Model Selection: Learning Goals

1. "Confusion" look at accuracy
2. Receiver Operating Curve
3. K-fold cross validation
4. Model complexity

- Russell 18.4

Closer Look at Accuracy

- So far we ‘ve used accuracy – the percentage of correct predictions (of both positive and negative examples) – to evaluate a tree
- Confusion matrix (binary classification) provides a closer look

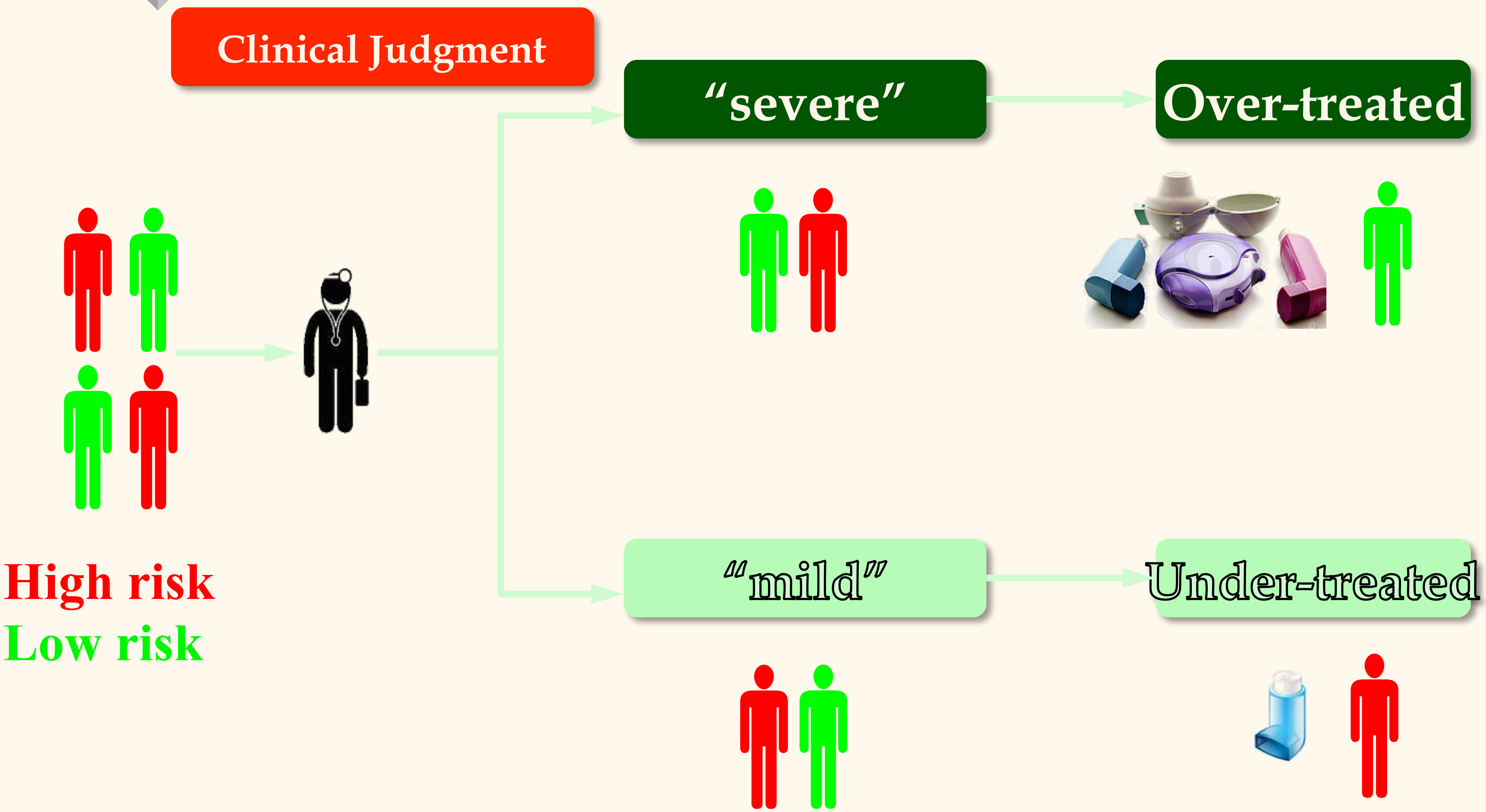
	Predicted 1	Predicted 0
True 1	True Positive	False Negative
True 0	False Positive	True Negative

Accuracy = (TP+TN)/(TP+FN+FP+TN)

Sensitivity = TP/(TP+FN)

Specificity = TN/(TN+FP)

Recall case study: COPD



Closer Look at Accuracy (2)

- COPD: positive = identifying high risk patients
  - Sensitivity: proportion of high risks identified as such
  - Specificity: proportion of low risks identified as such
- What is the cost of FN? Cost of FP?
- Often times, the two costs are not considered equal
  - Easier to tolerate FP than FN, e.g., SARS quarantine
  - Easier to tolerate FN than FP, e.g., toxic medication

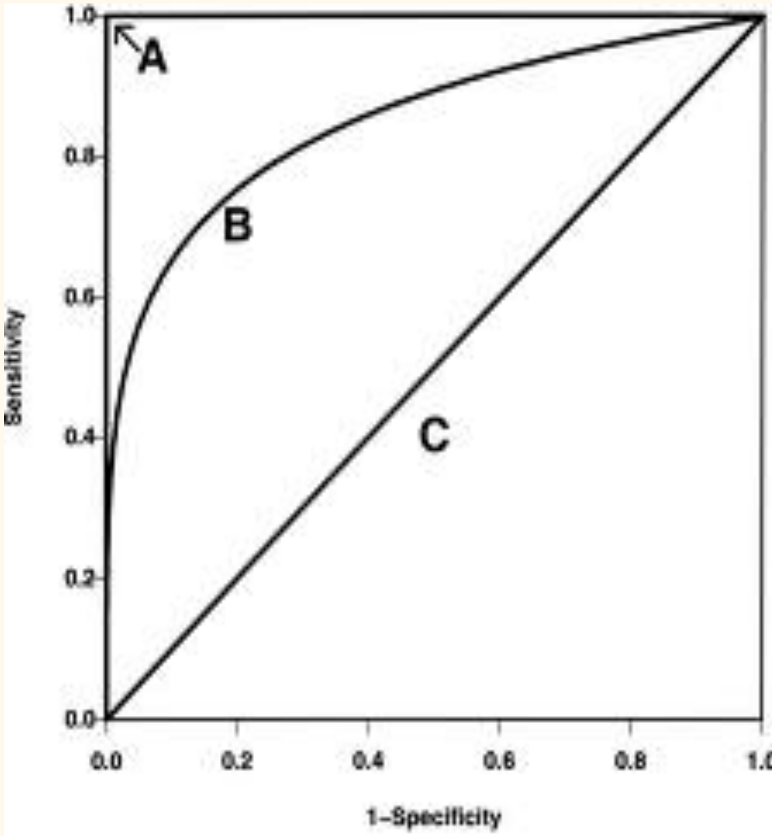
	Predicted 1	Predicted 0
True 1	TP	FN
True 0	FP	TN

Optimizing FP vs FN

- When we compare the performance of two classifiers A and B:
  - Pick A if both sensitivity and specificity are better
  - What if A has better sensitivity but B better specificity?
  - The choice depends on minimizing FP vs minimizing FN

Receiver Operating Graph

- A plot with the x-axis being *(1 – specificity)* and the y-axis being *sensitivity*
- The ideal classifier is the upper left corner (0,1), i.e., specificity = 1 and sensitivity = 1
- A better classifier is one whose performance is closer to (0,1)

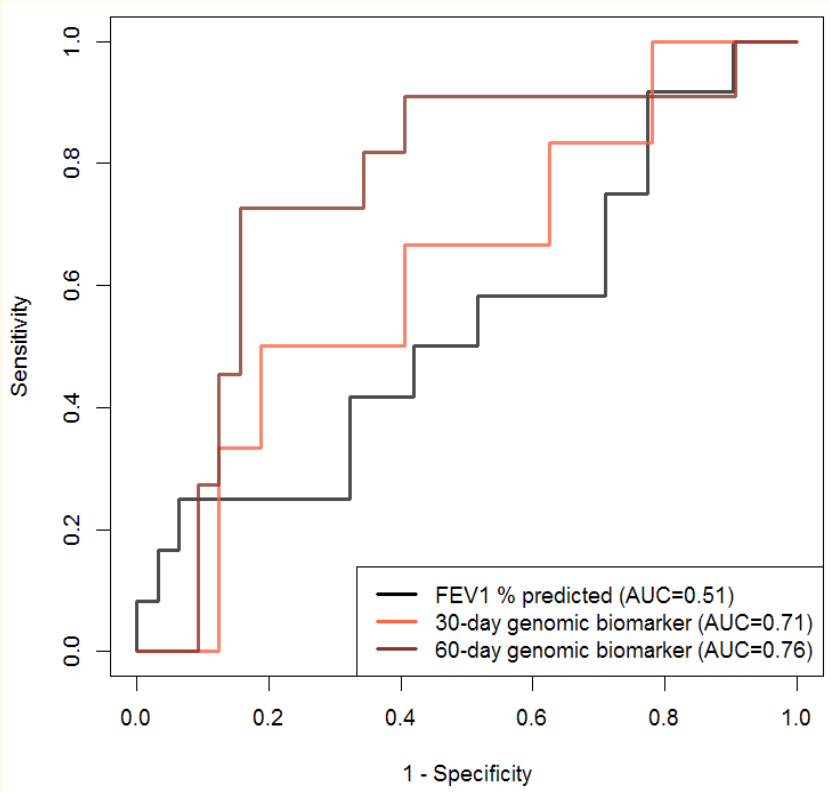


Receiver Operating Curve

- So far we've dealt with classifiers that generate only binary predicted values
- We will soon work with classifiers that give a probability that a test case is in the positive class
- To turn that into a binary predicted value, we have the additional freedom to pick a probability threshold α, i.e, called positive when $p \geq \alpha$ (which need not be 0.5)
  - This is exactly how we optimize the tradeoffs btw FP and FN
- Each α determines a sensitivity and specificity, i.e, one point in the RO plot
- It is desirable to evaluate all possible α's, which leads to a receiver operating curve for a classifier

Receiver Operating Curve (2)

- When we compare two classifiers, now we compare not just two points – but two *curves*
- The curve that is "on top" of the other gives a better classifier, i.e., closer to (0,1)
  - Better be on top of the "random guess" line



Comparing Classifiers: AUC

- The curve that rises as fast to (0,1) the better
- As discussed before, we need to minimize FP vs FN in picking a threshold
- But in general, in terms of information captured in a classifier, we often use the area under the ROC (AUC)
  - Averaging across all the threshold values
  - Theorem: the area is the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative one

Clicker Question

- What is the AUCs for the ideal classifier and the random classifier?

a) 1 and 0 respectively.

b) 0.5 and 0.

c) 1 and 0.5.

Imbalance Training Sets

- # positive examples « # negative examples
  - E.g., 1:10, 1:100 and even 1:1000
  - E.g., a rare disease
  - E.g., textbook: spam vs non-spam emails
- E.g., decision trees
  - A very large number of negatives dominate the space partitioning
  - Regions for positives become very small
  - E.g., a single low-risk example in the age-carType case

Correction Heuristics

- Without adjustment, performance would be dominated by how well the classifier predicts negative examples
- May want to give a much heavier weight on sensitivity, than on specificity
- Under-sampling the negative examples
  - remove noisy, redundant and border negatives
- Over-sampling the positive examples
  - interpolate positives by *assuming* that the objects between two positives are positive

Where is the Test Set?

- So far we've assumed that there is a test set
- More often than not, we have one *single* set of labeled data: (X_1,y_1), …, (X_N,y_N)
- How can we assess the performance of a classifier?
- Natural idea: split the set of labeled data into a training subset and a *disjoint* test subset
  - Disjoint is critical; absolutely nothing from the test set can be used in training
  - Otherwise, performance is inflated

What is the Right Proportion?

- To make sure that a classifier generalizes well, it is clear that we want to use as much data to train as possible
  - If use all N examples to train, nothing left to test
- If use (N-1) examples to train, only have 1 test case, which is a high variance
- The point is that for testing as well, the more examples the better
- 1-1 split? 2-1 split? 3-1 split?

What is the Right Proportion? (2)

- No correct answer to the proportion question
- Furthermore, how do we know that the test subset won't happen to be "lucky" (i.e., an inflated performance) or "unlucky" (i.e., a deflated performance)?
- Ideally, we would like every labeled example to be used for training and, if at all possible, to be used for testing – BUT *without* violating the disjointness requirement

2-fold Cross Validation

- Let us divide dataset into 2 folds:
  - Randomly pick N/2 examples into fold X, the remaining ones in fold Y
- Build a classifier C_X using fold X, test with fold Y and record performance
- *Don't stop:* build a new classifier C_Y using fold Y, test with fold X and record performance
- Finally, build the classifier **C_{X+Y} using all data**
- The **performance of C_{X+Y}** is *estimated* to be the ***average* performance of C_X and C_Y**

2-fold Cross Validation (2)

- Note that every example is used in training *and* in testing
- Yet the disjointness requirement is satisfied because the folds are disjoint
- Elegant!
- This is called 2-fold cross validation

K-fold Cross Validation

- Recall that we want to use as much data to train as possible
- We can go to 3-fold CV
  - Divide into X, Y, Z
  - Each time use two folds to train, the left-out fold to test
  - i.e., XY, XZ, YZ
- Generalize to k-fold CV, each time using (k-1) folds to train
- When k=N, each time use (N-1) examples to train
- This is called Leave-One-Out CV

K-fold Cross Validation (2)

- The larger k is, the more expensive, but gives a more accurate estimate on performance
- Best practice to estimate the performance of C_{all}
  - K = 10
  - Make sure that each fold has an equal number of positive examples
  - The act of randomly dividing the labeled dataset into k-folds is called partitioning
  - K-fold CV estimates depend on “luck” in partitioning
  - Do multiple, m, partitionings and thus, m K-fold CVs and take the average

Model/Classifier Complexity

- We’ve learned how to compare classifier performance (e.g., AUC, CV)
- A more complex model (e.g., a bigger tree) may have higher performance or not (if overfitting occurs)
- Surely a simpler model is easier to understand
- Where to draw the line?

Model Size and the Use of CV

- Let us use a parameter size to specify complexity, in our case, the maximum number of nodes allowed in a decision tree
- Start with size = 1; use k-fold CV to estimate performance
- Repeat with size = 2, 3, 4, …
- The performance improves initially and then declines after a certain *opt* size (Fig. 18.9)
- This is an example of parameter tuning
- Build the final classifier with *opt* size using all data

Model Size (2)

- Performance need not be just accuracy or AUC
- Can be sensitivity (imbalance case), specificity or any general loss function (Sec 18.4.2)
- Very soon we will talk about *regularization* which explicitly penalizes complex models

Concluding Remarks: Big Data

- K-fold CV is quite expensive but is worth the effort to get an unbiased estimate
- CV is inherently very parallelizable
- Parameter tuning is also inherently very parallelizable
- Thus, even for a big labeled dataset, measuring performance is practical computationally