Daniel Nee

MACHINE LEARNING

COMMON PITFALLS IN MACHINE LEARNING

JANUARY 6, 2015 | DN | 7 COMMENTS

Over the past few years I have worked on numerous different machine learning problems. Along the way I have fallen foul of many sometimes subtle and sometimes not so subtle pitfalls when building models. Falling into these pitfalls will often mean when you think you have a great model, actually in real-life it performs terribly. If your aim is that business decisions are being made based on your models, you want them to be right!

I hope to convey these pitfalls to you and offer advice on avoiding and detecting them. This is by no means an exhaustive list and I would welcome comments on other common pitfalls I have not covered.

In particular, I have not tried to cover pitfalls you might come across when trying to build production machine learning systems. This article is focused more on the prototyping/model building stage.

Finally, while some of this is based on my own experience, I have also drawn upon John Langford's Clever Methods of Overfitting, Ben Hamner's talk on Machine Learning Gremlins and Kaufman et al. Leakage in Data Mining. All are worth looking at.

Traditional Overfitting

Traditional overfitting is where you fit an overly complicated model to the trained dataset. For example, by allowing it to have too many free parameters compared to the number of training points.

To detect this, you should always be using a test set (preferably you are using cross validation). Plot your train and test performance and compare the two. You should expect to see a graph like figure 1. As your model complexity increases, your train error goes to zero. However, your test error follows an elbow shape, it improves to a point then gets worse again.

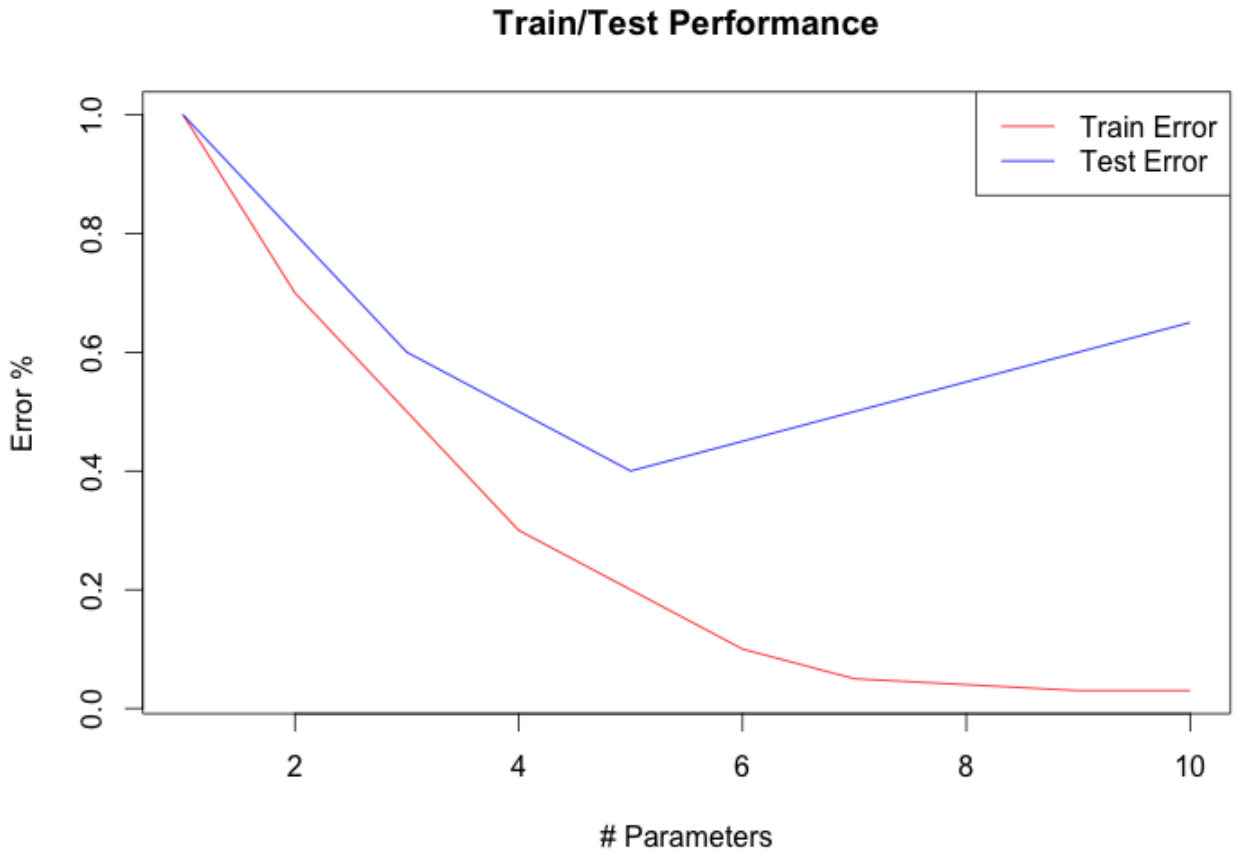


Figure 1. Train/Test Performance when varying number of parameters in the model

Potential ways to avoid traditional overfitting:

- Obtain more training data
- Use a simpler predictor function
- Add some form of regularisation, early stopping, pruning, dropout, etc. to the model
- Integrate over many predictors

More training data

In general, the more training data you have the better. But it may be expensive to obtain more.

Before going down this route it is worth seeing if more train data will actually help. The usual way to do this is to plot a learning curve, how the training sample size affects your error:

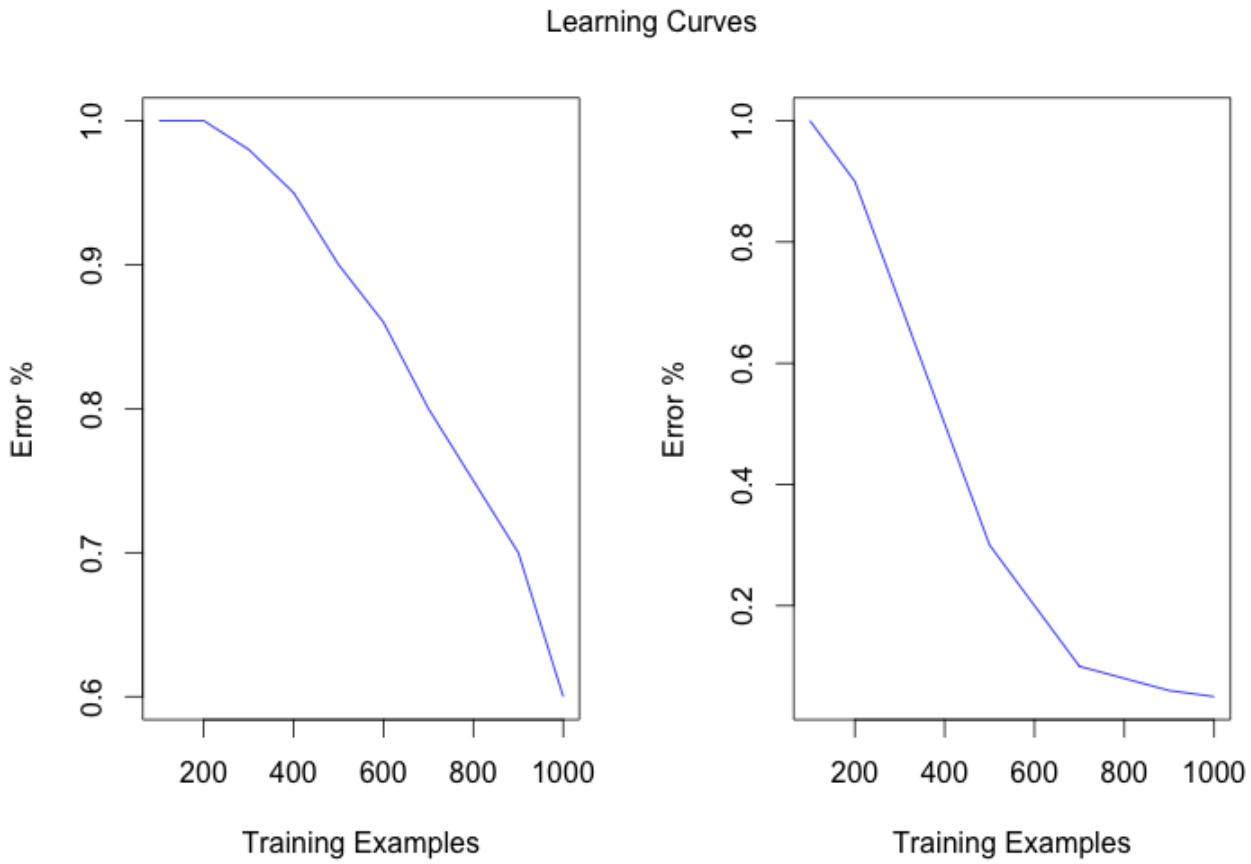


Figure 2. Two example learning curves

In the left-hand graph, the gradient of the line at our maximum training size is still very steep. Clearly here more training data will help.

In the right-hand graph, we have started to reach a plateau, more training data is not going to help too much.

Simpler Predictor Function

Two ways you could use a simpler predictor:

- Use a more restricted model e.g. logistic regression instead of a neural network.
- Use your test/train graph (figure 1) to find an appropriate level of model complexity.

Regularisation

Many techniques have been developed to penalise models that are overly complicated (Lasso, Ridge, Dropout, etc.). Usually this involve setting some form of hyper-parameter. One danger here is that you tune the hyper-parameter to fit the test data, which we will discuss in Parameter Tweak Overfitting.

Integrate over many predictors

In Bayesian Inference, to predict a new data point x:

$$p(x|\mathbf{D}, \alpha) = \int_\theta p(x|\theta, \alpha)\, p(\theta|\mathbf{D}, \alpha)$$

where α is some hyper-parameters, $\mathbf{D}$ is our training data and θ are the model parameters. Essentially we integrate out the parameters, weighting each one by how likely they are given the data.

Parameter Tweak Overfitting

This is probably the type of overfitting I see most commonly. Say you use cross validation to produce the plot in figure 1. Based on the plot you decide the 5 parameters is optimal and you state your generalisation error to be 40%.

However, you have essentially tuned your parameters to the test data set. Even if you use cross-validation, there is some level of tuning happening. This means your true generalisation error is not 40%.

Chapter 7 of the Elements of Statistical Learning discuss this in more detail. To get a reliable estimate of generalisation error, you need to put the parameter selection, model building inside, etc. in an inner loop. Then run a outer loop of cross validation to estimate the generalisation error:

```
// Outer Cross Validation Loop
for (i in 1:k) {
  Train = X[folds != i,]
  Test = X[folds == i,]

  // Inner Cross Validation Loop
  for (j in 1:k) {
    InnerTrain = Train[folds != j,]
    InnerTest = Train[folds == j,]

    // Train Model
    // Try multiple parameter settings
    // Predict on InnerTest
  }
  // Choose best parameters

  // Train model using best parameters from inner loop
  // Test performance on Test
}
```

For instance, in your inner loop you may fit 10 models, each using 5X CV. 50 models in total. From this inner loop, you pick the best model. In the outer loop, you run 5X CV, using optimal model from the inner loop and the test data to estimate your generalisation error.

Choice of measure

You should use whatever measure is canonical to your problem or makes the most business sense (Accuracy, AUC, GINI, F1, etc.). For instance, in credit default prediction, GINI is the widely accepted measurement.

It is good to practice to measure multiple performance statistics when validating your model, but you need to focus on one for measuring improvement.

One pitfall I see all the time is using accuracy for very imbalanced problems. Telling me that you achieved an accuracy of 95%, when the prior is 95% means that you have achieved random performance. You must use a measure that suits your problem.

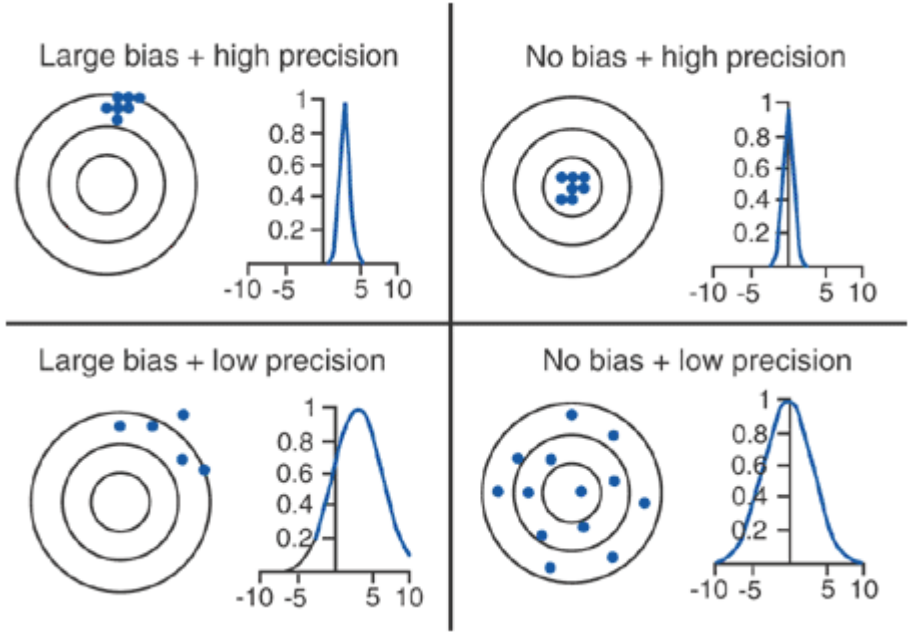
Resampling Bias and Variance

When measuring your test performance you will likely use some form of resampling to create multiple test sets:

- K-fold cross validation
- Repeated k-fold cross validation
- Leave One Out Cross Validation (LOOCV)
- Leave Group Out Cross Validation (LGOCV)
- Bootstrap

Ideally we want to use a method that achieves low bias and low variance. By that I mean:

- Low bias – The generalisation error produced is close to the true generalisation error.
- Low variance (high precision) – The spread of the re-sampled test results is small



In general, the more folds you use the lower the bias, but the higher the variance.

You will need to pick a resampling method to suit your problem. With large training sets, k-fold cross validation with a k of 5 may suit. With small training sets, you will need a larger k.

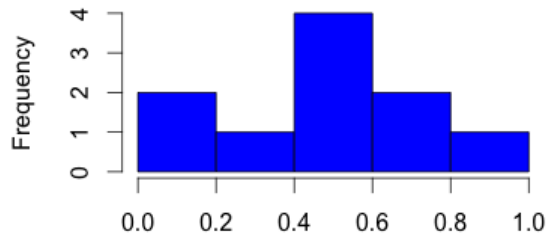
Max Kuhn has done a fantastic empirical analysis of the different methods in relation to bias and variance (part I, part II). In general repeated 10-fold cross validation seems to be quite stable across most problems in terms of bias and variance.

Bad Statistics

Once you have computed the results of your cross validation, you will have a series of measurement of your test performance. The usual thing to do here is to take the mean and report this as your performance.

However just reporting the mean may hide the fact there is significant variation between the measurements.

It is useful to plot the distribution of the performance estimates. This will give you and idea of how much variation there is:



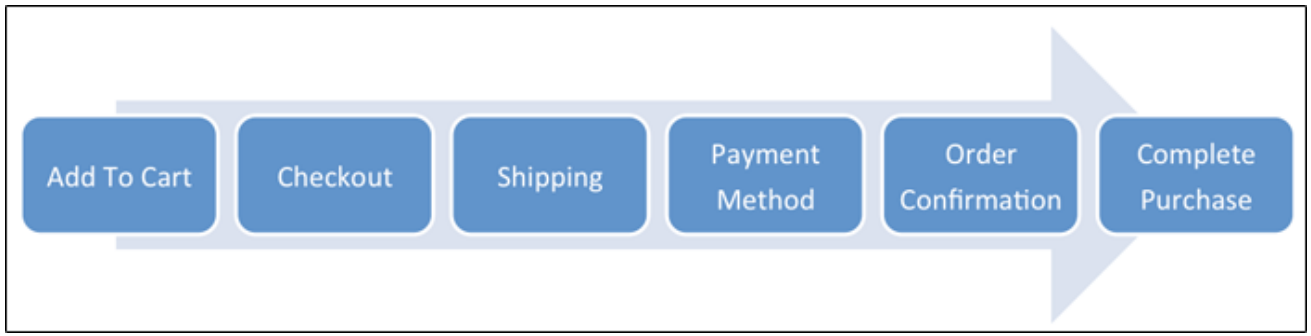
If you need to boil it down to a single number, computing the standard deviation (or variance) can be useful. However you often see performance quoted like: 80% +/- 2% where the 80% is the mean and the 2% is the standard deviation. Personally I dislike this way of reporting as it suggests the true performance is between 78%-82%. I would quote them separately; mean and standard deviation.

If you want to write 80% +/- 2%, you would need to compute bounds on the estimate.

Information Leakage

Information Leakage occurs when data on your training label leaks into your features. Potentially it is even more subtle, where irrelvant features appear as highly predictive, just because you have some sort of bias in the data you collected for training.

As an example, imagine you are an eCommerce website and want to predict converters from the way people visit your website. You build features based on the raw URLs the users visit, but take special care to remove the URLs of the conversion page (e.g. Complete Purchase). You split your users into converters (those reaching the conversion page) and non-converters. However, there will be URLs immediately before the conversion page (checkout page, etc.) that will be present in all the converters and almost none of the non-converters. Your model will end up putting an extremely high weight on these features and running your cross-validation will give your model a very high accuracy. What needed to be done here was remove any URLs that always occur immediately before the conversion page.



Feature Selection Leakage

Another example I see regularly is applying a feature selection method that looks at the data label (Mutual Information for instance) on all of the dataset. Once you select your features, you build the model and use cross-validation to measure your performance. However your feature selection has already looked at all the data and selected the best features. In this sense your choice of features leaks information about the data label. Instead you should have performed inner-loop cross validation discussed previously.

Detection

Often it can be difficult to spot these sorts of information leakage without domain knowledge of the the problem (e.g. eCommerce, medicine, etc.).

The best advice I can suggest to avoid this is to look at the top features that are selected by your model (say top 20, 50). Do they make some sort of intuitive sense? If not, potentially you need to look into them further to identify if their is some information leakage occurring.

Label Randomisation

A nice method to help with the feature selection leakage is to completely randomly shuffle your training labels right at the start of your data processing pipeline.

Once you get to cross validation, if your model says it has some sort of signal (an AUC > 0.5 for instance), you have probably leaked information somewhere along the line.

Human-loop overfitting

This is a bit of a subtle one. Essentially when picking what parameters to use, what features to include, it should all be done by your program. You should be able to run it end-to-end to perform all the modelling and performance estimates.

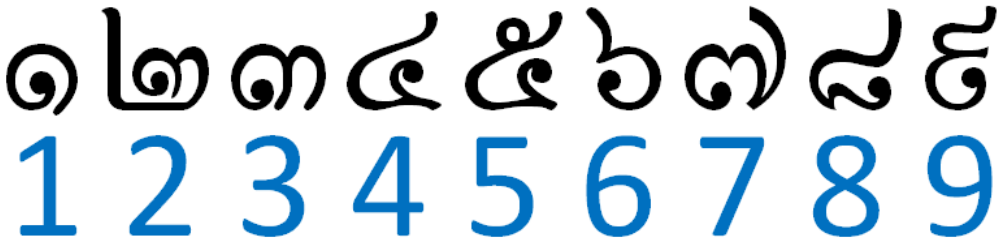
It is ok to be a bit more “hands-on” initially when exploring different ideas. However your final “production” model should remove the human element as much as possible. You shouldn’t be hard-coding parameter settings.

I have also seen this occurring when particular examples are hard to predict and the modeller decides to just exclude these. Obviously this should not be done.

Non-Stationary Distributions

Does your training data contain all the possible cases? Or could it be biased? Could the potential labels change in the future?

Say for example you build a handwritten digit classifier trained on the the MNIST database. You can classify between the numbers 0-9. Then someone gives you handwritten digits in Thai:



How will your classifier behave? Possibly you need to obtain handwritten digits for other languages, or have an other category that could incorporate non Western Arabic numerals.

Sampling

Potentially, your training data may have gone through some sort of sampling procedure before you were provided it. One significant danger here is that this was sampling with replacement and you end up with repeated data points in both the train and test set. This will cause your performance to be over-estimated.

If you have a unique ID for each row, check these are not repeated. In general check how many rows are repeated, you may expect some, but is it more frequent than expected?

Summary

My one key bit of advice when building machine learning models is:

If it seems to good to be true, it probably is.

I can't stress this enough, to be a really good at building machine learning models you should be naturally sceptical. If your AUC suddenly increases by 20 points or you accuracy becomes 100%. You should stop and really look at what you have done. Have you fallen trap of one of the pitfalls described here?

When building your models, it is always a good idea to try the following:

- Plot learning curves to see if you need more data.
- Use test/train error graphs to see if your model is too complicated.
- Ensure you are running inner-loop cross validation if you are tweaking parameters.
- Use repeated k-fold cross validation if possible.
- Check your top features – Do they make sense?
- Perform the random label test.
- Check for repeated unique IDs or repeated rows.

CROSS VALIDATION INFORMATION LEAKAGE MACHINE LEARNING OVERFITTING PITFALLS

7 THOUGHTS ON "COMMON PITFALLS IN MACHINE LEARNING"

Terje Kristesen

JANUARY 10, 2015 AT 9:33 AM

I think this is very improtant aspect that often are not considered.
Thank you.

Terje

Richard R. Liu

JANUARY 10, 2015 AT 11:50 AM

Good points. I would add: Understand the model on which the ML algorithm that you are using is based. Do the assumptions apply to your problem?

A good example is Random Forests. A common misunderstanding — especially of those who don't take the time to understand what RF is doing — is that it is prone to overfitting. This myth might derive from the knowledge that decision trees do, that's why they are pruned, whereas in RF decision trees are not pruned. It's the sampling by replacement that introduces some randomness into RF and prevents overfitting. When used to predict continuous outputs (regression) RF has a tendency to overfit, but not when used for classification.

Sam Savage

JANUARY 10, 2015 AT 11:59 AM

Great article Dan, thorough, practical and wide scope. I particularly like the point about using several measures of performance rather than a single number, and try to find ones that map to business needs. If one can actually deduce profitability from measures like precision, or costs from recall, then the measures suddenly become very powerful. Matthews correlation coefficient, F1 score & GINI might be fancy ways to turn collections of points into a single easy to digest number, but if that number cannot be mapped to the domain, they are pointless.

I think this and the points about over-fitting have a common root cause, that is a lack of philosophical and foundational understanding. It's easy to use a bunch of libraries to try out a bunch of models and produce a bunch of measures, where good Data Scientists add value is being able to *understand* what these things actually *mean*. I am shocked at how few "Data Scientists" I interview really understand, even the basics, of what they are doing. Their CV will rattle off a huge list of algorithms, but my first question is always "What is probability? Give me the definition." Many are stumped either by this or other seemingly basic questions ("what does standard deviation actually mean?", "what distribution assumptions does model X make", etc, etc).

As you point out looking at a curve is often more valuable than turning that curve into a single number. One must know exactly what those numbers really *mean* with respect to PROBABILITY – the fundamental building block of all machine learning and uncertain

reasoning.

All models, measures and statistics follow a simple recipe; probability theory plus some assumptions about the data to make the problem tractable. The less assumptions the closer one gets to pure Bayesian nets & Information Theory and often further from tractability.

So my take away addition is this:

For any model, measure, or statistic, if you cannot formally define it in terms of PROBABILITY and it's assumptions, then do not use it since business actions based off incomplete understanding and false assumptions are very dangerous indeed.

Pingback: Condensed News | Data Analytics & R

Tim Mazumdar

MARCH 18, 2015 AT 1:15 AM

Overfitting takes many forms , the way to overfit is in many cases reflected as a higher order polynomial.

Overfitting can introduce variations in a decision surface . A lower order decision surface such as a line will not have variations. Over-fitted higher order polynomials will make a
decision surface have curves.

Pingback: Comprehensive list of data science resources [updated May 6, 2016] – WebProfIT Consulting

---

Pingback: Comprehensive list of data science resources | causality.io | causality.io