Machine Learning

CSE 142

Xin (Eric) Wang

Monday, November 22, 2021

Today

- Model ensembles (cont.) (Ch. 11)
- ML experiments (Ch. 12)

Model ensembles

Chapter 11 in the textbook

Model ensembles

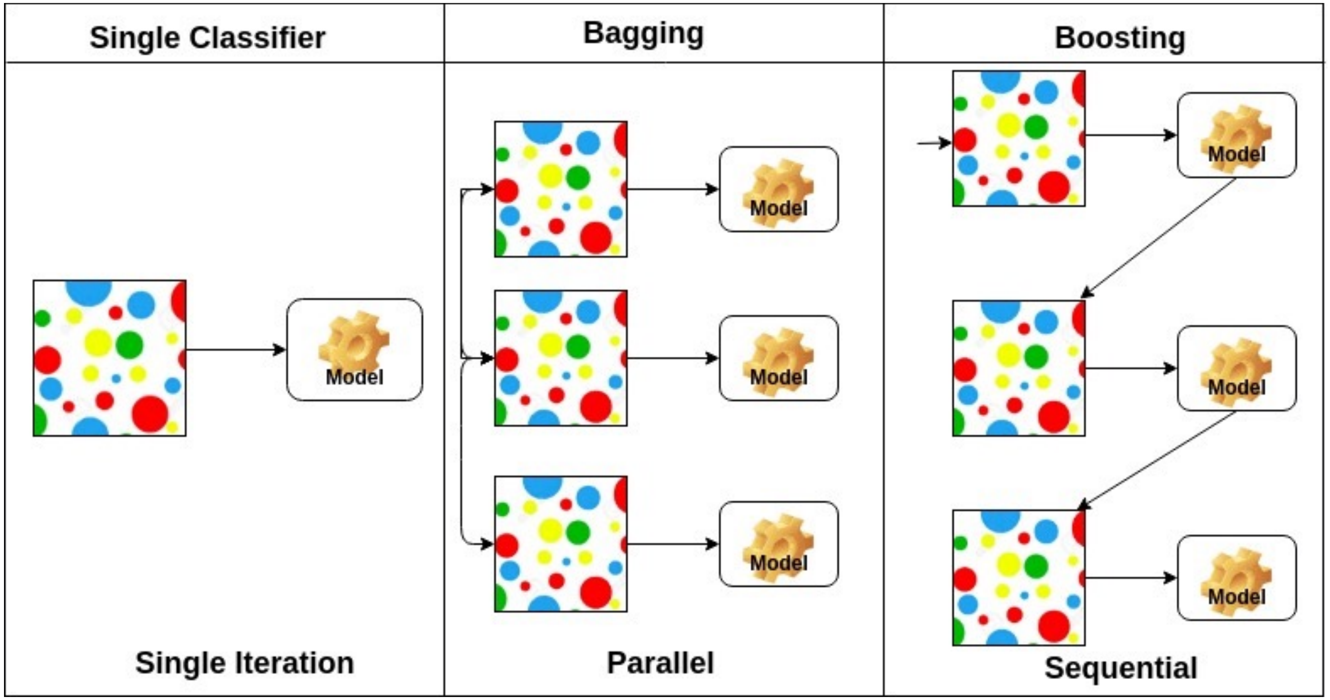
- We've seen how combining features can be beneficial
- We can also combine models to increase performance
  - Combinations of models are known as model ensembles
  - Potential for better performance at the cost of increased complexity
- General approach to model ensembles:
  - Construct multiple different models from adapted versions of the training data (e.g., reweighted or resampled)
  - Combine the predictions of these models in some way (averaging, voting, weighted voting, etc.)
- Two of the best-known ensemble methods are bagging and boosting
- These are "meta" methods – i.e., they are independent of the particular learning method (linear classifier, SVM, etc.)

Bagging

- Bagging = "bootstrap aggregation"
  - Create T models on different random samples of the training data set
  - Each sample is a set of training data called a bootstrap sample
    - Could be any size – often |D| is used, the size of the training set
- Bootstrap samples: Sample the data set with replacement (i.e., a data point can be chosen more than once in a bootstrap sample)
  - For $j = 1$ to |D|, choose with uniform probability from training data points $D = \{ d_1, d_2, \ldots, d_{|D|} \}$
  - Gather these into the bootstrap sample D_i
- Use $\{ D_1, D_2, \ldots, D_T \}$ to train models $\{ M_1, M_2, \ldots, M_T \}$, then combine the predictions of the T models
- Differences between the T bootstrap samples create diversity among the models in the ensemble

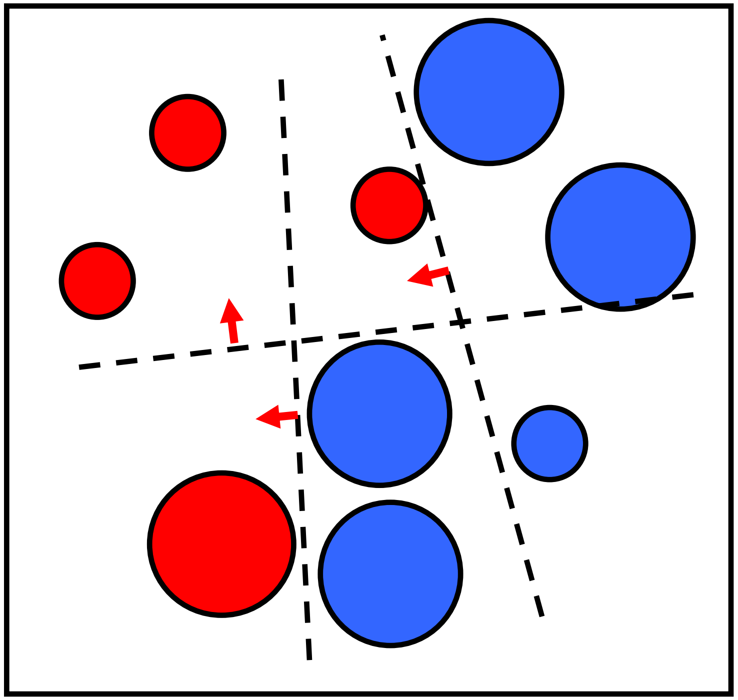
Boosting

- Boosting is similar to bagging, but it uses a more sophisticated technique to create diverse training sets
- The focus is on adding classifiers that do better on the misclassifications from earlier classifiers
  - By giving them a higher weight



Boosting

- How much should the weights of training examples change?
  - Assign half of the total weights to the misclassified examples
- How to combine the individual models?
  - Confidence factor α_t for each model
  - High α_t to model with low error rate

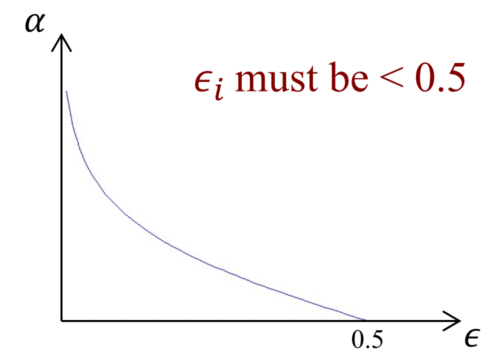


Boosting

Assumption is complex problem, lots of data, no perfect classifier. I.e., $\epsilon_t > 0$

- Procedure:
  - Assign equal weights w_j to training data points
  - Train a classifier; assign it a confidence factor α_t based on the error rate ϵ_t

$$\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$$



  - Give misclassified instances a higher weight
    - Assign half of the total weight to the misclassified examples

Misclassified points

$$w' = \frac{w}{2\epsilon_t}$$

Correctly classified points

$$w' = \frac{w}{2(1-\epsilon_t)}$$

Weights will sum to 0.5+0.5=1

  - Repeat for T classifiers or until $\epsilon_t \geq 0.5$
  - The ensemble predictor is a weighted average of the models (rather than majority vote)

$$M(x) = \sum_{t=1}^{T} \alpha_t M_t(x)$$

  - Threshold for binary output

Boosting

Computational complexity: $O(TDK)$
(ensemble size x data size x dimensionality)

Algorithm Boosting($D, T, \mathscr{A}$) – train an ensemble of binary classifiers from reweighted training sets.

Input : data set D; ensemble size T; learning algorithm $\mathscr{A}$.
Output : weighted ensemble of models.

$w_{1i} \leftarrow 1/|D|$ for all $x_i \in D$; // start with uniform weights
for $t = 1$ to T **do**
  run $\mathscr{A}$ on D with weights w_{ti} to produce a model M_t;
  calculate weighted error ϵ_t;
  if $\epsilon_t \geq 1/2$ **then**
    set $T \leftarrow t - 1$ and break
  end
  $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$; // confidence for this model
  $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$ for misclassified instances $x_i \in D$; // increase weight
  $w_{(t+1)j} \leftarrow \frac{w_{tj}}{2(1-\epsilon_t)}$ for correctly classified instances $x_j \in D$; // decrease
end
return $M(x) = \sum_{t=1}^{T} \alpha_t M_t(x)$

Boosting example

Misclassified points

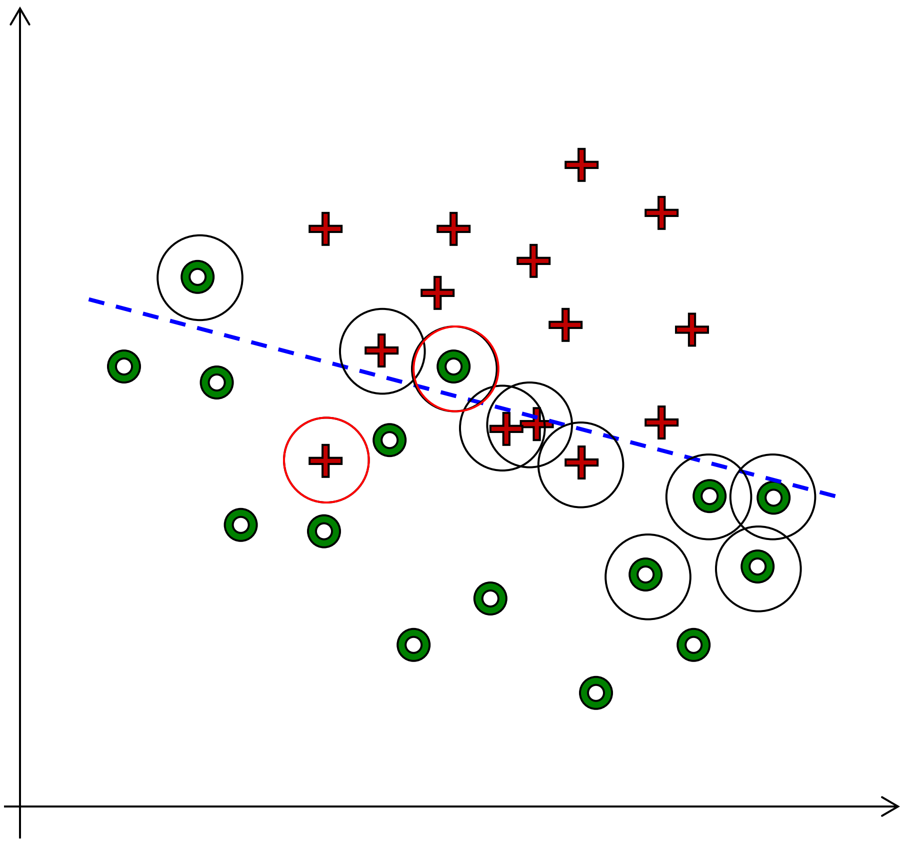
$$w' = \frac{w}{2\epsilon_t}$$

Correctly classified points

$$w' = \frac{w}{2(1-\epsilon_t)}$$

29 points

Initial weights $w = 1/29 = 0.034$



Round 1:

6 errors

$$\epsilon_1 = \frac{6}{29} = 0.21$$

$$\alpha_1 = \frac{1}{2}\ln\frac{1-\epsilon_1}{\epsilon_1} = 0.67$$

$$w' = \{\frac{w}{2\epsilon_t}, \frac{w}{2(1-\epsilon_t)}\} = \{\ 0.082, 0.021\ \}$$

Round 2:

7 errors => $\epsilon_2 = \frac{7}{29} = 0.24$?

$$\epsilon_2 = \frac{2 * 0.082 + 5 * 0.021}{1} = 0.28$$

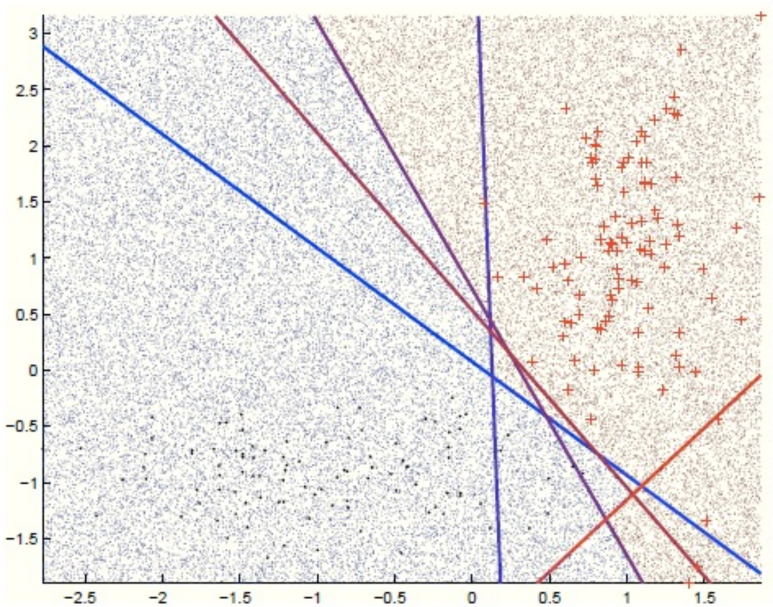
$$\alpha_2 = \frac{1}{2}\ln\frac{1-\epsilon_2}{\epsilon_2} = 0.47$$

$$w' = \{\frac{w}{2\epsilon_t}, \frac{w}{2(1-\epsilon_t)}\ \}$$

Continue for T iterations or until $\epsilon \geq 0.5$

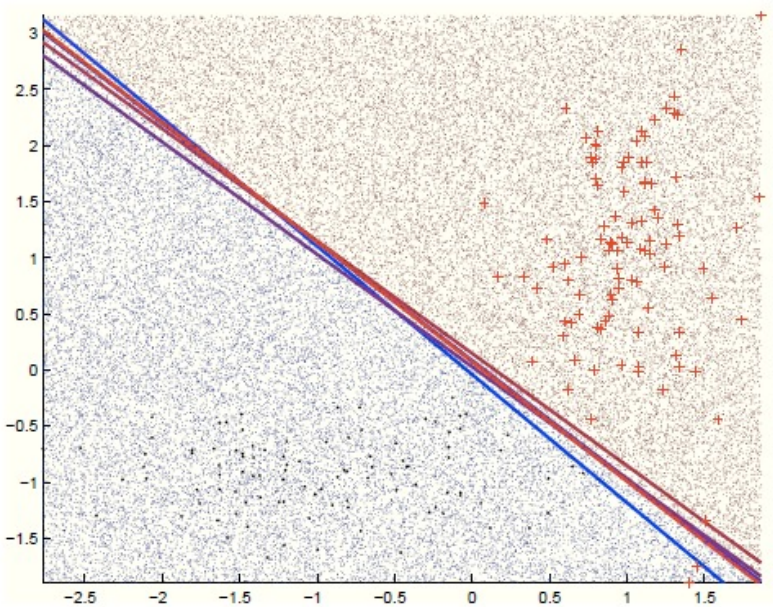
Boosting vs. bagging

Boosting



Can achieve zero training error by focusing on the misclassifications

Bagging



With relatively large bootstrap sample sets, there tends to be little diversity in the learned models

Ensemble methods: summary

- Model ensemble methods are meta-methods: they are ways to combine ML models, but they are agnostic to the kind of model being used
  - Simple linear classifier, Perceptron, SVM, neural network, etc. (weighted versions)
- They provide an opportunity to improve performance and (mostly) avoid over-fitting to the training data
  - Typically using randomization of some sort
- Bagging is essentially a variance reduction technique (increase consistency), while boosting is essentially a bias reduction technique (increase accuracy)
- There are many other ensemble approaches in machine learning

Machine learning experiments

Chapter 12

Machine learning experiments

- While there is a lot of important theory in machine learning (e.g., algorithms, proving convergence, proving performance bounds, establishing computability, etc.), it is a practical field, applying algorithms to data for practical use
- ML experiments are critical for confirming the various assumptions in a ML problem and establishing realistic expectations for performance
- Some key questions for ML experiments in a domain D:
  - How does a specific model perform on data from D?
  - Which of a set of models has the best performance on D?
  - How do models from learning algorithm A perform on D?
  - Which learning algorithm A produces the best model(s) for domain D?
- How do we define performance?

False positive rate (FPR) $= \frac{FP}{N} = \alpha$

False negative rate (FNR) $= \frac{FN}{P} = \beta$

True positive rate (TPR) $= \frac{TP}{P}$ = Sensitivity = Recall = 1 - β

True negative rate (TNR) $= \frac{TN}{N}$ = Specificity = 1 - α

$\boxed{\text{Accuracy}} = \frac{TP+TN}{P+N} = \left(\frac{P}{P+N}\right) TPR + \left(\frac{N}{P+N}\right) TNR$

$\boxed{\text{Error rate}} = \frac{FP+FN}{P+N}$

Precision $= \frac{TP}{\hat{P}}$

Accuracy + error rate = 1

Average recall $= \frac{TPR+TNR}{2}$

Contingency table:

		Actual class C: 1	0	
Predicted class $\hat{C}$	1	TP	FP	*Estimated positive* $\hat{P}$
	0	FN	TN	*Estimated negative* $\hat{N}$
		Positives P	*Negatives* N	TOTAL

Example

		c	
		1	0
$\hat{c}$	1	50	0
	0	50	10
		P=100	N=10

Accuracy = 60/110 = 0.55
Error rate = 1 – 0.55 = 0.45

Average recall = (0.5+1.0)/2 = 0.75

		c	
		1	0
$\hat{c}$	1	50	0
	0	50	100
		P=100	N=100

Accuracy = 150/200 = 0.75
Error rate = 1 – 0.75 = 0.25

Average recall = (0.5+1.0)/2 = 0.75

Performance measure

- If we choose accuracy (or error rate) as the measure of performance, we're making an assumption that the class distribution of the test set is representative of the real problem domain
- If we choose average recall, we're making an assumption that the real problem consists of uniform class distributions (equally likely)
- In general, when running an experiment, we should keep a record of a sufficient set of measurements to enable reproduction of the contingency table – in order to compute additional values later on if needed
  - Need four of the independent values (e.g., FP, FN, P, N)

Performance measure (testing)

- We want to estimate the true error rate = the error rate on the entire population (not just on your data set)

- You cannot use your full dataset to train a model, estimate the model parameters, and estimate the performance (e.g., accuracy or error rate) of your model
  - This results in overfitting and overly optimistic results!

- Testing data is used to predict the performance of your ML model on the real problem
  - Applied after the model is finalized; it is not used to modify the model
  - In other words, testing is just a substitute for deploying your model in a product and gathering data from millions of users
  - Only training and validation data can change your model (*)

Training, validation, and testing

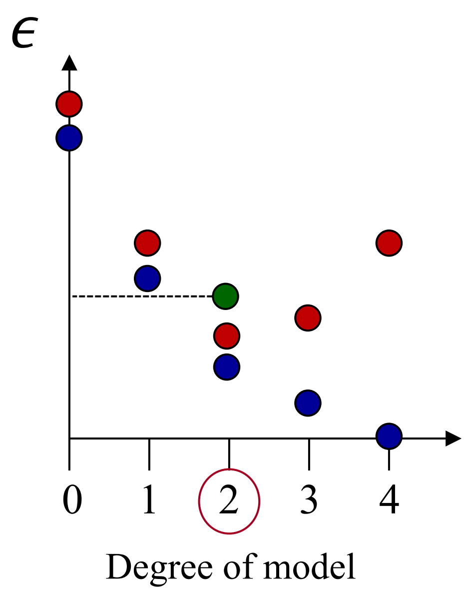
- Testing on the training data, or training on the testing data, is cheating!
  - Training/validation data and testing data must be completely separate
  - The test data is (conceptually) stored in a vault until ready to be used for testing the completed model
  - You cannot modify the model after assessing it with the testing data
    - At least, you can't modify it and re-test

- Instead, we split the training data into disjoint sets for validation
  - Holdout method
    - Keep some data separate (typically 10-30%) for testing
  - k-fold cross-validation: average the k different performance estimates
  - Leave-one-out cross-validation
    - When k = the number of data points

Typical training/validation/testing example

- For a 1D regression problem, let's train different models (different regression functions):
  - 0-degree polynomial (a constant), 1-degree polynomial (a line), 2nd-degree polynomial, 3rd-degree polynomial, 4th-degree polynomial

- Errors on the training data (blue dots):



Which is the best model?

Answer: We don't know!

Use the validation data (red dots) to determine:

The second-degree polynomial has the lowest error on this validation data

Now run cross-validation on the data set to estimate the performance of your model

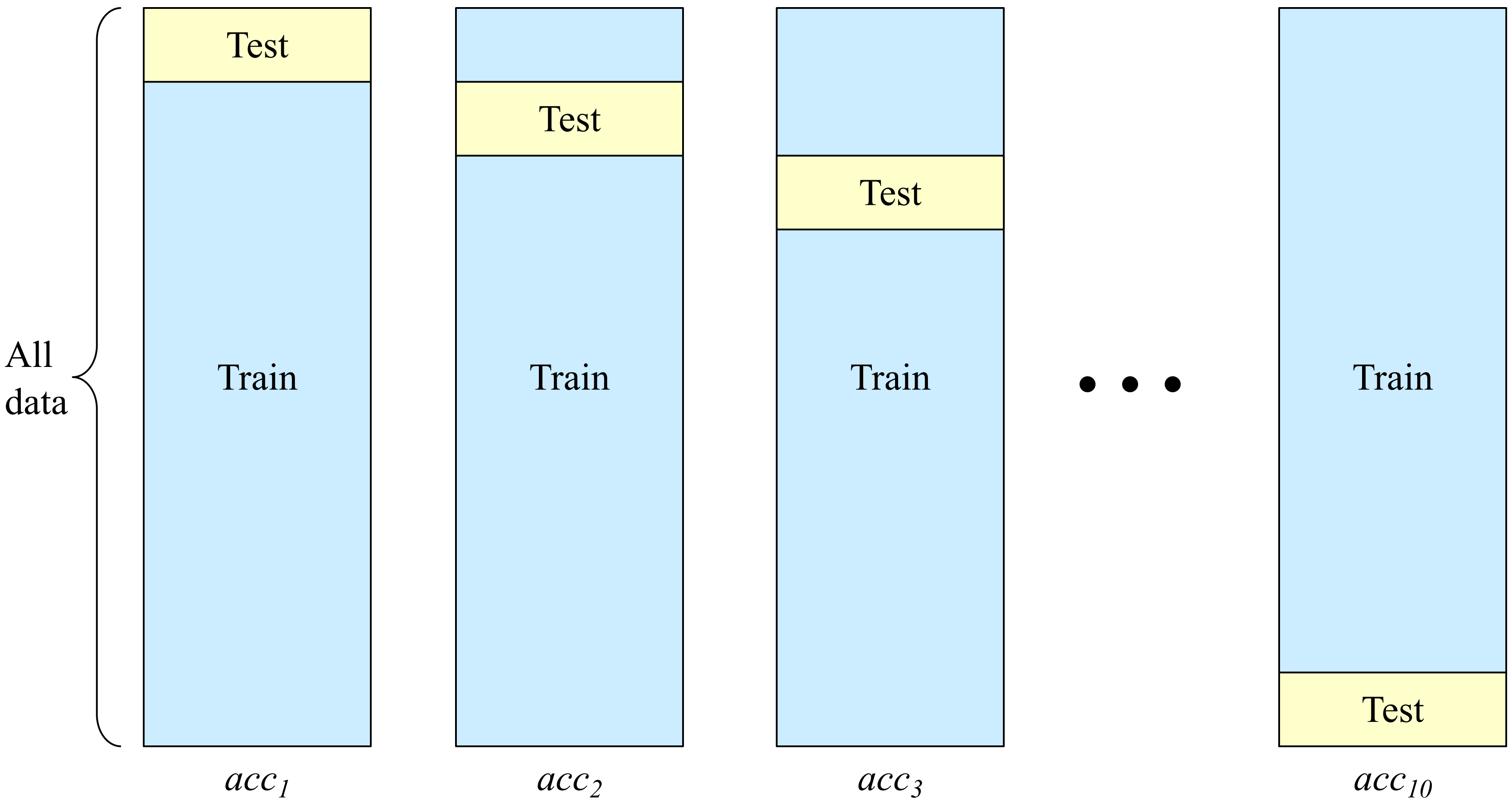
Produce the final model by using all your data

Ship it!

Cross-validation

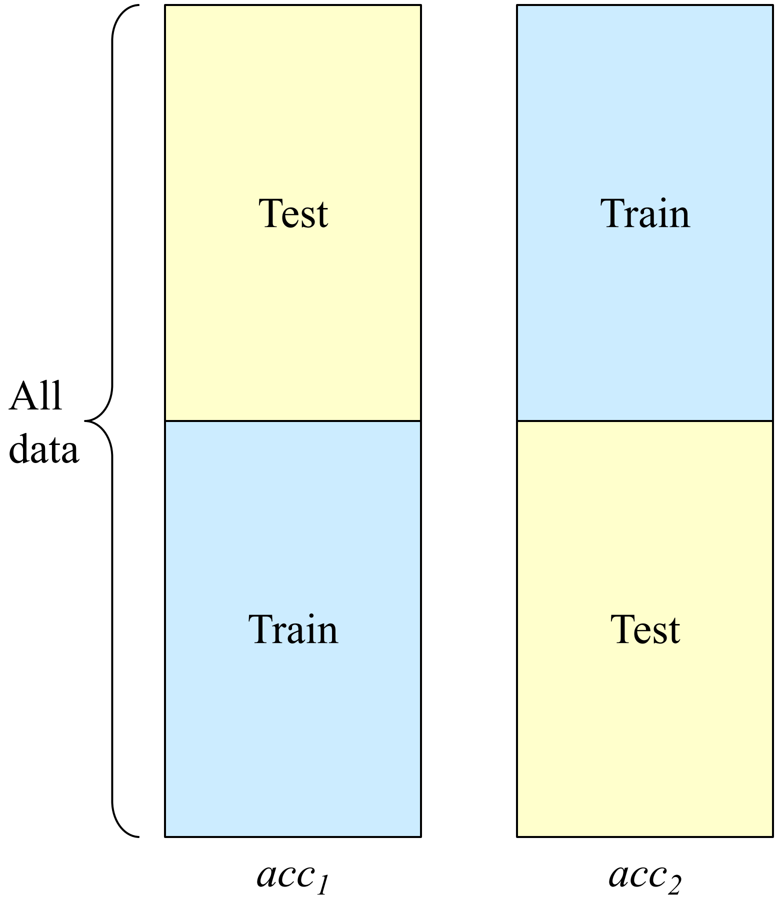
- The cross-validation process for experimental evaluation:
  - Randomly partition the experimental data into k parts (“folds”)
  - Train the model on k–1 folds
  - Evaluate it on the remaining test fold
  - Repeat for a total of k times, evaluating on each test fold
  - Average the performance measure (e.g., accuracy), over the k trials
- This is k-fold cross-validation
  - Frequently, $k = 10$, but other values are also used
    - $k = 2$ or 5 is quite common
  - Rule of thumb: folds should contain at least 30 instances
    - Implying $\geq$300 instances in the data set for $k = 10$
  - If $k = n$ (the number of data points), this is leave-one-out cross-validation

Example: 10-fold cross-validation



$$acc = \text{Average}(acc_i)$$

Example: 2-fold cross-validation

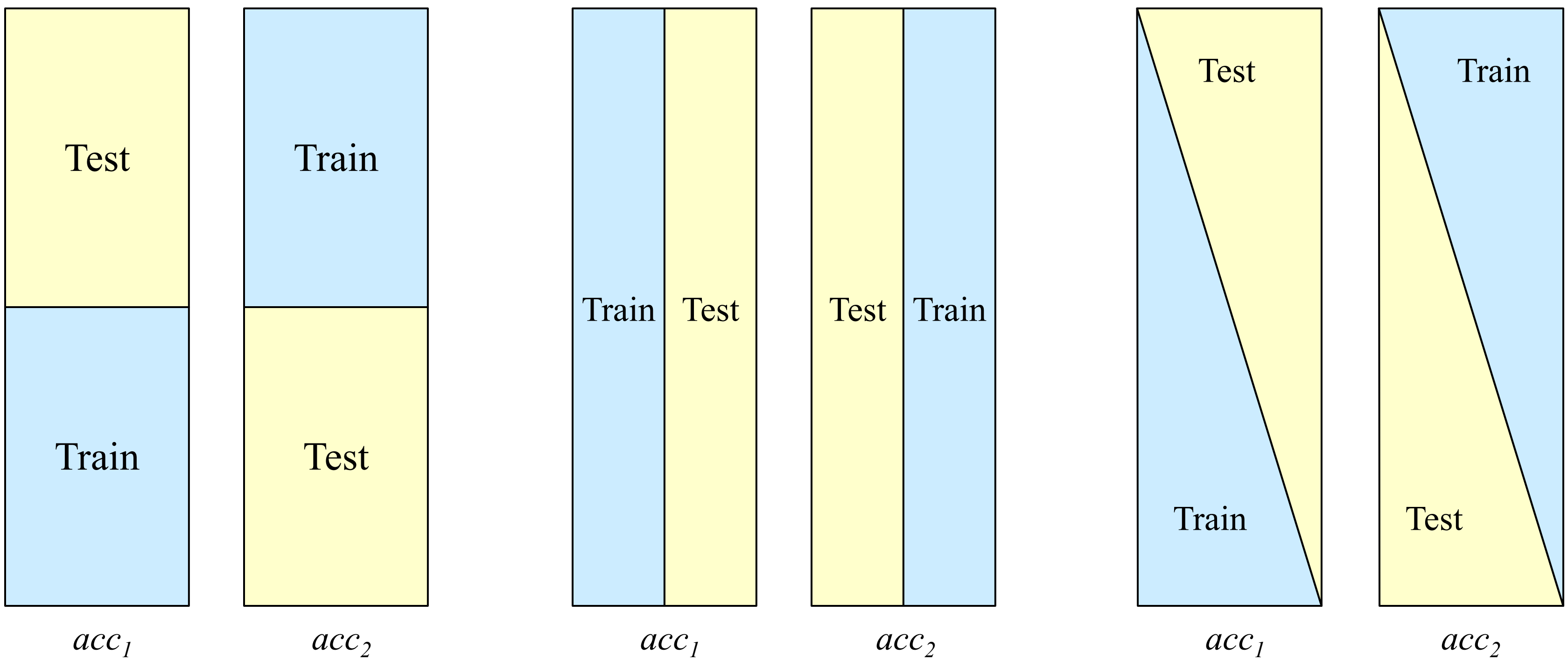


$$acc = \tfrac{1}{2}\,(acc_1 + acc_2)$$

Repeated random sub-sampling validation

Randomly split the data into k folds N times, and then average the N results of k-fold cross-validation

– Typically, $k = 2$



$acc_a = ½\,(acc_1 + acc_2)$ $\quad$ $acc_b = ½\,(acc_1 + acc_2)$ $\quad$ $acc_c = ½\,(acc_1 + acc_2)$

$$\boxed{\hat{a} = (acc_a + acc_b + acc_c)/3}$$

Cross-validation

- In practice, the choice of the number of folds depends on the size of the data set
  - For large data sets, fewer folds are needed
  - For sparse data sets, leave-one-out may be best
- We should have some notion of acceptable variance for the cross-validation performance measure
  - If the variance is high (among the *k* performance measures), we probably need more data!
- If we are satisfied with the performance of our learning algorithm, we than run it over the entire data set (i.e., train on the complete data set) to produce the final model (*)
  - Hence the term cross-validation rather than cross-testing!
  - (In this case – oddly – we never test the *final* model.)

Notes on terminology

- Train (training), validate (validation), test (testing), evaluate (evaluation)
  - There can be some ambiguity among these term, depending on the context
- Keep in mind:
  - We can evaluate a candidate model
    - Validation
  - We can evaluate the final system performance
    - Testing
  - We can evaluate performance on the training data
    - This might be done as part of the training algorithm or as part of the validation process
  - A given data point (from your data set) can play multiple roles (training, validation, and testing)
    - But never use data for validation or testing that was used for training on the same model!

Model parameters and hyperparameters

- Machine learning algorithms learn the model parameters
  - You choose what model is to be learned (linear regression, decision tree, SVM, neural network, etc.)
  - Training sets the values of various parameters (e.g., the ***w*** vector, the DT decision nodes, the α values for a kernel perceptron, etc.)
- Models may also have hyperparameters – settings or parameters that are determined outside the learning algorithm itself, fixed during training
  - E.g., the kernel function used, the highest order of a polynomial function, the value of k for k-NN, the number of hidden layers in a NN, etc.
- We can think of validation as the process of determining the preferred hyperparameters
  - Train to set parameters; validate to determine hyperparameters

Quiz: ML experiments

- See the quiz “ML experiments” on Canvas