immune to the specifics of the training data provided and rather somehow pick out the essential characteristics of the phenomenon that is invariant across any training data set for the

problem. So it is generally better for a model to be not too sensitive to the specifics of the data set on which it has been trained. Complex models tend to change wildly with changes in the training data set. Again using the machine learning jargon **simple models have low variance, high bias and complex models have low bias, high variance**. Here 'variance' refers to the variance in the model and 'bias' is the deviation from the expected, ideal behaviour. This phenomenon is often referred to as the **bias-variance tradeoff**.

4. **Simpler models make more errors in the training set** — that's the price one pays for greater predictability. **Complex models lead to overfitting** — they work very well for the training samples, fail miserably when applied to other test samples.

3.1 Overfitting

Overfitting is a phenomenon where a model becomes way too complex than what is warranted for the task at hand and as a result suffers from bad generalization properties. Let's consider a couple of examples to illustrate this.

1. A trivial 'model' that would have 'learnt' perfectly from any given dataset is one that just memorizes the entire dataset. Clearly the error committed by such a model on the dataset which it was 'trained' on would be zero. However it is clear that such a model can do nothing other than answer questions (though perfectly) about the dataset it was trained on. The 'model' will effectively be incapable of doing anything better than a random guess on test data point that is outside the training dataset. This would be an extreme example of the overfitting phenomenon — perfect on the training data but unacceptably large error on test data.

2. Consider a set of points of the form $(x,\ 2x+55+\epsilon)$ for various values of x, where ϵ is a Gaussian noise with mean zero and variance 1. We can carry out a linear regression on this dataset, but the regression line is unlikely to pass through any of the points exactly. We can also compute a polynomial fit on this dataset that will pass through almost every one of the given points perfectly. See Figure 1 where a straight line fit and a polynomial fit are both used to interpolate for a wide-range of values of x. It is obvious from the figure that the polynomial fit is wildly off the mark for x values that are not part of the ones used for the 'training' (these points are marked with a small circle in the figure). The polynomial model badly *overfits* the given data whereas the straight line fit is likely to behave predictably for a large of values of x and y that follow a similar pattern.

Code to Generate Figure 1

```
n <- 20
noise <- ts(rnorm(n+1, mean = 0, sd = 1))
x <- seq(n,2*n); names(x) <- x
y <- 2*x + 55 + noise; names(y) <- y
df <- data.frame(x=x, y=y)
plot(df$x, df$y, type='b', xlab='x', ylab='y',
     main='Straight Line vs Polynomial Regression')
```

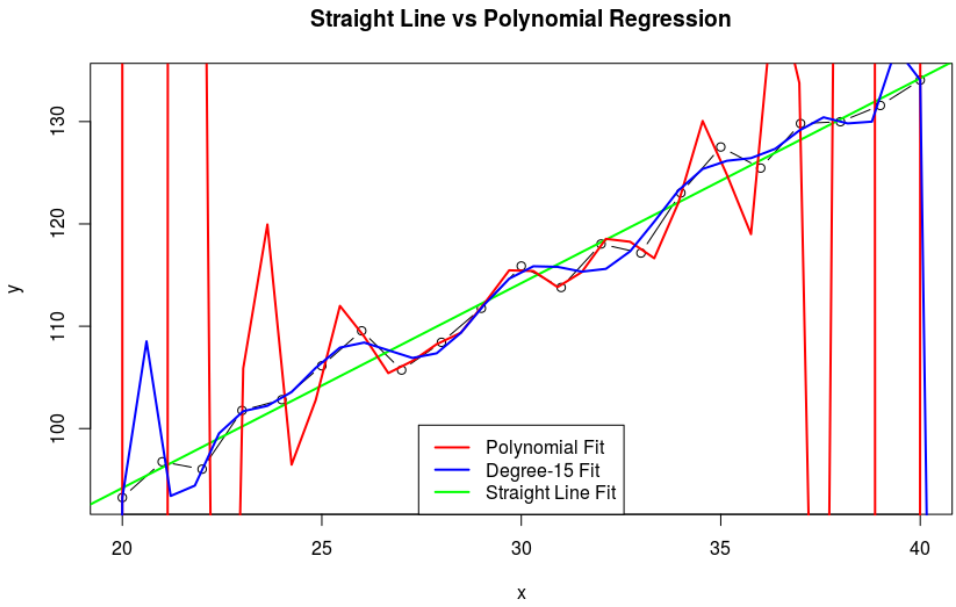

Figure 1: Polynomial vs Straight Line Fit

```
fitseq <- seq(0, 3*n, length.out=100)
lr <- lm(y ~ x, data=df)
lines(fitseq, predict(lr, data.frame(x=fitseq)), col='green', lwd=2)

polr <- lm(y ~ poly(x, n), data=df)
lines(fitseq, predict(polr, data.frame(x=fitseq)), col='red', lwd=2)

legend("bottom", c("Polynomial Fit","Straight Line Fit"),
       col=c("red","green"), lwd=2)
```

3.2 Bias-Variance Tradeoff

Consider the example of a model memorizing the entire training dataset, that we considered earlier. Clearly this 'model' will need to change for every little change in the dataset. The model is therefore very unstable and extremely sensitive to any changes in the training data. However 'simpler' model that abstracts out some pattern followed by the data points given is unlikely to change wildly even if more points are added, points are removed or if some of the given points are perturbed a little. The 'variance' of a model is the variance in its output on some test data with respect to changes in the training dataset. In other words *variance* here refers to the degree of changes in the model itself with respect to changes in the training data.

Bias quantifies how accurate is the model likely to be on future (test) data. Complex models, assuming you have enough training data available, can do a very accurate job of prediction. Models that are too näive, are very likely to do badly. Again a trivial example of an utterly näive

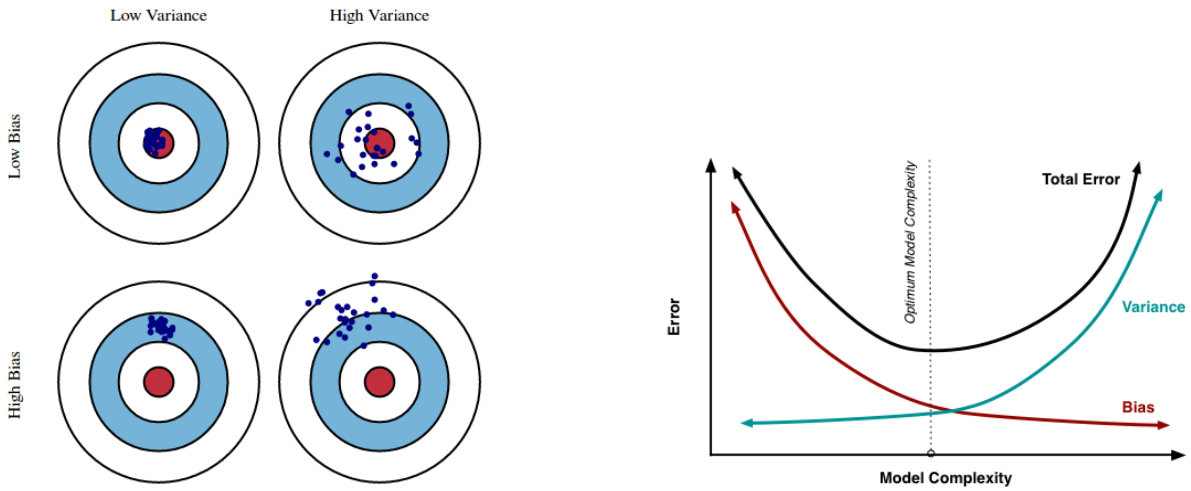

Figure 2: (Left) Illustration of Bias and Variance; (Right) Bias-Variance Tradeoff

model is one that gives the same answer to all test inputs — makes no discrimination whatsoever, and just repeats the same answer no matter what the question is!! This model will indeed have a very large bias — its expected error across test inputs is very high.

The left-hand side of Figure 2 illustrates Bias and Variance using a target shooting analogy. A 'model' whose shots are clustered together is one that has a small variance and one whose shots are close to the "bull's eye" has a small bias. A 'consistent' shooter will have a small variance and a 'good' shooter will have a small bias. So in other words variance is about consistency and bias is about accuracy.

The right-hand side of Figure 2 illustrates the typical tradeoff between Bias and Variance — low complexity models have high bias and low variance and high complexity models have low bias but high variance. Notice that our näive 'constant' model will never change and hence its variance is zero, though the bias is high. The 'best' model for a task is one that balances both — achieves reasonable degree of predictability (low variance) without compromising too much on the accuracy (bias).

3.3 Regularization

Regularization is the process used in machine learning to deliberately simplify models. Through regularization the algorithm designer tries to strike the delicate balance between keeping the model simple yet not making it too näive to be of any use. For the data analyst, this translates into the question "How much error am I willing to tolerate during training, to gain generalizability?"

The simplification can happen at several levels — choice of simpler functions, keeping the number of model parameters small, if the family is that of polynomials then keeping the degree of the polynomial low, etc. These are done by the designer and is typically not referred to as regularization. Regularization is the simplification done by the training algorithm to control

the model complexity. Some typical regularization steps for various classes of ML algorithms include:

1. For Regression this involves adding a regularization term to the cost that adds up the absolute values or the squares of the parameters of the model.
2. For decision trees this could mean 'pruning' the tree to control its depth and/or size.
3. For neural networks a common strategy is to include a *drop out* — dropping a few neurons and/or weights at random.

We will explore the specifics of these regularization strategies when we cover these algorithms in detail later in the course.

4 Model Evaluation

The discussion has so far been on the choice of a class of models (quadratic, etc.); the learning algorithm (like say linear regression) will then go ahead and find the best among this class of models to fit the given data. In this section we will discuss how we evaluate a specific model. Remember that evaluating how a model performs on the data it has been trained on is relatively straightforward. What we need to estimate however, to assess the usefulness of the model, is to evaluate its performance on data that it hasn't seen yet. More often than not, we will need to compare two models and finally choose one, rather than evaluating how good a model is in absolute terms.

We will first look at a couple of generic strategies for making best use of available data in order to arrive at a model that will generalize reasonably well. Of course in a situation where the available data is in abundance, this is hardly a concern. However when data limited, which is often the case, notice that we are dealing with a fundamental dilemma — we need the training set to be as large as possible (small training set leaves us with the danger of missing out on crucial pieces of information required for the learning) yet we need a way to estimate the reliability of the model in test scenarios not covered by the training data. In both the methodologies — *Hold-Out Strategy* and *Cross Validation* — the basic idea is the same: to keep aside some data that will not in any way influence the model building. The part of the data that is kept aside is then used as a 'proxy' for the unknown (as far as the model we have built is concerned) test data on which we want to estimate the performance of the model.

We also look at *hyperparameters* in this context. Refer Figure 3. A typical Machine Learning scenario is where we have a data source (possibly implicit) and an underlying system that we are trying to mimic. For example the data source could be an email repository or an email server. The 'system' is a human gate-keeper who is trying to segregate spam emails from the rest. The machine learning model we are trying to build is essentially trying to mimic a human 'system' of discriminating between spam emails and the rest. There is a *Learning Algorithm* that takes samples from the data source (this will constitute the training data) and the expected responses from the 'system' and comes up with a *model* that behaves a lot like the system we are trying to mimic. The model will often be used to predict what the system would have presumably

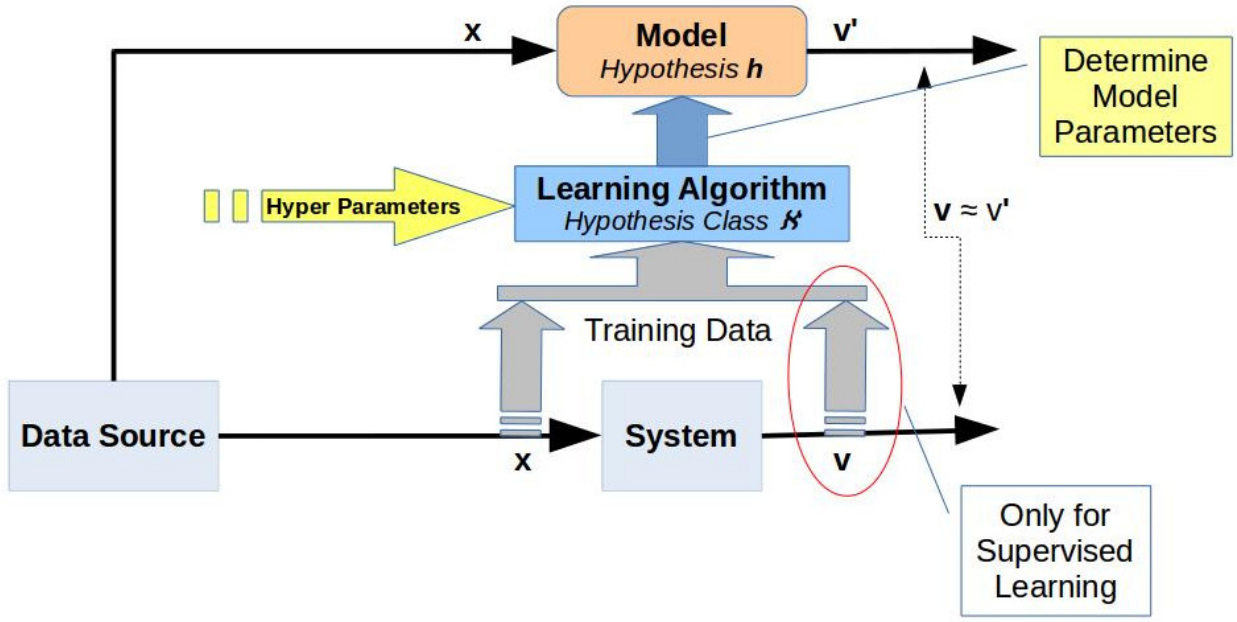

Figure 3: Machine Learning — A Broad Framework

done on any new test input that it is given. In the figure, the inputs are denoted by the vector $\boldsymbol{x}$, the expected 'system' output as $\boldsymbol{v}$ and model output as $\boldsymbol{v}'$. The learning algorithm works with a *hypothesis class* — each learning algorithm searches for the best possible model from a fixed class of models specific to the algorithm. For example standard linear regression finds the best 'straight line' to fit the data, a decision tree algorithm finds the best decision tree to explain the the dataset, a logistic regression model with a single explanatory variable looks for all models of the form $1/\left(1+e^{-(b+a_1 x)}\right)$. The output of the model is a specific hypothesis or model. For the above three examples, for a straight line regression fit, the model is one specific straight line (with values for the coefficients) that fits the model best, for the logistic regression model we find specific values of a, b for the best fit, etc. *Hyperparameters* govern the way the learning algorithm produces the model.

4.1 Hold-Out Strategy

4.2 Cross Validation

4.3 Model Evaluation for Classification

4.4 Model Evaluation for Regression

5 Appendix: Legend and Conventions, Notation

There are several parts of the test that have been highlighted (in boxes). The document uses three kinds of boxes.

Example 0: Title

One of the running examples in the document. The reader must be able to reproduce the observations in these examples with the corresponding R code provided in Section **??**.

Title

Good to remember stuff. These boxes typically highlight the key take-aways from this document.

Title

Things that are good to be aware of. This is primarily targeted at the reader who wishes to explore further and is curious to know the underlying (sometimes rather profound) connections with other fields, alternative interpretations, proofs of some statements, etc. However one could safely skip these boxes completely if you are only interested in the primary content/message of this document.

Title

`Code snippets. Implementation Tips. Do's and Dont's.`

$\boldsymbol{x}$	A *vector* $\boldsymbol{x}$
x	A scalar quantity x
term	Refer to the glossary as a quick reference for 'term'. These are typically prerequisite concepts that the reader is expected to be familiar with.
`code`	Code snippets, actual names of library functions, etc.