CS 383/613 – Machine Learning

Supervised Data Sets

Slides adapted from material created by E. Alpaydin
Prof. Mordohai, Prof. Greenstadt, Pattern Classification (2nd Ed.),
Pattern Recognition and Machine Learning

Objectives

- Evaluation
- Generalization/Overfitting
- Validation & Cross-Validation

Supervised Datasets

- When we talked about our data, and the different ML problems, we talked about *supervised* and *unsupervised* data.
- Each of these have the *observable data,* X.
- However, *supervised* data also comes with the *target values* as Y.
- Principle Component Analysis was an example of an *unsupervised* algorithm.
  - It didn’t need the target values to do its job.
- Conversely, feature selection via entropy, was an example of a *supervised* approach.
  - We needed to know the target class labels to compute the entropy.

Evaluating Supervised Datasets

- If we have target values, *evaluating* the quality of a machine learning is relatively easy.
- Let observation $\boldsymbol{x}$ have target value y.
- Then, a given machine learning algorithm can make a *prediction* for this observation as $\hat{y}$.
- How can we quantitatively determine how well this algorithm is doing at the task at hand?
- Depends on what we're doing!

Evaluation: SE and RMSE

- If we have target value y and its prediction $\hat{y}$ are *continuous valued*, then we could use the *squared error:*

$$SE = (y - \hat{y})^2$$

- Taken over an entire dataset (X, Y) we then have:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \hat{Y}_i\right)^2$$

- It is often numerically more logical to look at the square root of this, which we call the *root mean squared error (RMSE)*:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \hat{Y}_i\right)^2}$$

Evaluation: SMAPE

- A drawback of using RMSE as our metric, is that it is scale-dependent.
- An alternative metric we can use is the symmetric mean absolute percent error (SMAPE) defined as:

$$SMAPE = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|Y_i - \widehat{Y}_i\right|}{|Y_i| + \left|\widehat{Y}_i\right|}$$

Evaluation: Accuracy

- If our target values are *discretized* (as they are with classification), then it is natural to just evaluate as the percentage of times we are correct:

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N} Y_i == \hat{Y}_i$$

- This is referred to as *accuracy*

Class Priors

- For classification, we want to compare our accuracy against the *highest class prior.*
- A class prior is the probability of the class occurring, i.e
$$P(y=0), P(y=1), \dots, P(y=K-1)$$
- Each prior is computed as the percentage of the observations that came from that class:
$$P(y=k) = \frac{1}{N}\sum_{i=1}^{N} Y_i == k$$

Class Imbalance

- If one of the class's has a much higher prior than the others, we call this *imbalanced.*
- As a result, the algorithms will essentially learn to predict most things as the majority class, not helping much with the minority classes.
- The simplest ways to overcome this is to either *undersample* or *oversample.*
- **Undersampling:**
  - Grab some percentage of samples from the smallest class, and then grab that same number of samples (at random) from the other classes.
- **Oversampling**
  - Grab samples at random, *with replacement*, from all classes.
- **Oversampling w/ SMOTE** (**S**ynthetic **M**inority **O**versampling **Te**chnique)
  - *Synthetically* generate samples for under-represented classes by interpolating between a randomly selected sample and one of its randomly selected nearest neighbors.

Evaluation: Binary Classification Error Types

- Many times, we only have two possible outcomes.
- This is referred to as *binary classification.*
- There are many ways that we can refer to the two classes:
  - 0 vs 1
  - 1 vs 2
  - Positive vs Negative
  - Etc..
- Regardless, this type of problem often comes with additional types of evaluation…

Evaluation: Binary Classification Error Types

- If we refer to the two classes as the positive and negative class, then we have four different possibilities:
  - True positive = Hit
  - True negative = Correct rejection
  - False positive = False Alarm (Type 1 error)
  - False negative = Miss (Type 2 error)

	Predicted positive	Predicted negative	
Positive examples	**True positives**	**False negatives**	
Negative examples	**False positives**	**True negatives**	

Evaluating your Classifier

- From the four error types, we can establish some binary-classification-specific measurements:
- *Precision* – percentage of things that were classified as positive and actually were positive

$$Precision\ = \frac{TP}{TP + FP}$$

- *Recall* – the percentage of true positives (*sensitivity*) correctly identified

$$Recall\ = \frac{TP}{TP + FN}$$

- *f-measure* – The weighted harmonic mean of precision and recall

$$F_1 = \frac{2 * precision * recall}{precision + recall}$$

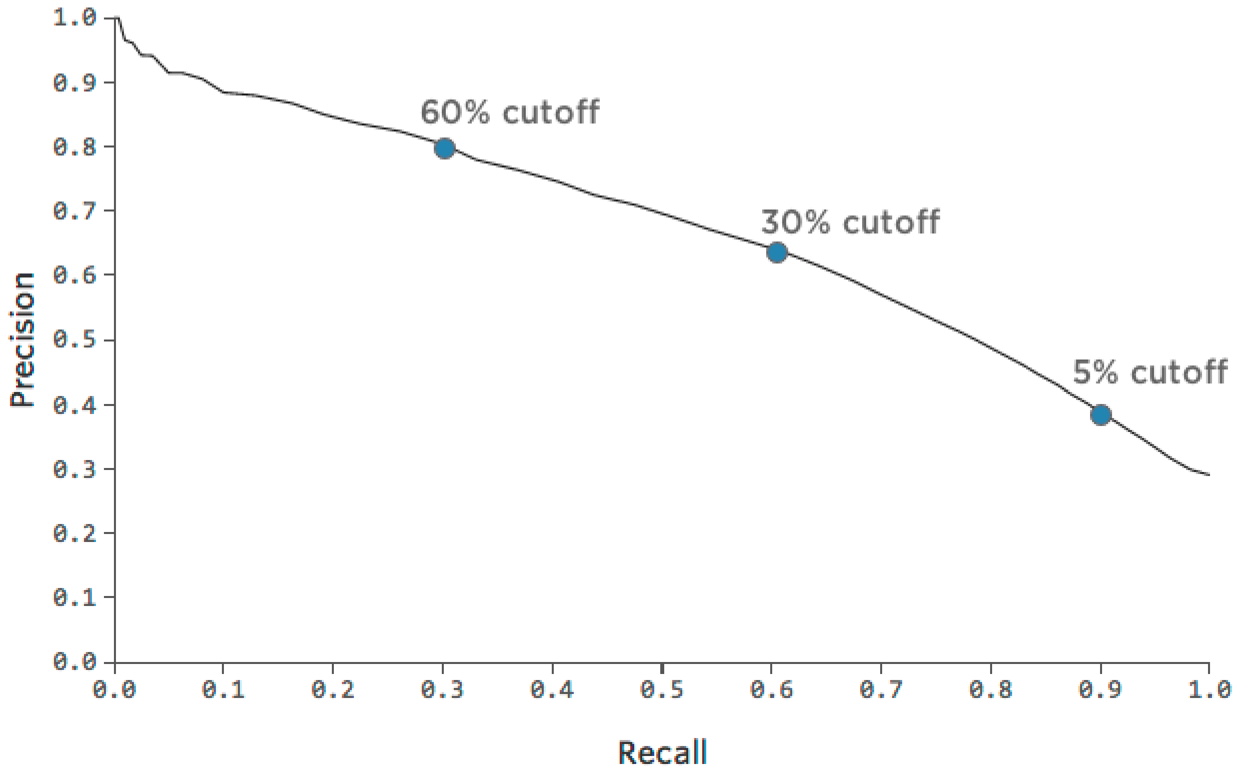
Using Class Likelihood

- Some classifiers don't just return what class an observation belongs to, but also return the probability of belonging to that class:
$$P(y = i|x)$$
- In these cases, we can use a *threshold* to determine what class an observation belongs to.
- For instance, for binary classification we can say:
$$\hat{y} = \begin{cases} Positive & P(y = Positive|x) > t \\ Negative & otherwise \end{cases}$$

Precision/Recall Tradeoff

- We can explore the effect of this threshold on the precision and recall values.
- The plot of precision vs recall as a function of the threshold creates something called a *precision-recall* curve (PR)

Precision/Recall Curve

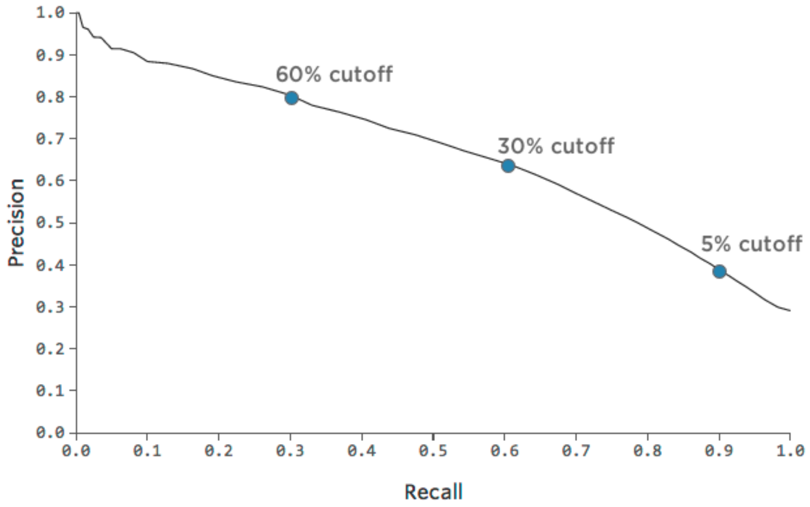


Precision/Recall Curve

- To evaluate a binary classifier, we can also compute the *area under the curve (AUC)* of a PR curve
- Given points on the curve, (R_k, P_k) we can approximate the AUC as:

$$AUC = 1 - \frac{1}{2}\sum_{k=1}^{n}(P_k + P_{k-1})(R_k - R_{k-1})$$

- An ideal PR curve will have an AUC of 1.0



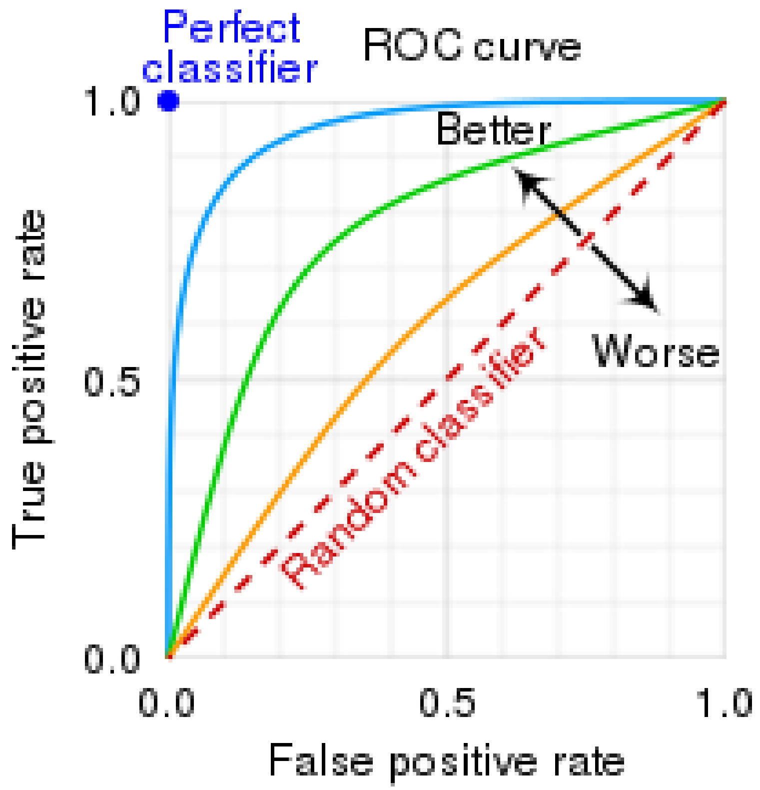
Receiver Operating Characteristic (ROC)

- Similar to how a Precision-Recall curve compares the tradeoff between precision and recall, a *receiver operating characteristic (ROC)* curve compares the tradeoff between the true positive rate and the false positive rate

$$TPR = \frac{TP}{TP+FN}$$

$$FPR = \frac{FP}{FP+TN}$$

- Again, an ROC curve with a larger area-under-the-curve is considered better.
- However, note that here the optimal location is on the top-left (as opposed to PR curve where it is the top-right).



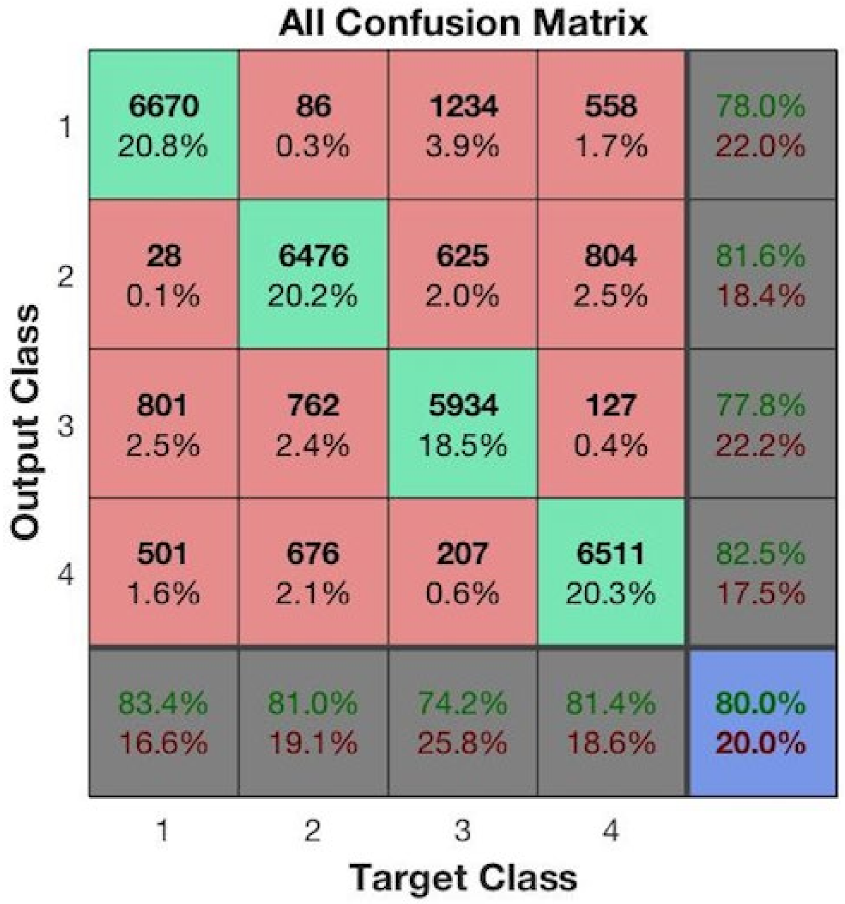
Multi-Class Evaluation

- Just like binary classification, we can evaluate the accuracy of a multi-class classifier:

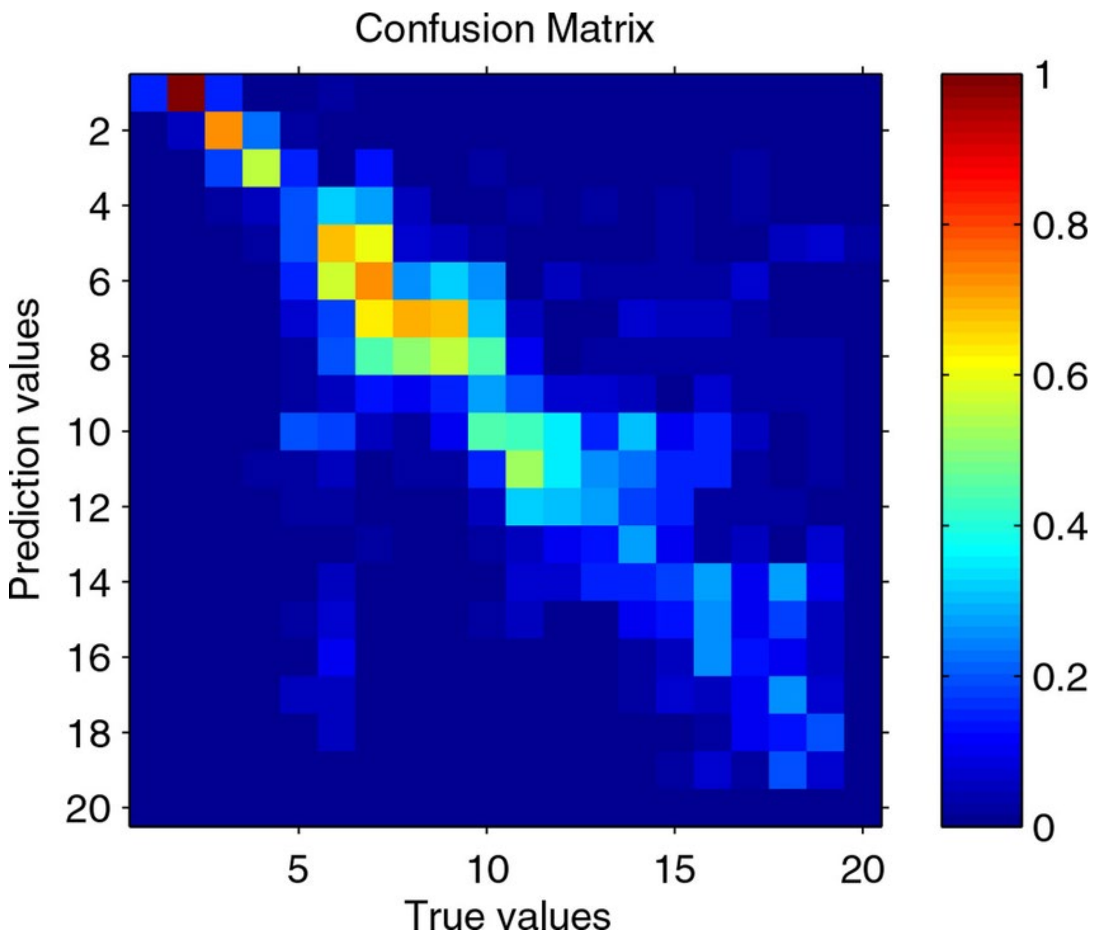
$$accuracy = \frac{1}{N}\sum_{i=1}^{N}(Y_i = \widehat{Y_i})$$

- In addition, particular to multi-class classification, we may be interested in investigating which classes get confused with which other classes
- To observe this, we can look at a *confusion matrix*

Confusion Matrix



Confusion Matrix



Learning Function

- In general, with supervised learning, with the *absence* of *noise* and with complete data in Z, we can say there is some function $f(\boldsymbol{z})$ such that

$$y = f(\boldsymbol{z})$$

- However, in reality, we observe a limited set of features and data
  - And some of it can be noisy

$$X \subset Z + \epsilon$$

- So, we want to do is to learn a function $g(\boldsymbol{x})$ that is an approximation of the true underlying $f(\boldsymbol{z})$

$$g(\boldsymbol{x}) \approx f(\boldsymbol{z})$$

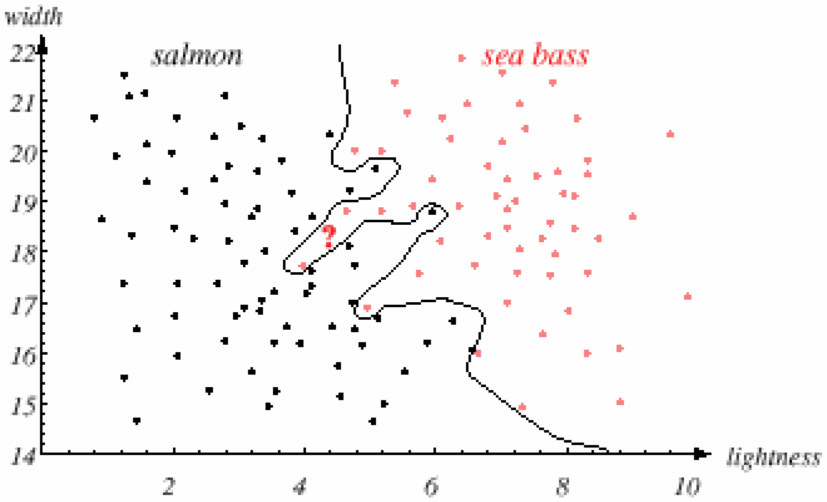
Generalization

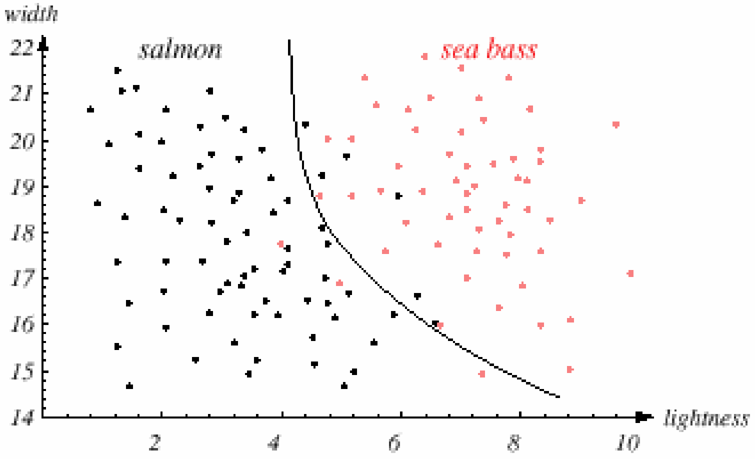
$$g(\boldsymbol{x}) \approx f(\boldsymbol{z})$$

- In addition, since our system/function/model is typically built off a subset of all the possible data, want to make sure also does well on data it was **not** built on.
  - In fact, this is even more important!
    - It’s pretty easy to do well on things you were built on…
- How well a function/system does on data it wasn’t trained on, is referred to as *generalization.*

Overfitting

- A model that doesn't generalize well is said to *overfit.*
- We're basically finding a function for the training data, not the function of the entire set of possible data.





Data Sets

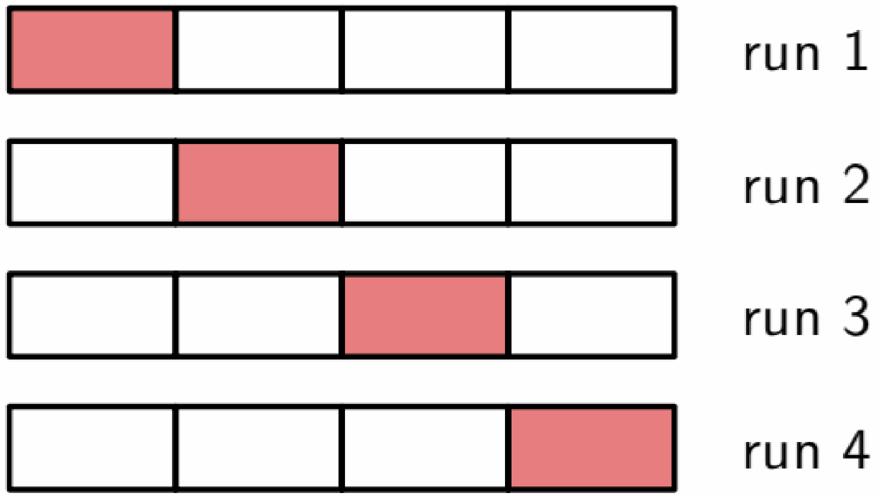
- To help us determine how well our model generalizes, we typically split our data into two groups:
  1. **Training Data**
  2. **Validation Data**
- Typically, this is done as a 2/3 training, 1/3 validation split
- We then build/train our system using the training data and check the generalizability of our system using the validation set.
- The key to a good model to have the training data and validation data pulled from the same distribution.
- **NOTE:** If you standardize or z-score your data, only do so with the training data.
  - Get the mean and std from the training data, and apply that to both the training and validation datasets.

Cross Validation

- What if we don't have that much data?
  - After all, the more data in the training set, the better!
- Then we can do something called *cross-validation*
- Here we do several training/validation runs
  - Keeping track of all the errors
- We can then compute statistics for our classifier based on the list of errors.

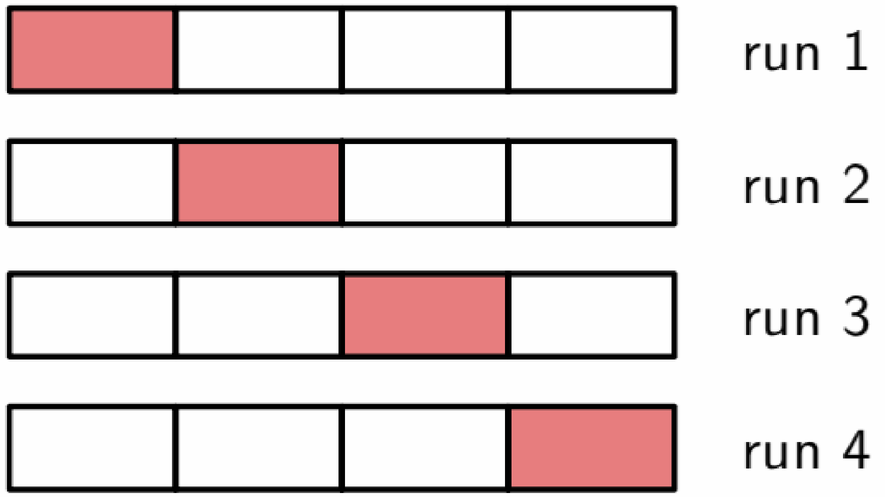
S-Folds Cross Validation

- There are a few types of cross-validation
  - S-Folds: Here we'll divide our data up into S parts, train on $S-1$ of them and validate the remaining part. Do this S times
  - Leave-one-out: If our data set is really small, we may want to build our system on $N-1$ samples and validate on just one sample. And do this N times (so it's basically N-folds).
- Again, for each system, if you are standardizing, just use the training portion to extract the mean and std.



S-Folds Cross Validation

- As long as S is large, each “system” is more robust/stable.
- What is the training/validation split of each system if we use
  - $S = 4$?
  - $S = 10$?



Cross Validation

- How do we “combine” all these different models?
- We (typically) can’t/don’t.
- The statistics give us a bound on what to expect for our final model.
- When it’s time to create a model to deploy, use ALL the data for training!