Week 12: Cross-Validation

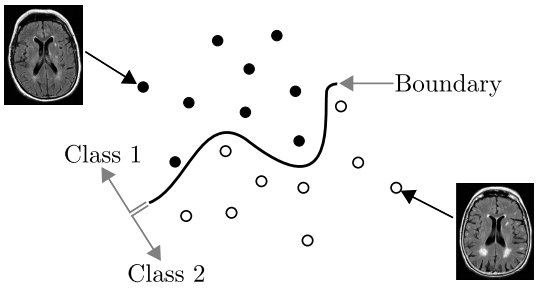
Instructor: Daniel L. Pimentel-Alarcón

GO GREEN. AVOID PRINTING, OR PRINT 2-SIDED MULTI-PAGE.

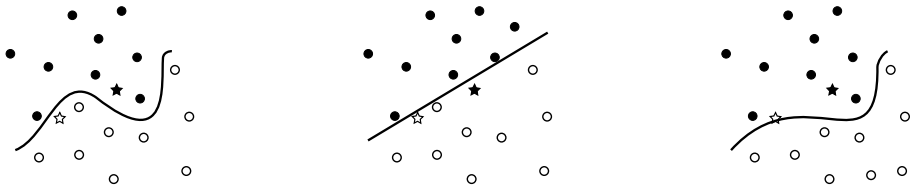
12.1 Introduction

Several image analysis methods, like the Hough transform, or the Sobel filter are data-agnostic. For example, the Hough transform of a certain image is produced by a deterministic procedure that does not depend on other images. However, other methods, like Deep Learning or Logistic regression, are data-dependent. For example, how a *test* image is classified by a Neural Network heavily depends on the classification of other *training* images. Depending on the choice of test and training samples, a method may *appear* to be working well (or badly) 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 Neural Network to diagnose Alzheimers. To this end we will use a dataset containing of 2400 images, evenly split among two classes: Healthy and Alzheimers. To this end, we can follow a standard approach: use 2000 images for training and 400 images for testing, split randomly from each class. After training, the Neural Network will produce a classifier/function/boundary that separates these two classes:



Then we can use our 400 testing images to assess our network's accuracy. However, depending on the training/testing split, we might get different classifiers, 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):

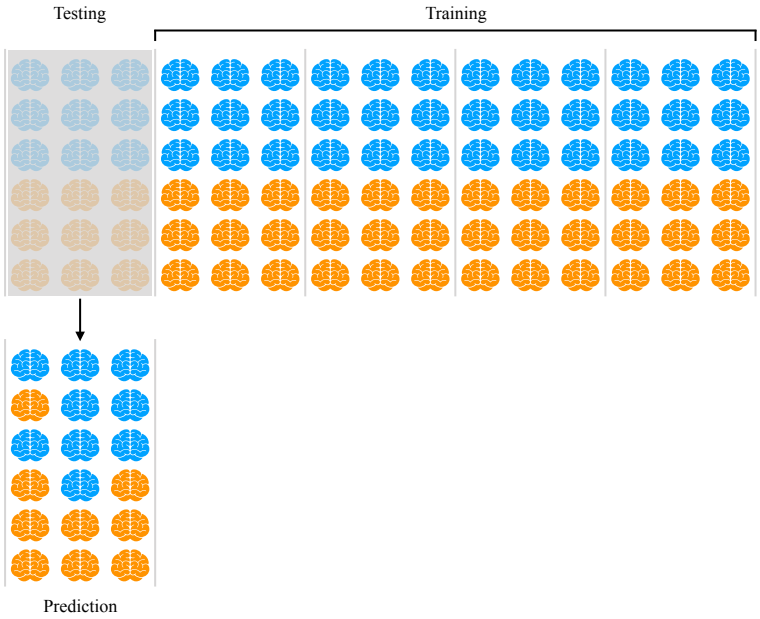


Hence, by mere chance in how we split the training and testing, we could conclude that our classifier generalizes very well (or very badly) to new data. Cross-validation mitigates this by running several trials with different testing/training splits.

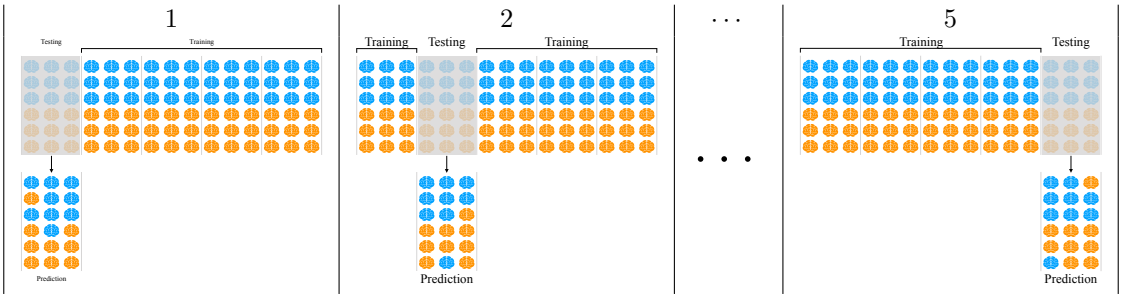
12.2 K-Fold Cross-Validation

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 all our data into K subsets of equal sizes, so that classes are balanced within each fold. This can be easily done by simply splitting each class into K even subsets. For example, here $K = 5$ (typical values for K are 5 and 10):

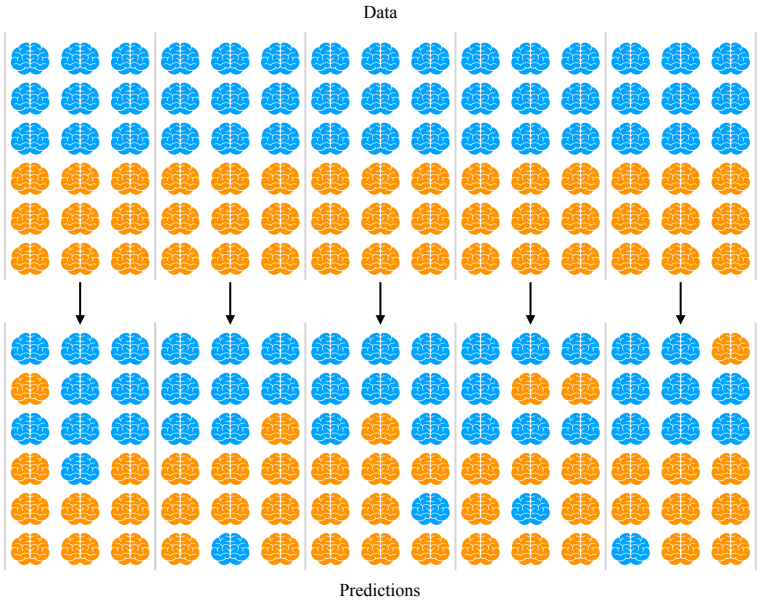
Train the classifier using all but one part of the data, and predict the classification of the left-out set:



Repeat the same for each k:



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%.

12.3 Leave One Out Cross-Validation

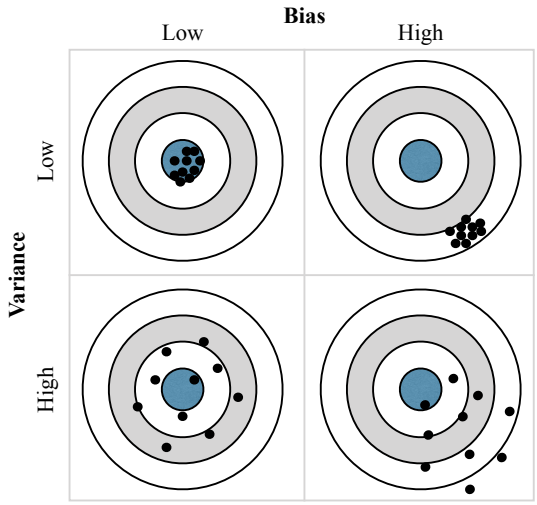
This is the same as K-fold cross-validation with K equal to the number of samples. In our example above, we would predict the classification of the each image using the remaining 89 as training.

12.4 Holdout Method

Remember at the beginning we said we could randomly select 2000 images for training and 400 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).

12.5 Bias and Variance

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



12.6 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 2400 samples, only 24 have Alzheimers. Then a classifier can simply assign everything as healthy, and still have 99% 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 12.1. 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.

12.7 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.