CS 4260 and CS 5260
Vanderbilt University

Lecture on Evaluation

This lecture assumes that you have
- Read Section 7.1 through 7.2 of ArtInt and

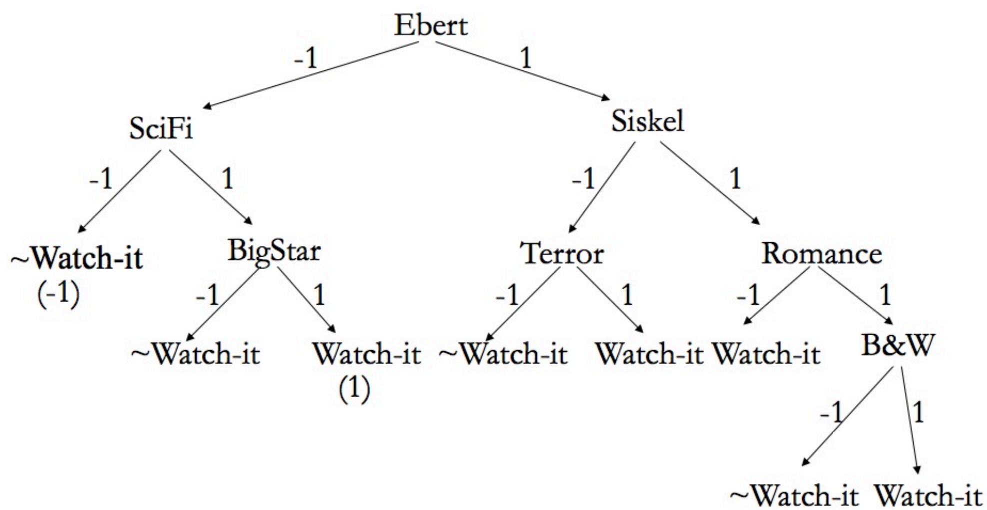
ArtInt: Poole and Mackworth, Artificial Intelligence 2E
at http://artint.info/2e/html/ArtInt2e.html
to include slides at http://artint.info/2e/slides/ch04/lect1.pdf

We have see two supervised machine learning strategies

- Naïve Bayesian learning (was optional during ML week; required later))

  P(C=c1) * P(SciFi = -1|c1) * P(Terror = 1|c1) * P(Romance = -1|c1) * P(Ebert = 1|c1) * P(Siskel = 1|c1) * ….]

- Decision tree learning



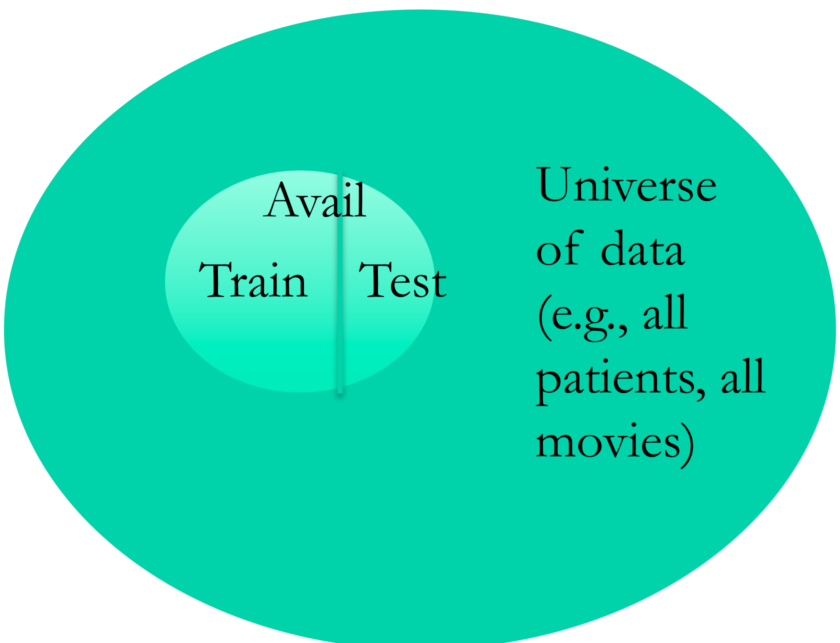
How can we compare them?

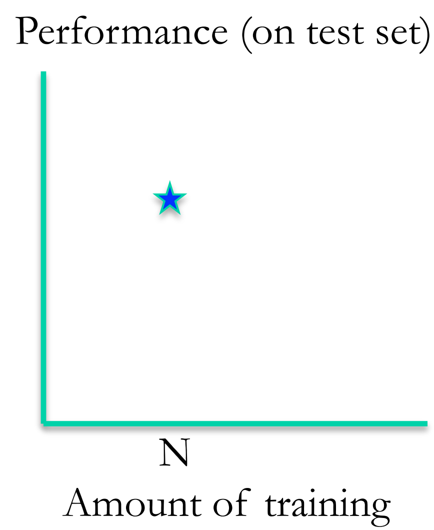
How can we characterize their learning independent of each other?

How do we parameterize each of them to maximize performance?

Training and Test Data Sets

- Testing a classifier/predictor on data that was used for training is overly optimistic,
  - even if the method doesn't memorize each data per se
- More realistic, in most cases, is to test on previously unseen data
- What does this tell us?
- If there are N training data, then test set accuracy (or error) approximates (to an unknown extent) the performance of classifiers constructed by the learning method on N training data





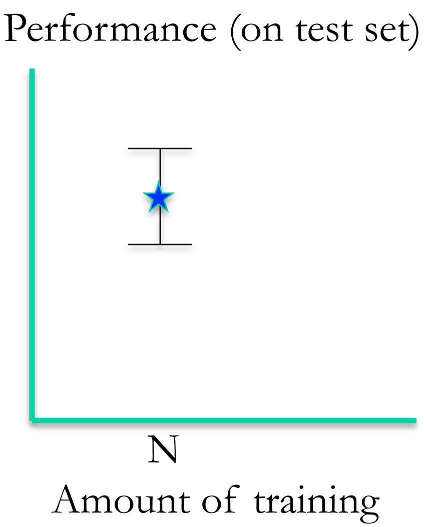
Douglas H. Fisher

But typically, most interested in performance on universe

Using repeated Train and Test splits

Given: M data available, Avail
Learning Trials, L
Training set size, N
Test set size, M-N

Local: Training Set, Train
Test Set, Test
Classifier
AggregatedPerformance (e.g., Mean, Median, Mode)

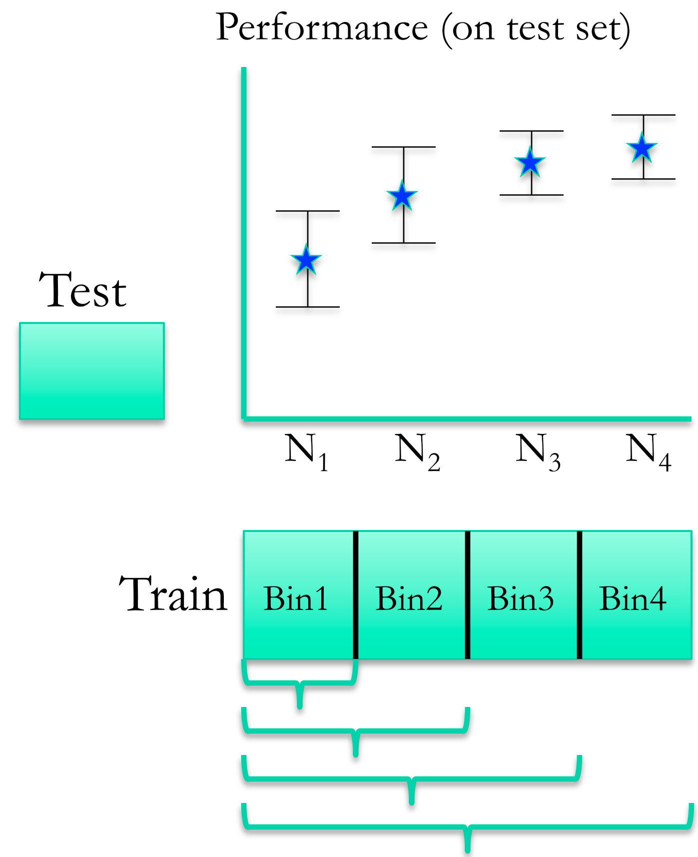


Initialize AggregatedPerformance
Repeat L times {
    Train ← Randomly draw N data from Avail, "without replacement"
    Test ← Avail – Train
    Classifier ← Learn(Train)
    AggregatedPerformance ← Performance(Classifier, Test) + AggregatedPerformance
}
Return AggregatedPerformance

This provides approximation of performance (on Universe)
of learning method at training size of N

Douglas H. Fisher

Generating Learning Curves through repeated Train and Test splits



Given: M data available, Avail
Learning Trials, L
Training set sizes, $N_1 \ldots N_{max}$
Test set size, $M\text{-}N_{max}$

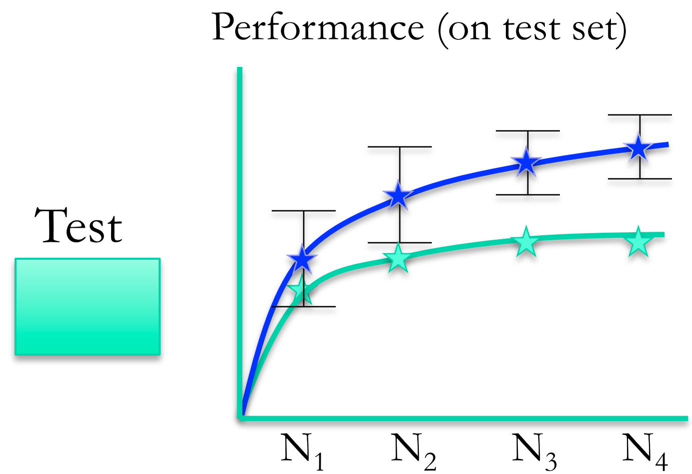
Local: Training Set, Train
Test Set, Test
Classifier
AggregatedPerformanceVector

Initialize AggregatedPerformanceVector
Repeat L times {
    Train $\leftarrow$ Randomly draw N_{max} data from Avail, "without replacement"
    Test $\leftarrow$ Avail – Train
    Partition Train into 1 to max bins, $TrainBin_1$ through $TrainBin_{max}$
    For k = 1 to max {
        Classifier $\leftarrow$ Learn(Union of $TrainBin_1$ through $TrainBin_k$)
        AggregatedPerformanceVector[k]
            $\leftarrow$ Performance(Classifier, Test) + AggregatedPerformanceVector[k]
}
Return AggregatedPerformanceVector

Douglas H. Fisher

This provides approximation of performance (on Universe) of learning method at different training N

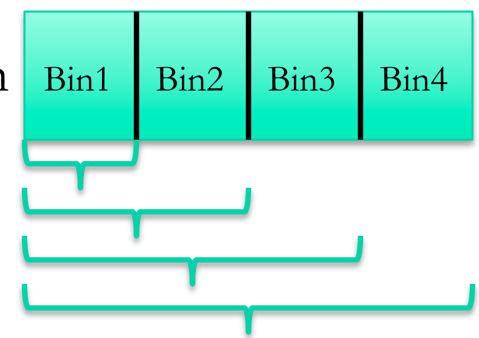
Comparing Learning Curves of Different Supervised Machine Learning Methods



Given: …
Local: …

Initialize AggregatedPerformanceVectorMethod1 (e.g., TDIDT)
Initialize AggregatedPerformanceVectorMethod2 (e.g., NBC)
Repeat L times {
    Train ← Randomly draw N_{max} data from Avail, "without replacement"
    Test ← Avail – Train
    Partition Train into 1-to-max bins, $TrainBin_1$ through $TrainBin_{max}$
    For k = 1 to max {
        Classifier1 ← Learn1(Union of $TrainBin_1$ through $TrainBin_k$)
        AggregatedPerformanceVectorMethod1[k]
            ← Performance(Classifier1, Test) + AggregatedPerformanceVector1[k]
        Classifier2 ← Learn2(Union of $TrainBin_1$ through $TrainBin_k$)
        AggregatedPerformanceVectorMethod2[k]
            ← Performance(Classifier2, Test) + AggregatedPerformanceVector2[k]
}
Return AggregatedPerformanceVectorMethod1, AggregatedPerformanceVectorMethod2

Train



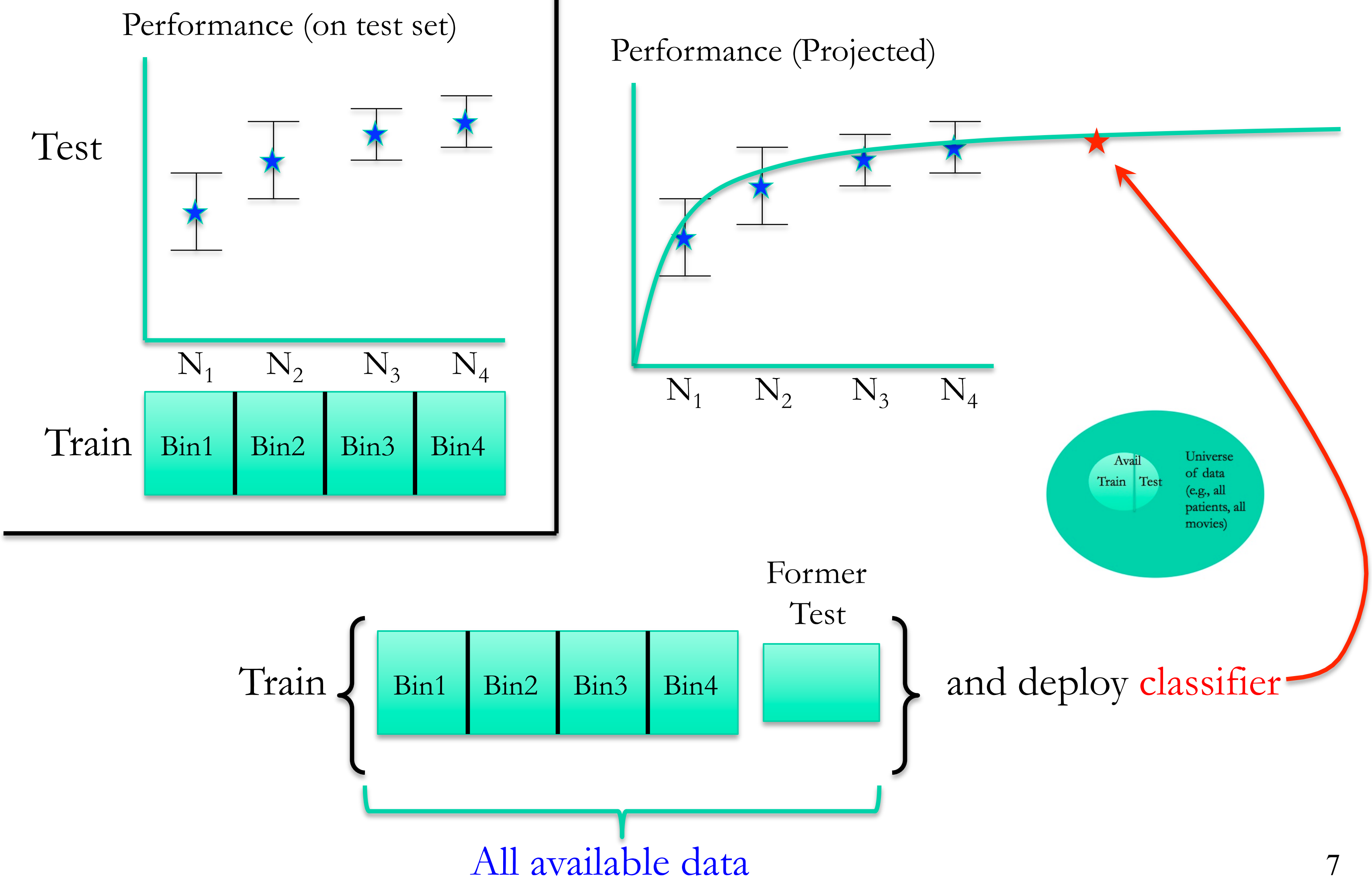
This provides approximations of performance (on Universe) of each method at different training N

Douglas H. Fisher

How Might we use in Real Setting

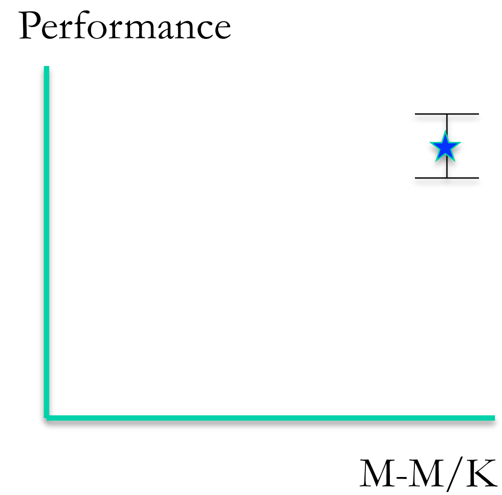
1. Use Training/Test splits to plot performance

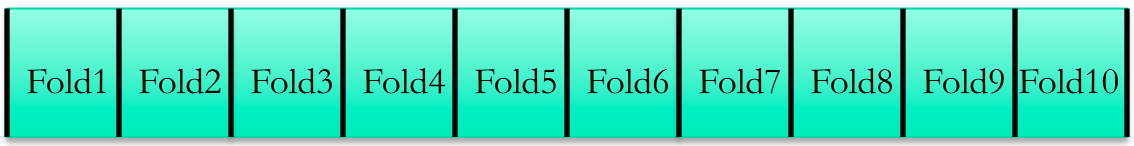
2. Curve fit learning behavior to project performance (on universe) at larger N



K-Fold Cross Validation

1. Randomize order of available M data
2. divide available data into K (e.g., 10) equal size bins or folds
3. For I = 1 to K {
   - Train on union of all folds, except fold_I
   - Test on fold_I
4. Average results





All available data

M-Fold Cross Validation
(or leave-one-out cross validation)

Divide a data set of size M into M singleton folds, and follow algorithm above
(e.g., for each dataum, train on M-1 other data and test on the datum)

This is often regarded as the best way to leverage the existing data and get as close as one can to estimating performance on a final deployed classifier trained on all data