CS 760: Machine Learning Spring 2022

Topic 7: Cross-Validation

Instructor: Daniel L. Pimentel-Alarcón

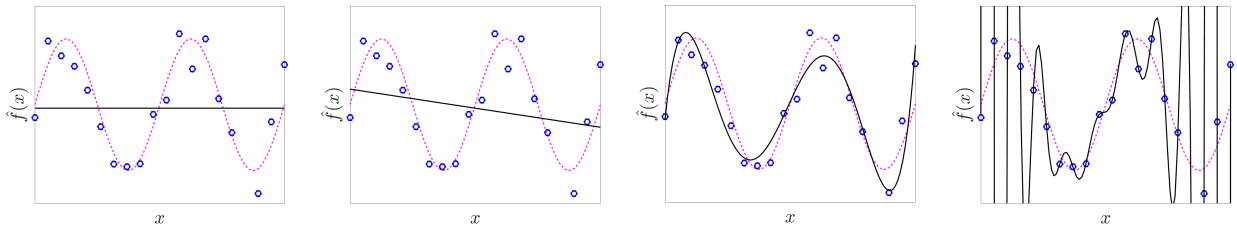
DO NOT POLLUTE! AVOID PRINTING, OR PRINT 2-SIDED MULTIPAGE.

7.1 Introduction

Common practice in Machine Learning involves splitting all available data in two sets, one for *training* and one for *testing*. As the name suggests, *training* data is used to learn the function $\hat{f}$ that best estimates the desired function $f^\star$, and *testing* data is used to evaluate the performance of our function $\hat{f}$ on unseen data, i.e., how well it generalizes, and whether it *overfits*:

We say that $\hat{f}$ *overfits* the training data if there is an alternative function f that has lower error than $\hat{f}$ on all the available data (training and testing), but higher error than $\hat{f}$ on the training data alone.

Intuitively, overfitting means you do too well on the training data, but poorly on the entire dataset, suggesting that $\hat{f}$ is only *memorizing* answers, rather than *learning* how to perform a task. This often happens when $\hat{f}$ has far too many parameters. For example, in the next figures, the function $f^\star$ that we are trying to learn is the sinusoidal function in magenta; the slightly noisy data that we observe are the blue circles; in black are four polynomial approximations $\hat{f}$ with different degrees (parameters):

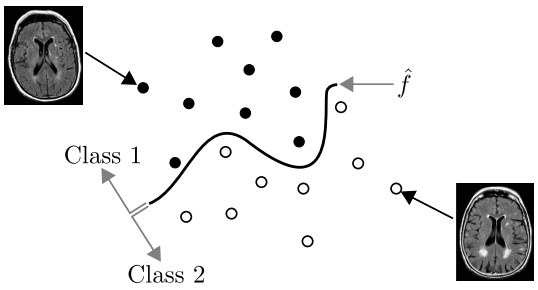


Degree = 0 Degree = 1 Degree = 5 Degree = 19

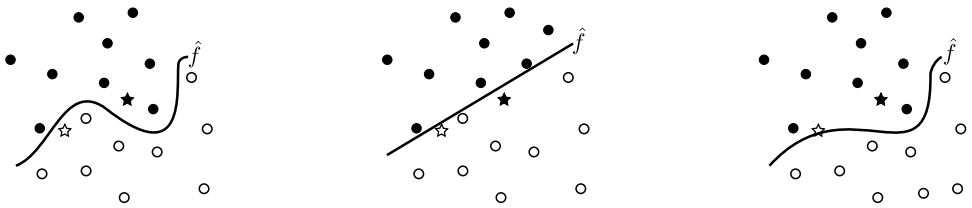
The degrees of freedom (parameters) of $\hat{f}$ clearly play a crucial role in overfitting. However, depending on the way we split samples into test and training, a method may also *appear* to be overfitting (or not) by mere chance. Cross-validation is a procedure to minimize this chance, and give some reliability to our conclusions of how a method will generalize to new data.

To build some intuition, suppose we want to use a machine learning algorithm (e.g., logistic regression, decision tree, neural network, etc.) to diagnose Alzheimers. To this end we will use a dataset containing of 90 images, evenly split among two classes: Healthy and Alzheimers. Following a standard approach, we use

30 images for training and 60 images for testing, randomly split between the two classes. After training, the algorithm will produce a classifier function $\hat{f}$ that essentially separates these two classes:



Then we can use our 30 testing images to assess our function's accuracy. However, depending on the training/testing split, we might get different classifier functions $\hat{f}$, and in turn, different results in the testing set. For example, here are three classifiers produced by training sets (circles) with just one different sample; notice the different classifications we would get for the same testing samples (stars):

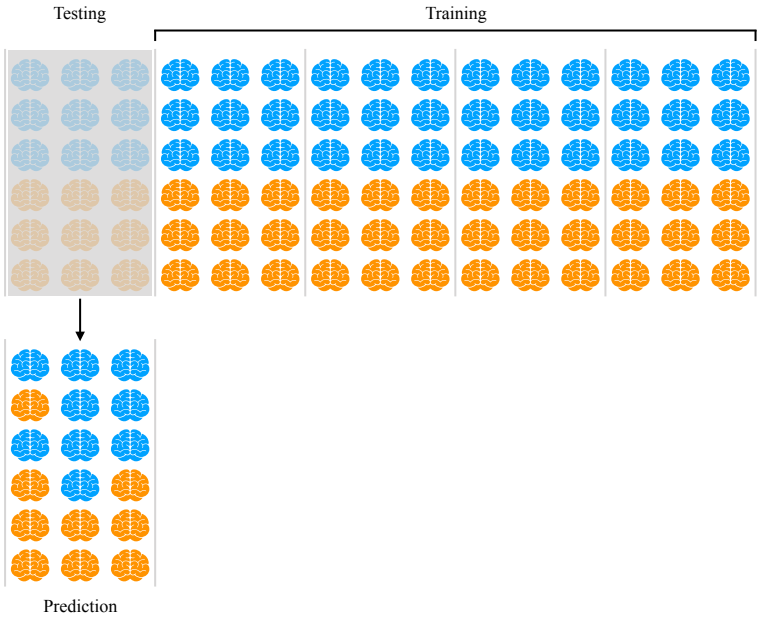


Hence, by mere chance in how we split the training and testing, we could conclude that our classifier generalizes very well (or very badly) to new data. In some cases we can analytically estimate the error that $\hat{f}$ will have on new data. For example, in linear regression, it can be shown that under mild assumptions, the expected mean squared error on new data will be about $\frac{N+D+1}{N-D-1}$ times the error on training data, where N denotes the number of samples and D denotes the number of features [1]. However, for other methods (like neural networks) this analysis can be quite daunting. In cases like these, cross-validation offers a generally applicable approach to estimating the performance of a method on new data, by running several trials with different testing/training splits.

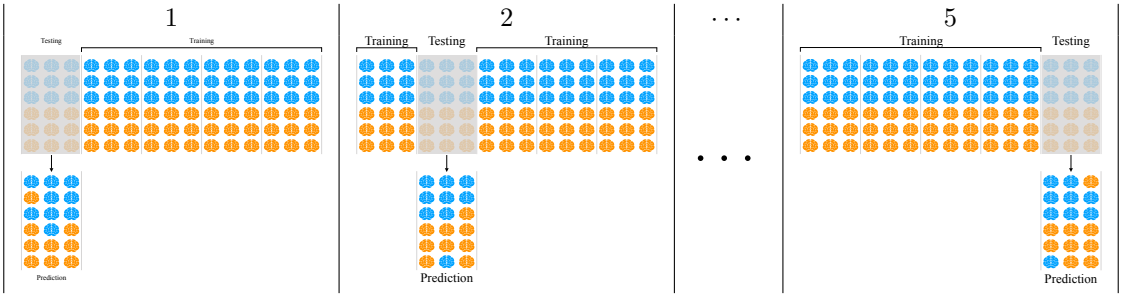
7.2 K-Fold Cross-Validation

The main idea of K-fold cross-validation is to split all data into K subsets, treat each one as testing, and average the results. More precisely, first we split each class into K even subsets, so that classes are balanced within each fold. Then each fold is formed by merging one different batch of each class. Typical values for K are 5 and 10. In our Alzheimers example, our K = 5 folds would look as follows:

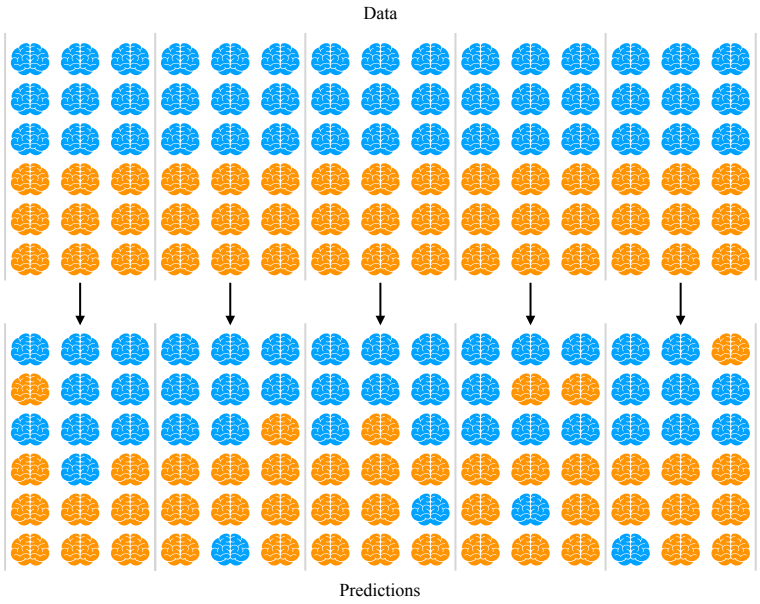
Then we train the classifier using all but one part of the data, and predict the classification of the left-out set:



Repeat the same for each $k = 1, \ldots, K$:



In the end we will have one prediction for each sample:



At this point we can compute accuracy as the fraction of correctly classified samples. In this example we have 11 misclassified points (79 correctly classified) out of 90, producing an accuracy of 88%.

7.3 Leave One Out Cross-Validation

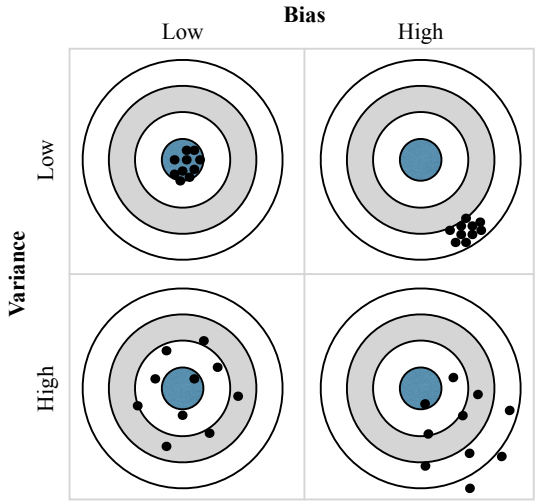
This is the same as K-fold cross-validation with K equal to the number of samples. In our figure above, we would predict the classification of the each image using the remaining 89 as training.

7.4 Holdout Method

Remember at the beginning we said we could randomly select 60 images for training and 30 for testing (keeping classes evenly distributed in each set)? Well, this method is called holdout. As mentioned before, you may get an unreliable result, and is suboptimal in the sense that you are not exploiting all your data (because you are keeping some for testing).

7.5 Bias and Variance

In general, we want the performance of our classifier to have low bias and low variance across trials (folds):



7.6 Imbalanced Data

So far we have been assuming data is balanced across classes. That is, each class has about the same number of samples. If this is not the case, some classifiers can become *lazy*, and simply classify everything as the larger class. For example, imagine that out of your 90 samples, only 1 have Alzheimers. Then a classifier can simply assign everything as healthy, and still have 98.9% accuracy.

Hence it is often useful to measure the accuracy of each class. In the example above, we can compute how often we can correctly detect Alzheimers, and how often we can detect a healthy case:

Prediction \ Truth	Healthy	Alzheimers
Healthy	39	5
Alzheimers	6	40

There are several other ways to deal with imbalanced data, for example resampling with replacement the smallest group to generate similar sample sizes, or penalizing errors in the smallest group (for example, in the cost function of a neural network).

Remark 7.1. Notice that if classes are evenly distributed, a random classifier will generally obtain an accuracy of $1/C$, where C is the number of classes. If the accuracy of a method is below this, then something is probably wrong. Typical reasons are outliers or biased datasets.

7.7 Peeking Bias

Peeking bias refers to the bias induced when the test and training sets are not independent. For example, this will happen if:

- The MRI of certain subject is included in the training data, and another MRI of the same subject is included in the testing data.
- As part of your training you normalize (subtract mean and divide by standard deviation) across *all* data, including testing data.

To avoid this one must verify that training is done without any involvement of the testing data:

- Keep all observations contributed from the same subject together (either all on the training set or all on the testing set).
- Estimate normalization parameters from training data alone.
- Estimate model parameters from training data alone.
- Estimate tuning parameters from training data alone.

One way to verify whether you have peeking bias is to randomly reassign labels and repeat the entire procedure. If the accuracy is significantly different from a random classifier, chances are you have a peeking bias.

References

[1] L. Trippa, L. Waldron, C. Huttenhower, and G. Parmigiani, *Bayesian nonparametric cross-study validation of prediction methods*, The Annals of Applied Statistics, 2015.