Decision Trees, Evaluation

Intro to ML (466/566)
Fall 2022

The basis of these slides are courtesy of Alex Thomo. Thanks, Alex!

Administrivia

- 566 students: Need a project group?
  - Post to the Eclass forum
  - Stay after class today to find other groups

Administrivia

- Identifying promising projects:
  - Is this project easy to distribute?
    - Minimize communication overhead
  - Does this project have a core task that seems do-able, as well as some more risky (and interesting!) extensions?
  - Does the data already exist?
    - I <u>highly suggest</u> you use available data
    - If not, are the resources to get/create the data available? A project that gets stuck in the data phase won’t be successful.
  - Predicting the stock market
    - Not an easy task, which can be discouraging.
    - Difficult to know if you’re doing “the right thing” or if your code is just buggy.
    - More difficult to write a project report when nothing works

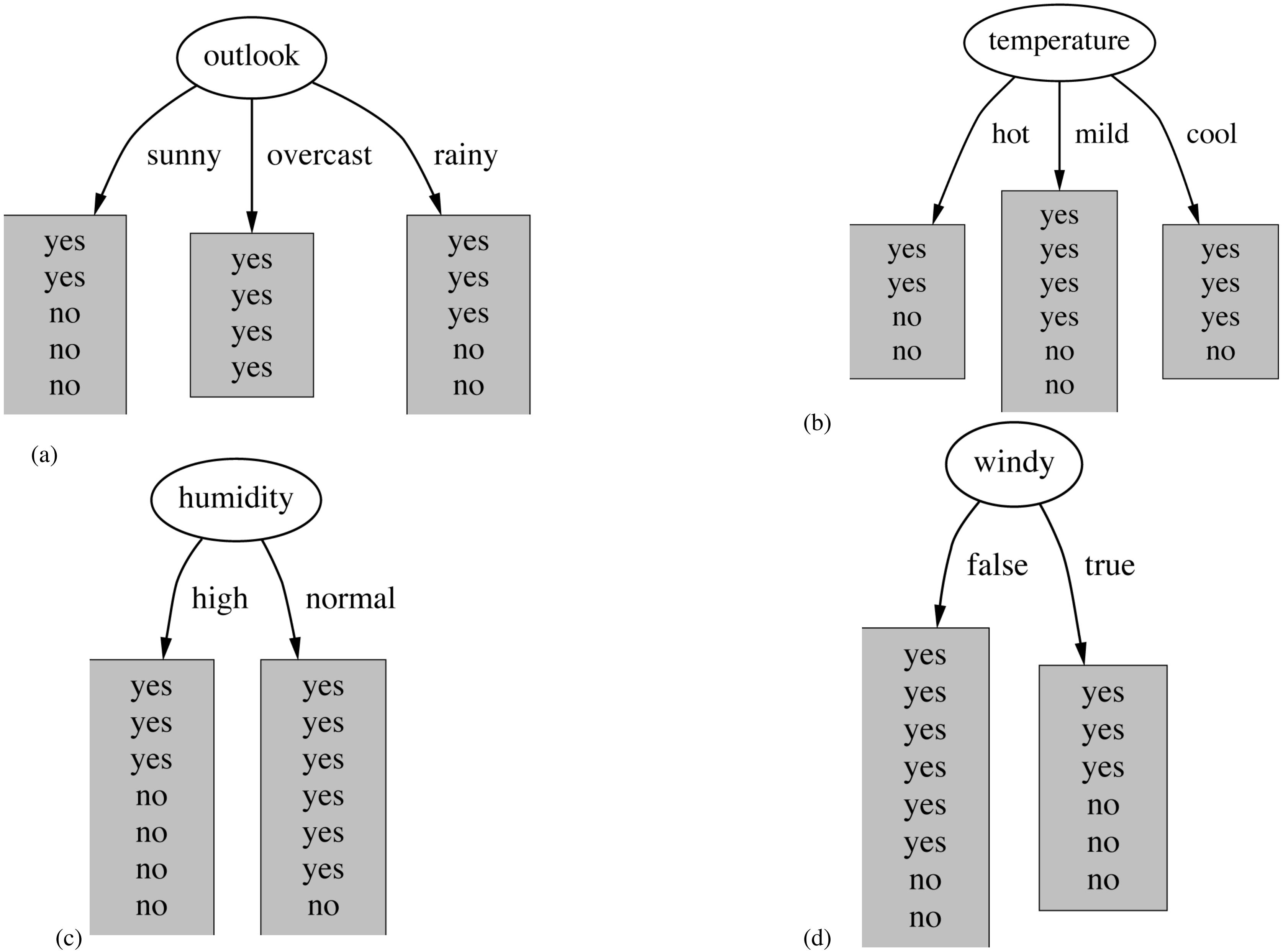
Recap from last time

- Decision Trees and Entropy

Example: Playing soccer

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Which attribute to select?



A criterion for attribute selection

- Which is the best attribute?

- The one which will result in the *smallest* tree (fewest decisions)
  - Heuristic: choose the attribute that produces the "purest" nodes

- Popular impurity criterion: **entropy** of nodes
  - **Lower the entropy, purer the node**.

Entropy

- H(X) = E(I(X)) **Expected** value of the **information** in X
- Expected value:

$$E(f(X)) = \sum_i P(x_i) * f(x_i)$$

- Information:

$$I(x_i) = -\log_2 P(x_i)$$

- Entropy:

$$H(X) = E(I(X)) = \sum_i P(x_i) I(x_i) = -\sum_i P(x_i) \log_2 P(x_i)$$

- **Strategy: choose attribute that results in lowest entropy of the children nodes.**

Measuring Purity with Entropy

- Entropy is a measure of **disorder.** Also called **amount of information**.
  - The higher the entropy, the messier the bag
  - The lower the entropy, the purer the bag

yes
yes
yes
no
no
no
no

high entropy - BAD

yes
yes
yes
yes
yes
yes
no

Low entropy - GOOD

yes
yes
yes
yes

Zero entropy - PERFECT

$$E(a/d, b/d) = -(a/d)*\log_2(a/d) - (b/d)*\log_2(b/d)$$

where:

d=total # of rows

a = # yes

b = # no

yes
yes
yes
no
no
no
no

yes
yes
yes
yes
yes
yes
no

yes
yes
yes
yes

$(0/4)*\log_2(0/4)$ = **$0*\log_2(0)$** is **indeterminate**. We consider it to be 0.

$E(4/4,0/4) =$
$-(4/4)*\log_2(4/4)-(0/4)*\log_2(0/4) =$ **0**

$E(3/7,4/7) =$
$-(3/7)*\log_2(3/7)-(4/7)*\log_2(4/7) =$ **.985**

$E(6/7,1/7) =$
$-(6/7)*\log_2(6/7)-(1/7)*\log_2(1/7) =$ **.5917**

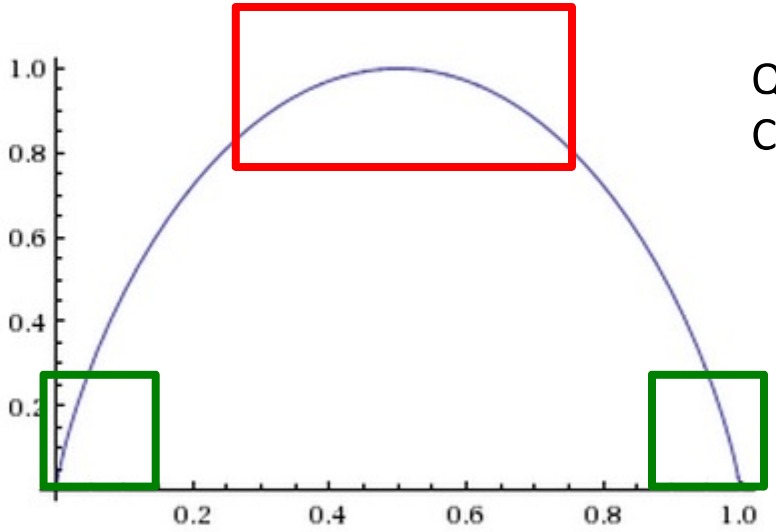
Entropy Chart

- In the entropy formula: a/d + b/d = 1
- Denote

  a/d with x

  b/d with 1-x.
- $E(a/d, b/d) = -(a/d)*\log_2(a/d) - (b/d)*\log_2(b/d) =$

$$-x*\log_2(x) - (1-x)*\log_2(1-x)$$



Question to think about:
Can entropy be larger than 1?

Entropy for more than two class values

For three class values:

$E(a/d, b/d, c/d) = -(a/d)*\log_2(a/d) - (b/d)*\log_2(b/d) - (c/d)*\log_2(c/d)$

$a/d + b/d + c/d = 1$

For more class values:

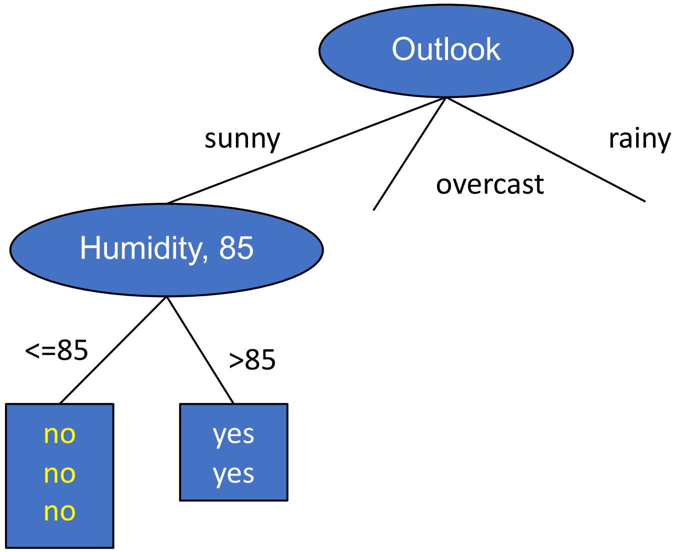
$E(a_1/d,\ldots, a_n/d) = -(a_1/d)*\log_2(a_1/d) - \ldots - (a_n/d)*\log_2(a_n/d)$

$a_1/d + \ldots + a_n/d = 1$

Continuous-valued attributes

- Some attributes can be numeric (continuous).
- No problem, we can have binary splits (≥v, <v), still use Entropy

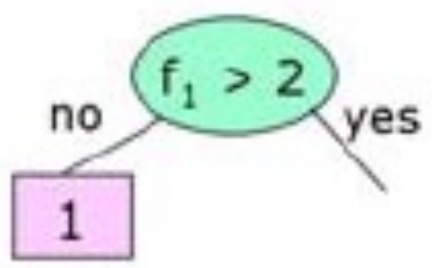
ID	Outlook	Temp	Humidity	Windy	Play
1	sunny	85	85	false	no
2	sunny	80	90	true	no
3	overcast	83	86	false	yes
4	rainy	70	96	false	yes
5	rainy	68	80	false	yes
6	rainy	65	70	true	no
7	overcast	64	65	true	yes
8	sunny	72	95	false	no
9	sunny	69	70	false	yes
10	rainy	75	80	false	yes
11	sunny	75	70	true	yes
12	overcast	72	90	true	yes
13	overcast	81	75	false	yes
14	rainy	71	91	true	no

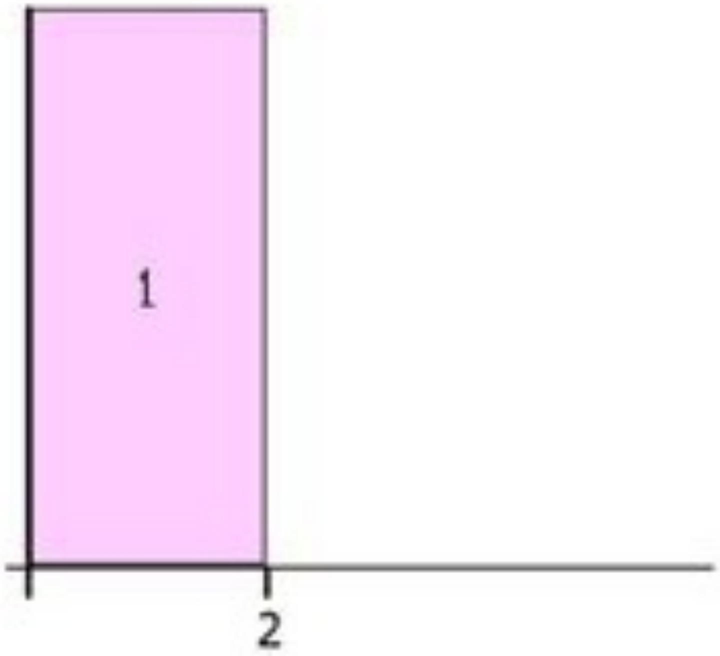


ID	Outlook	Temp	Humidity	Windy	Play
1	sunny	69	70	false	no
2	sunny	75	70	true	no
8	sunny	85	85	false	no
9	sunny	80	90	false	yes
11	sunny	72	95	true	yes

Numerical attributes revisited

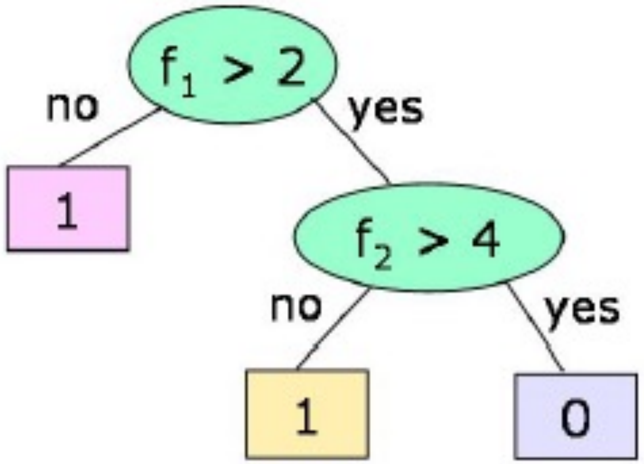
- Tests in nodes are of the form f_i > constant

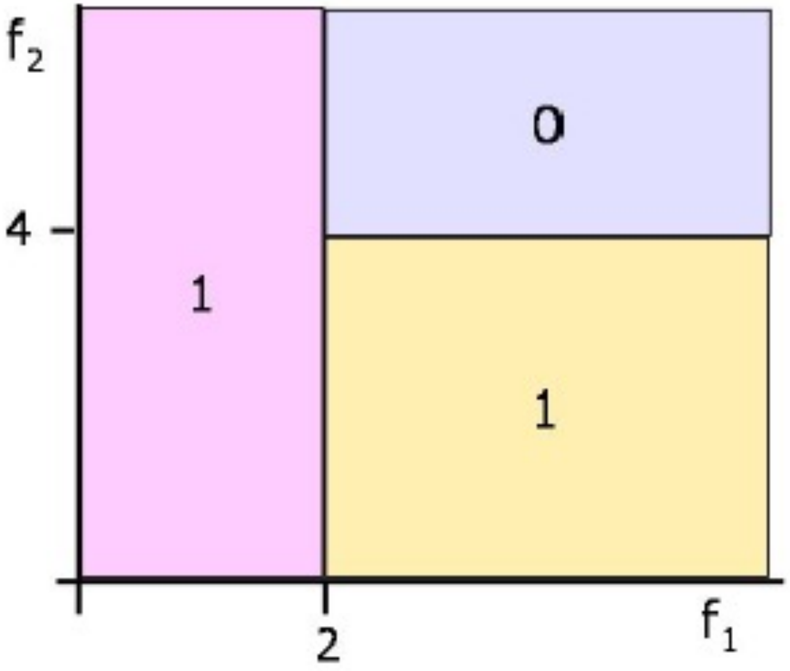




Numerical attributes

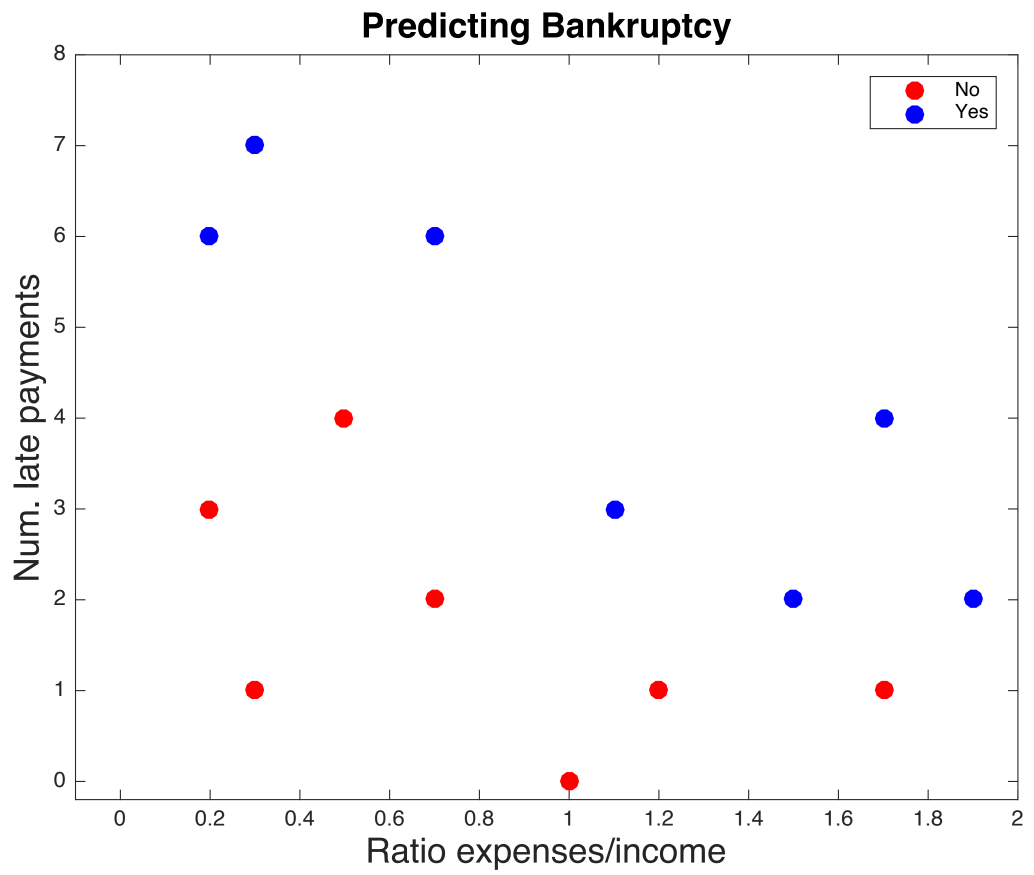
- Tests in nodes are of the form f_i > constant
- Divides the space into rectangles.





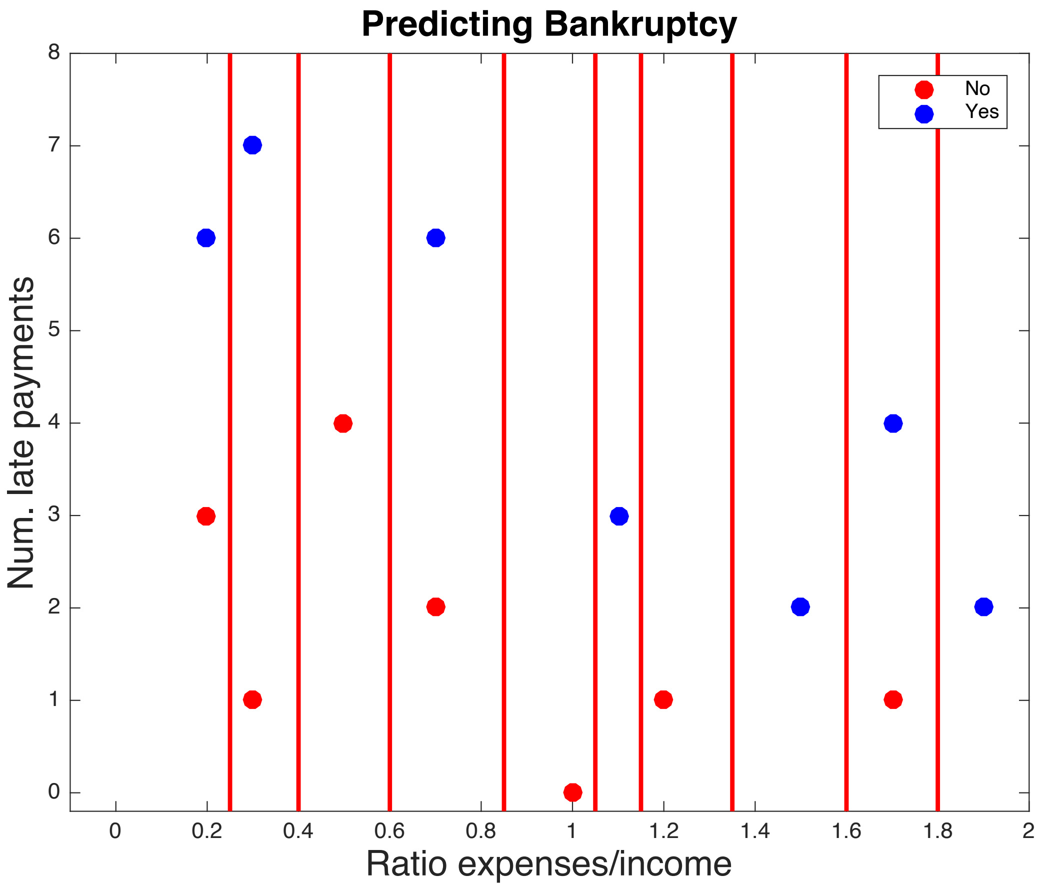
Predicting Bankruptcy

Late	Ratio	Bankruptcy?
3	0.2	No
1	0.3	No
4	0.5	No
2	0.7	No
0	1.0	No
1	1.2	No
1	1.7	No
6	0.2	Yes
7	0.3	Yes
6	0.7	Yes
3	1.1	Yes
2	1.5	Yes
4	1.7	Yes
2	1.9	Yes



Considering splits

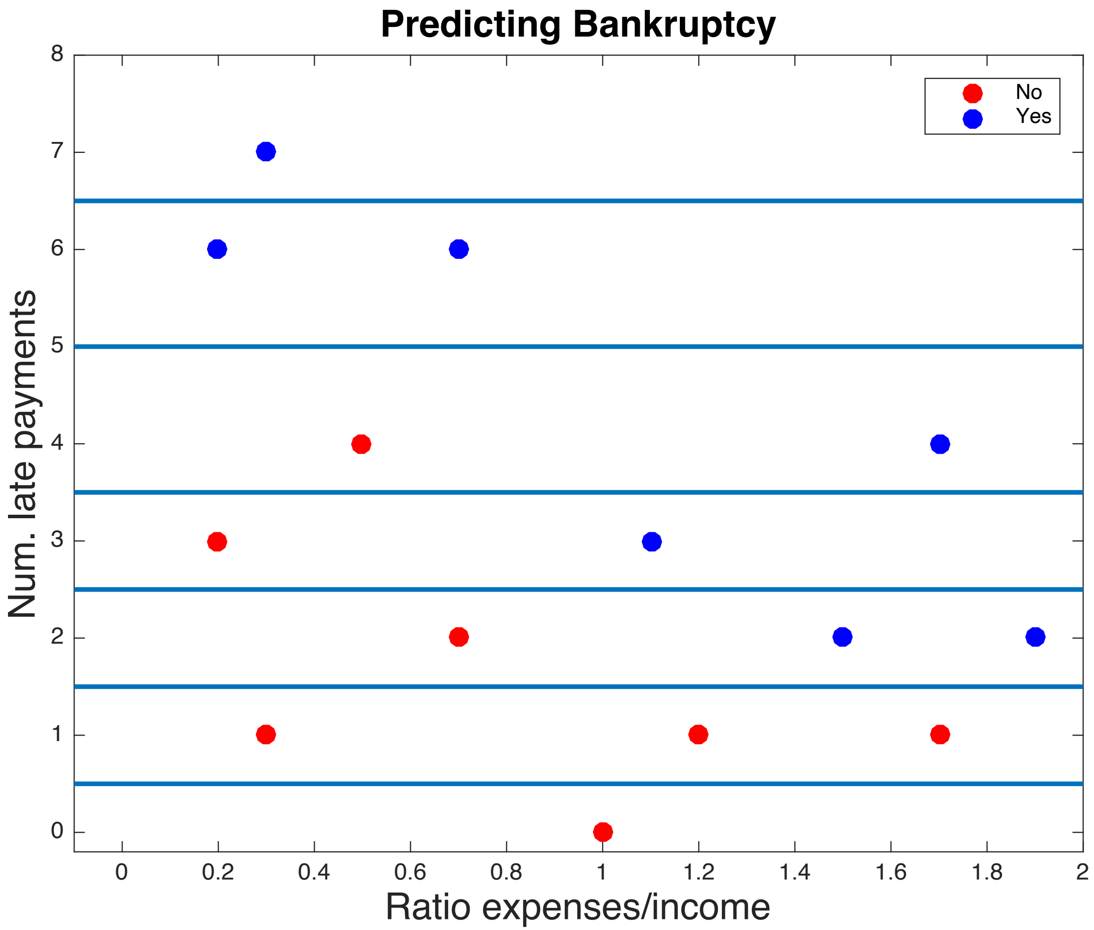
- Consider splitting between each data point in each



- So, here we'd consider 9 different splits in the ratio dimension

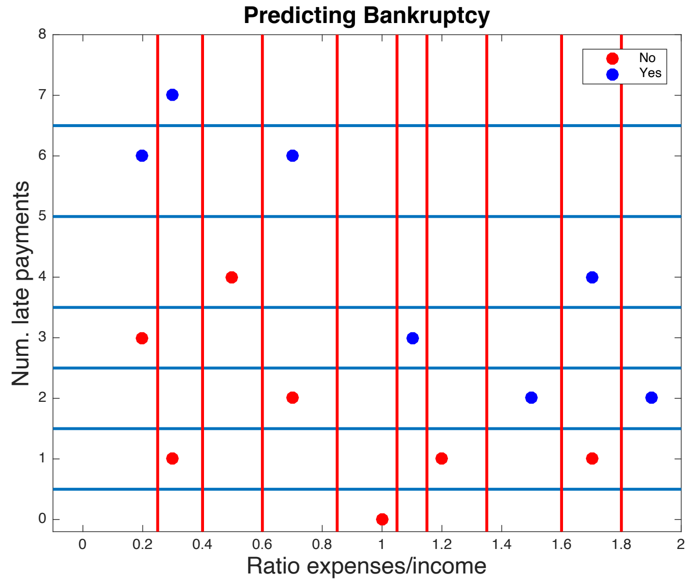
Considering splits II

- And there are another 6 possible splits in the late payments dim



Bankruptcy Example

thresh	Neg. less	Pos. less	Neg. greater	Pos. greater	AE
6.5	7	6	0	1	0.92
5.0	7	4	0	3	0.74
3.5	6	3	1	4	0.85
2.5	5	2	2	5	0.86
1.5	4	0	3	7	0.63
0.5	1	0	6	7	0.92



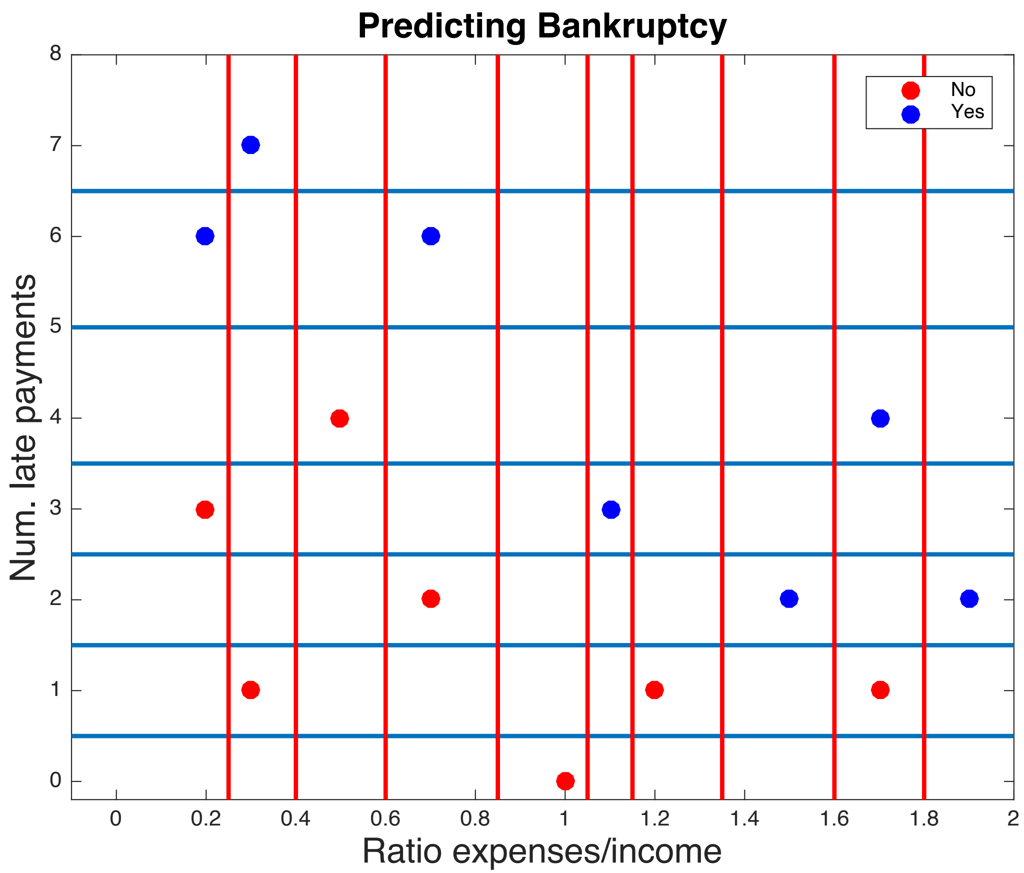
For the first and last row

info[7,6] = (-(7/13)*log2(7/13)-(6/13)*log2(6/13)) = .9957

info[0,1] = (-(0/1)*log2(0/1)-(1/1)*log2(1/1)) = 0

info([7,6],[0,1]) = .9957*13/14 + 0*1/14 = .92

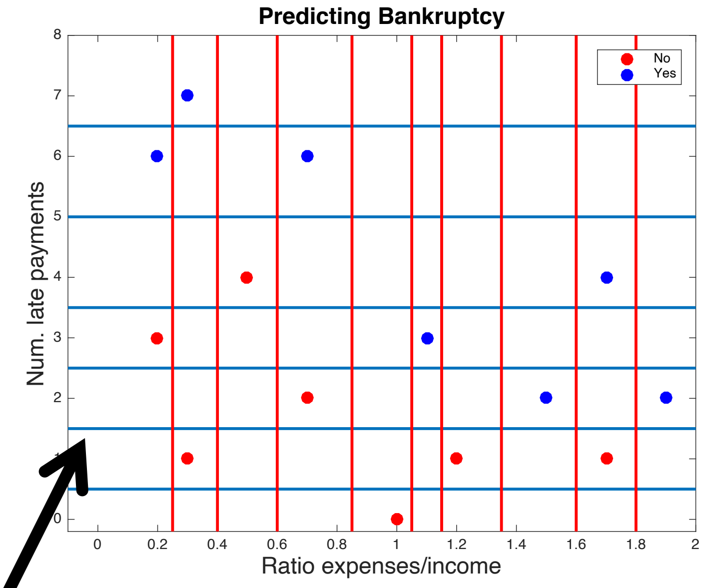
Bankruptc

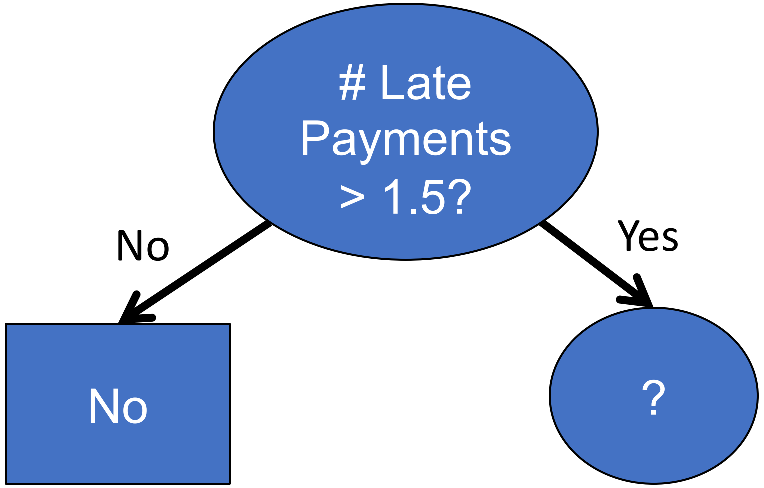


thresh	**0.25**	**0.4**	**0.6**	**0.85**	**1.05**	**1.15**	**1.35**	**1.6**	**1.8**
AE	1	1	0.98	0.98	0.94	0.98	0.92	0.98	0.92

Bankruptcy Example

thresh	Neg. less	Pos. less	Neg. greater	Pos. greater	AE
6.5	7	6	0	1	0.92
5.0	7	4	0	3	0.74
3.5	6	3	1	4	0.85
2.5	5	2	2	5	0.86
1.5	4	0	3	7	0.63
0.5	1	0	6	7	0.92

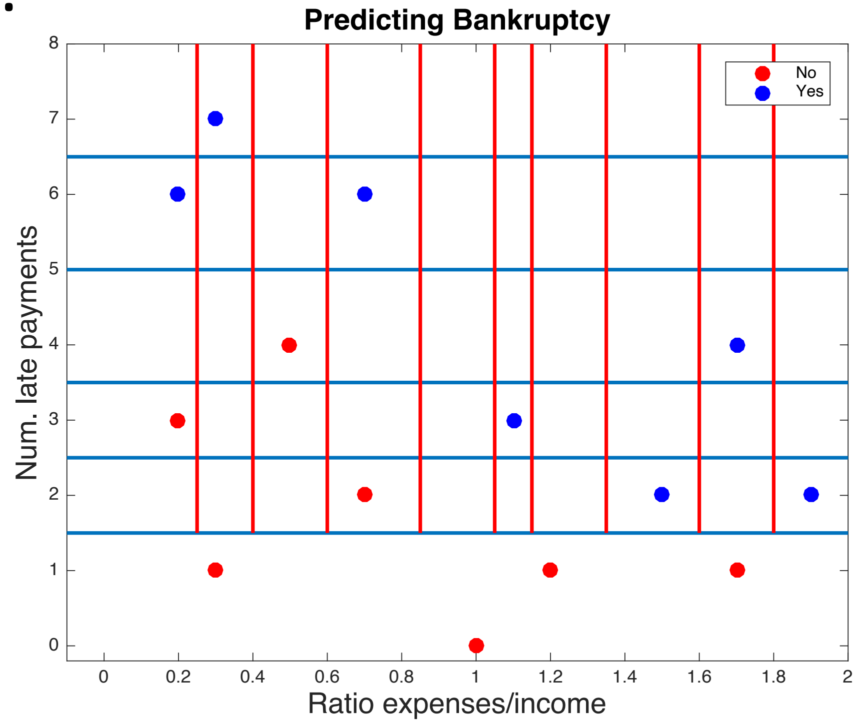




Bankruptcy Example

- Now, recurse on data points with num. late payments >= 1.5
- Can we just reuse these tables?

thresh	Neg. less	Pos. less	Neg. greater	Pos. greater	AE
6.5	7	6	0	1	0.92
5.0	7	4	0	3	0.74
3.5	6	3	1	4	0.85
2.5	5	2	2	5	0.86
1.5	4	0	3	7	0.63
0.5	1	0	6	7	0.92

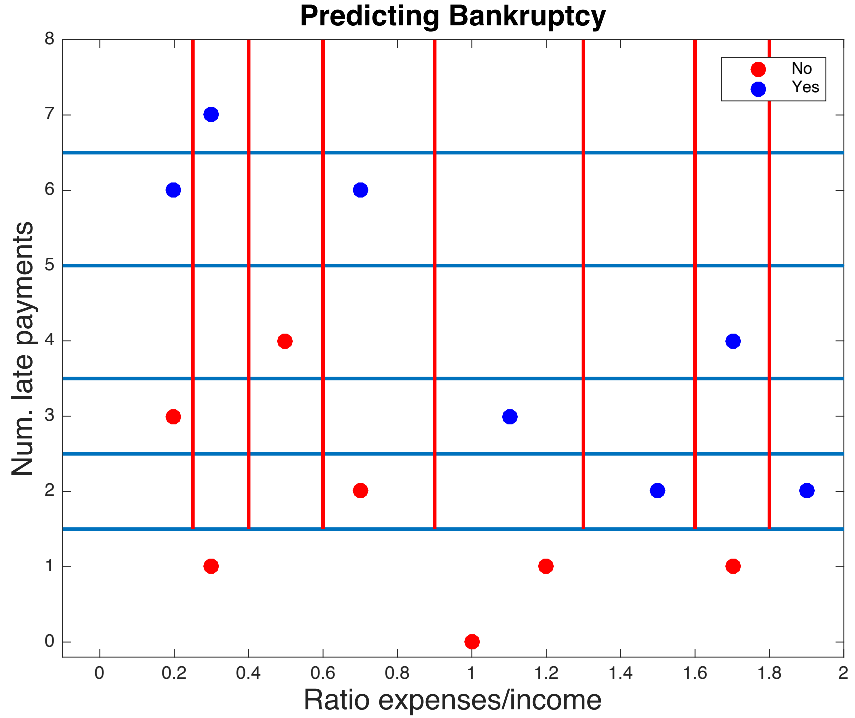


thresh	0.25	0.4	0.6	0.85	1.05	1.15	1.35	1.6	1.8
AE	1	1	0.98	0.98	0.94	0.98	0.92	0.98	0.92

Bankruptcy Example

- Have to make new tables

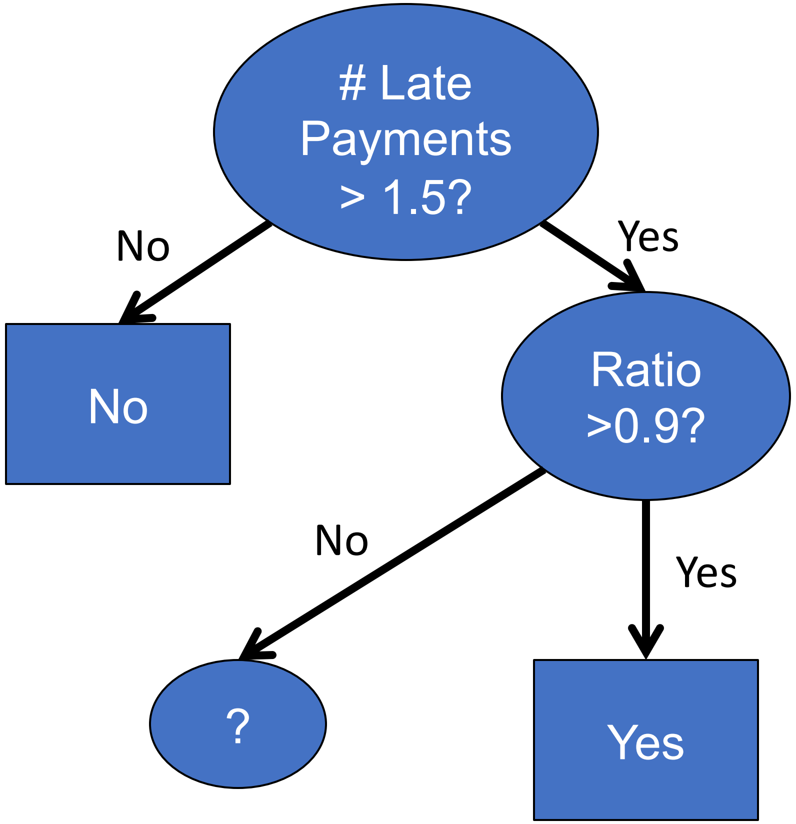
thresh	Neg. less	Pos. less	Neg. greater	Pos. greater	AE
6.5	6	3	0	1	0.83
5.0	4	3	0	3	0.69
3.5	3	2	4	1	0.85
2.5	2	1	5	2	0.88

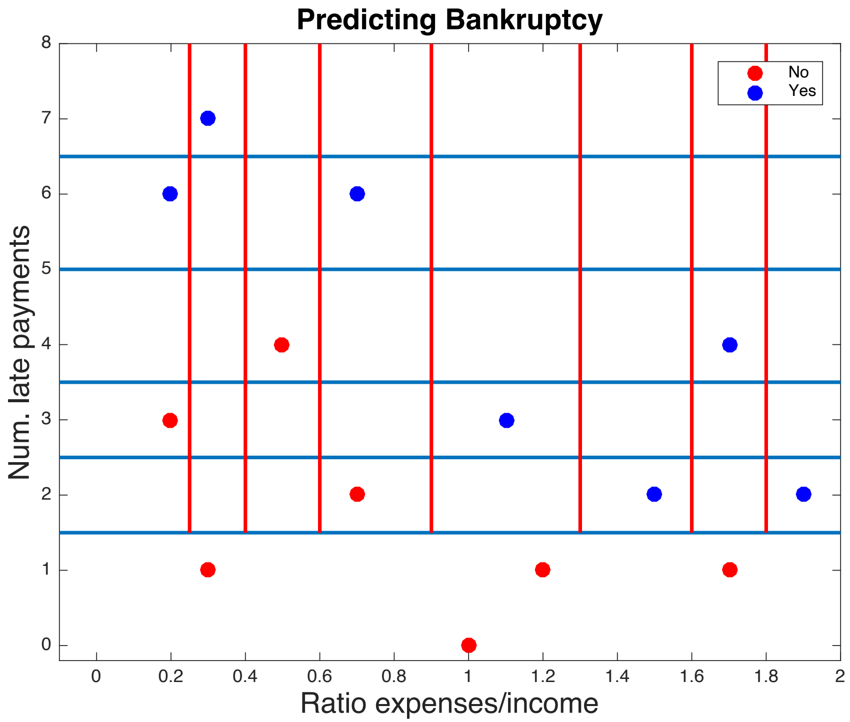


thresh	0.25	0.4	0.6	0.9	1.3	1.6	1.8
AE	0.85	0.88	0.79	0.6	0.69	0.76	0.83

Bankruptcy Example

- Have to make new tables

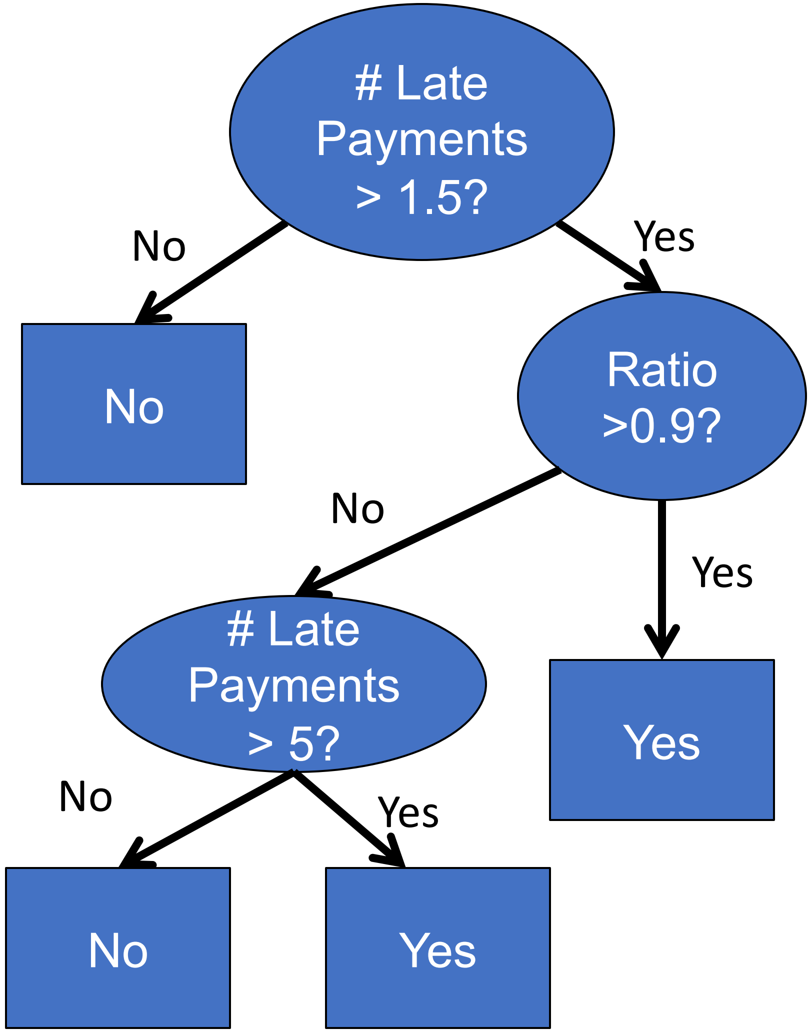


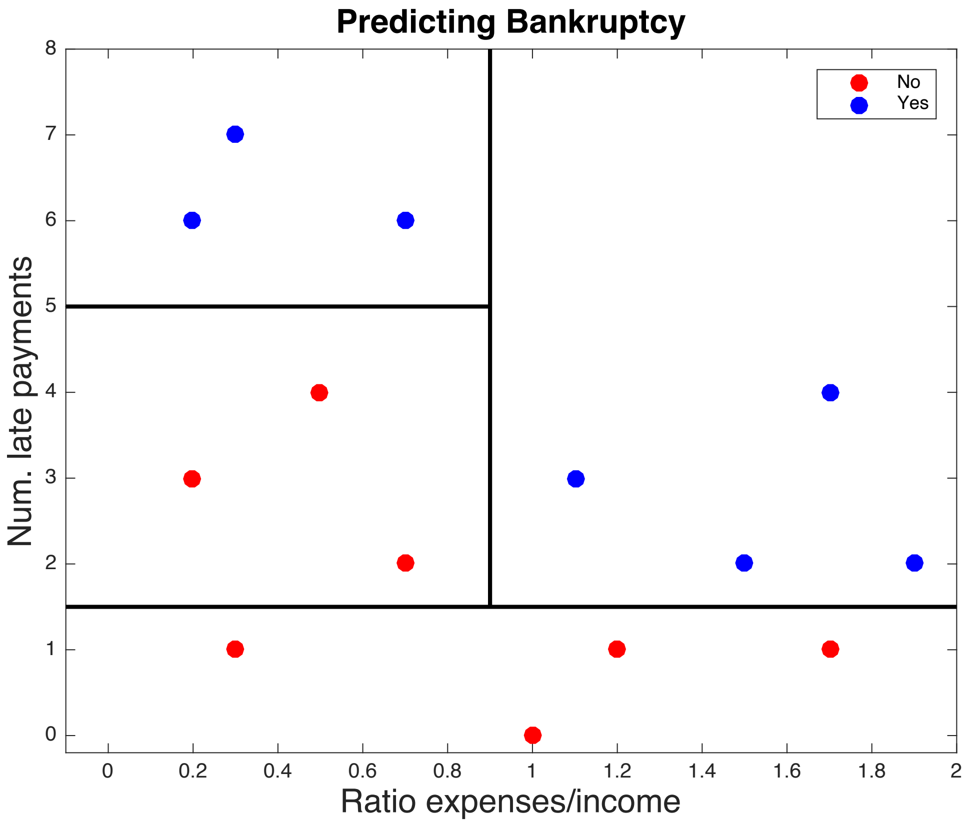


thresh	0.25	0.4	0.6	0.9	1.3	1.6	1.8
AE	0.85	0.88	0.79	0.6	0.69	0.76	0.83

Bankruptcy Example

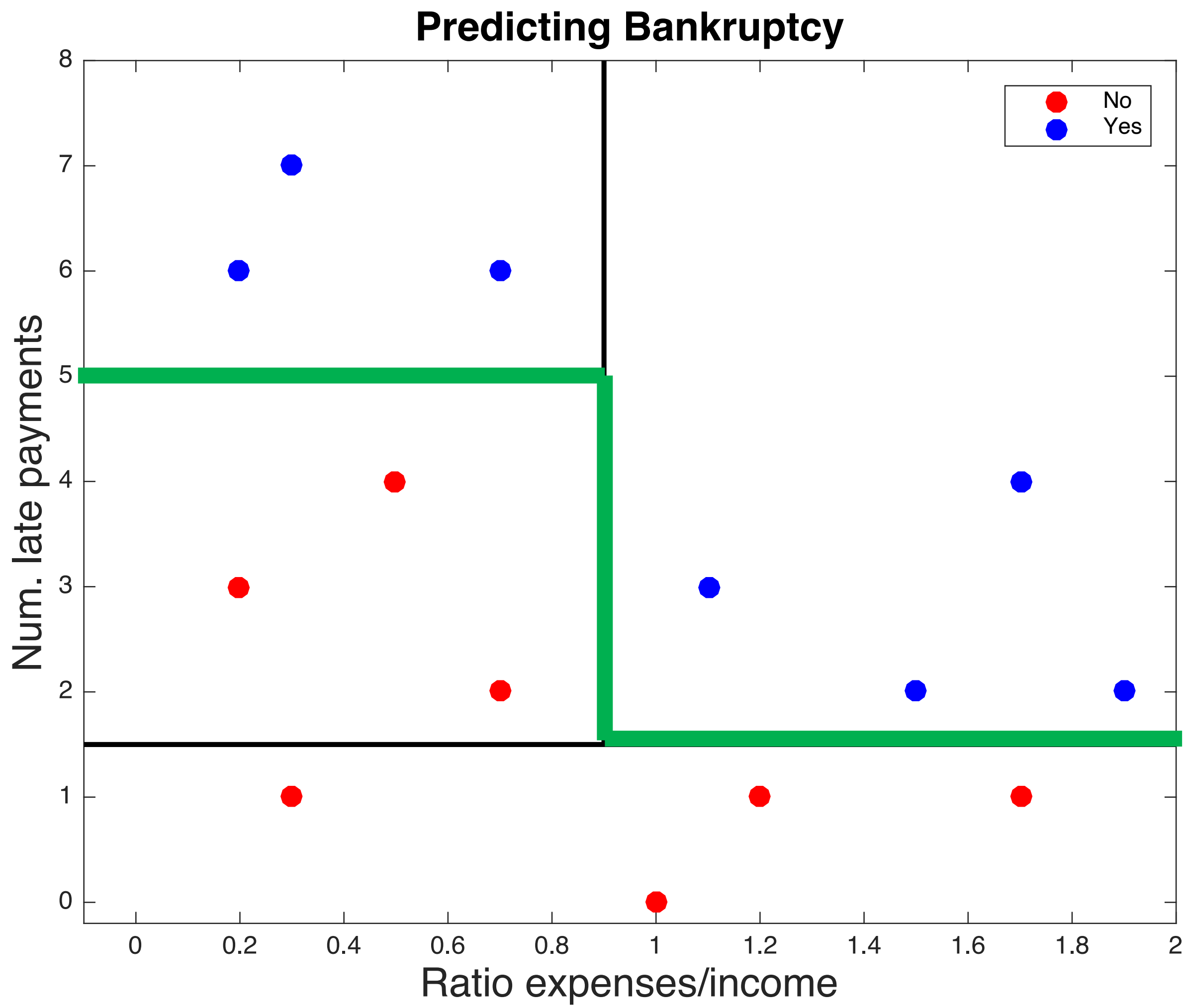
- Continue to obtain this tree:





Decision trees learn non-linear decision boundaries

- Can perform well when a non-linear boundary is required
- Can also overfit
  - (we will talk more about this shortly)

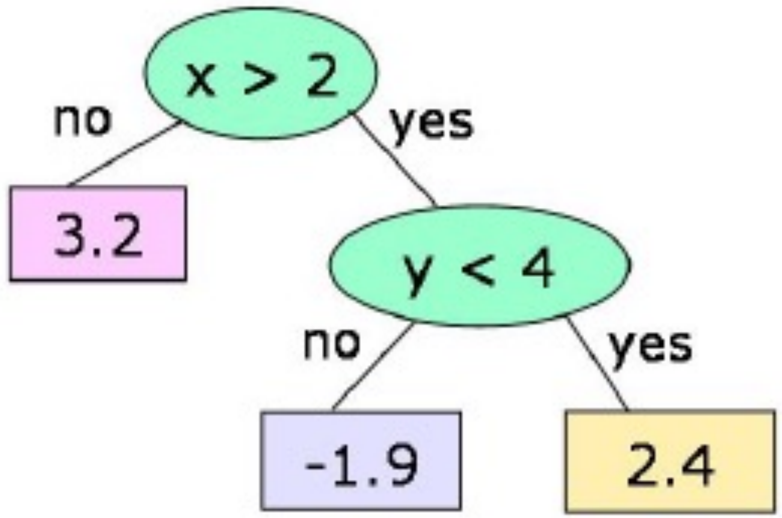


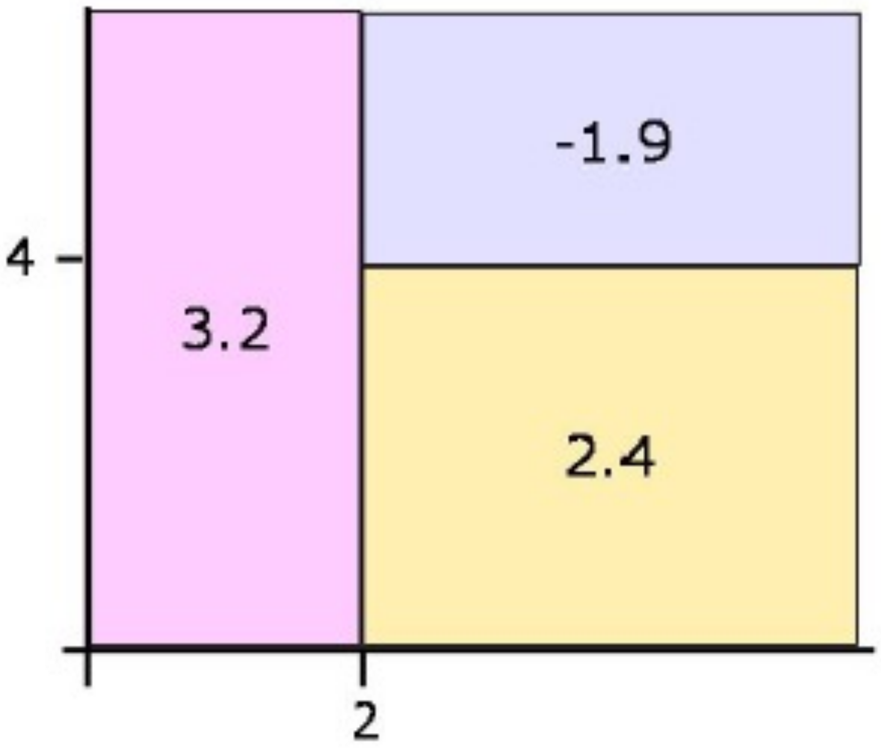
Classification vs Regression

- So far we have described classification
  - predicting one of a discrete set of labels
    - play tennis? yes/no
    - Neighbor’s behavior: walk/drive
    - Car type: luxury, mini, sports, van
- Sometime we want to predict a number in a range
  - E.g. age, forecast temperature, etc…
  - That is called ***regression*** (predicting a real number)

Regression Trees

- Like decision trees, but with real-valued constant outputs at the leaves.





Things to consider

- Prediction is a real number
  - Thus training labels are real numbers
  - Instead of {yes, yes, yes, no} at a leaf, we will have something like {0.1, 0.5, 1.3, 0.8}
- How to evaluate a candidate split?
  - How do we measure “purity” in real-valued numbers?
  - **How pure is {0.1, 0.5, 1.3, 0.8}?**

Things to consider

- What to predict based on the instances in a leaf node?
  - Since labels are continuous, most datapoints won’t have the same label
  - **What should you predict if a node contains {0.1, 0.5, 1.3, 0.8}**

Things to consider

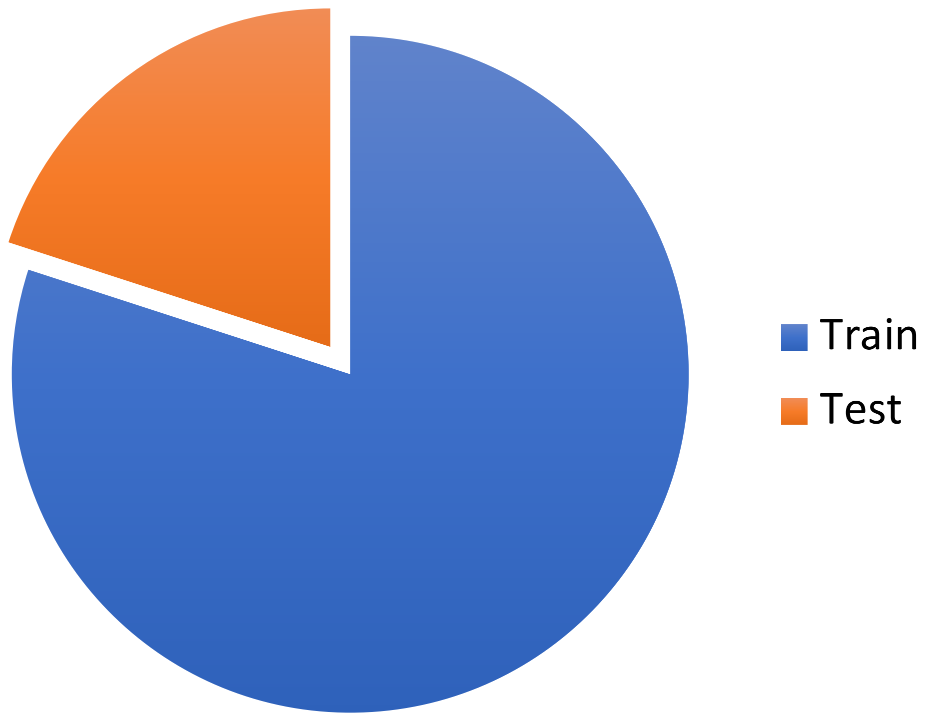
- When to stop splitting?
  - If datapoints don't have exactly the same label, if we spilt to perfect purity we'll end up with only one training example in each node
    - We'll end up with 4 leaf nodes: {0.1}, {0.5}, {1.3}, {0.8}
  - May not be good for **generalization**
- What could we measure to decide when to stop splitting?
  - Hint: What did we do previously with classification trees?

Pruning

- How can we reduce overfitting in our decision trees?
  - make the trees smaller (less complex)

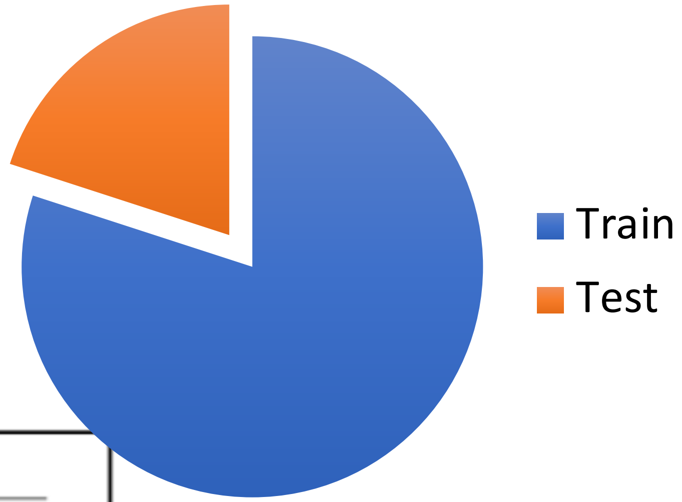
Overfitting

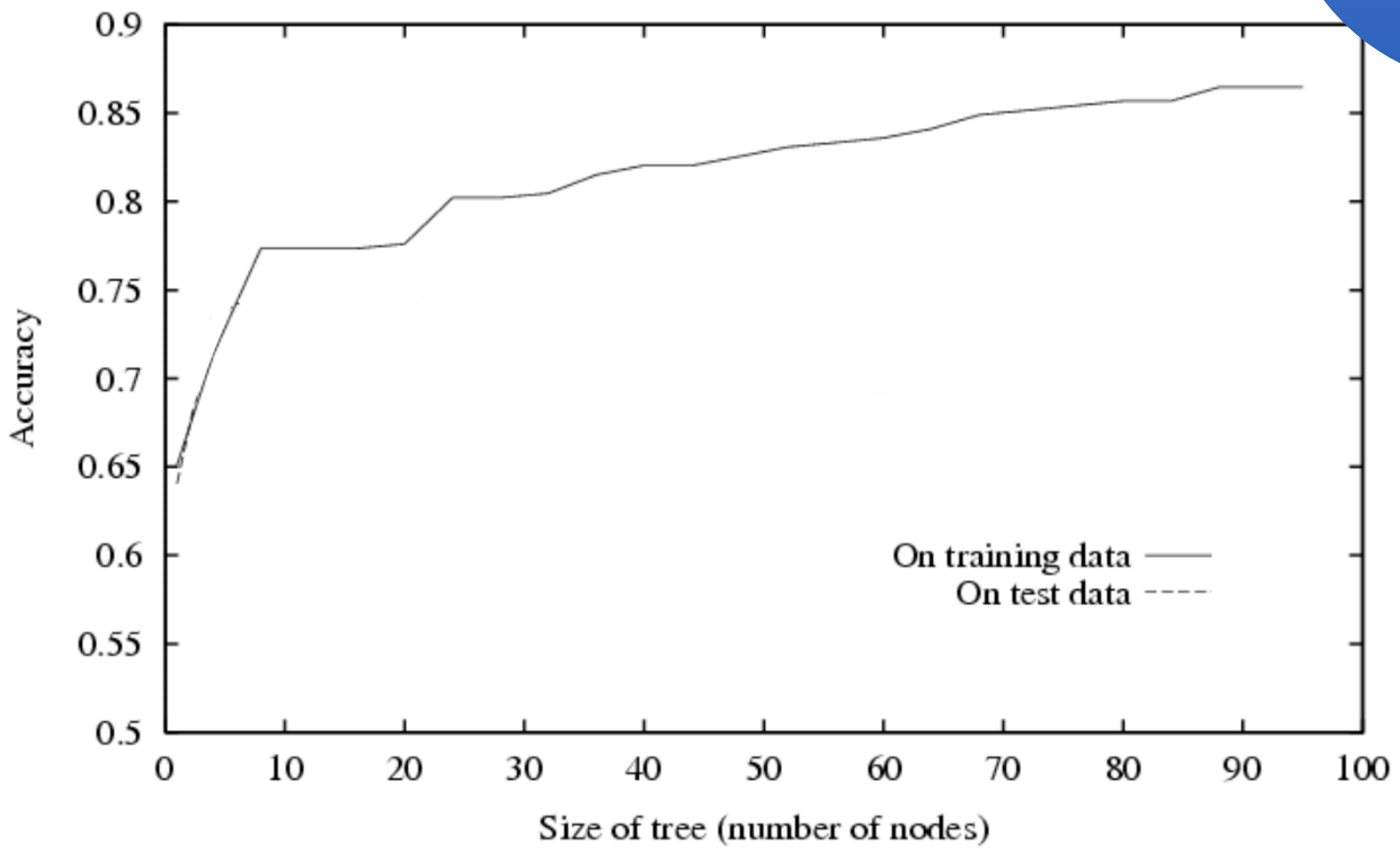
- What is overfitting?
- What does overfitting look like?



Overfitting

- What is overfitting?
- What does overfitting look like?





Why is it overfitting?

- Where does overfitting come from?
  - noise in labels
  - noise in features
  - too little data (lack of representative data)
  - model-data mismatch

Back to Decision Trees…

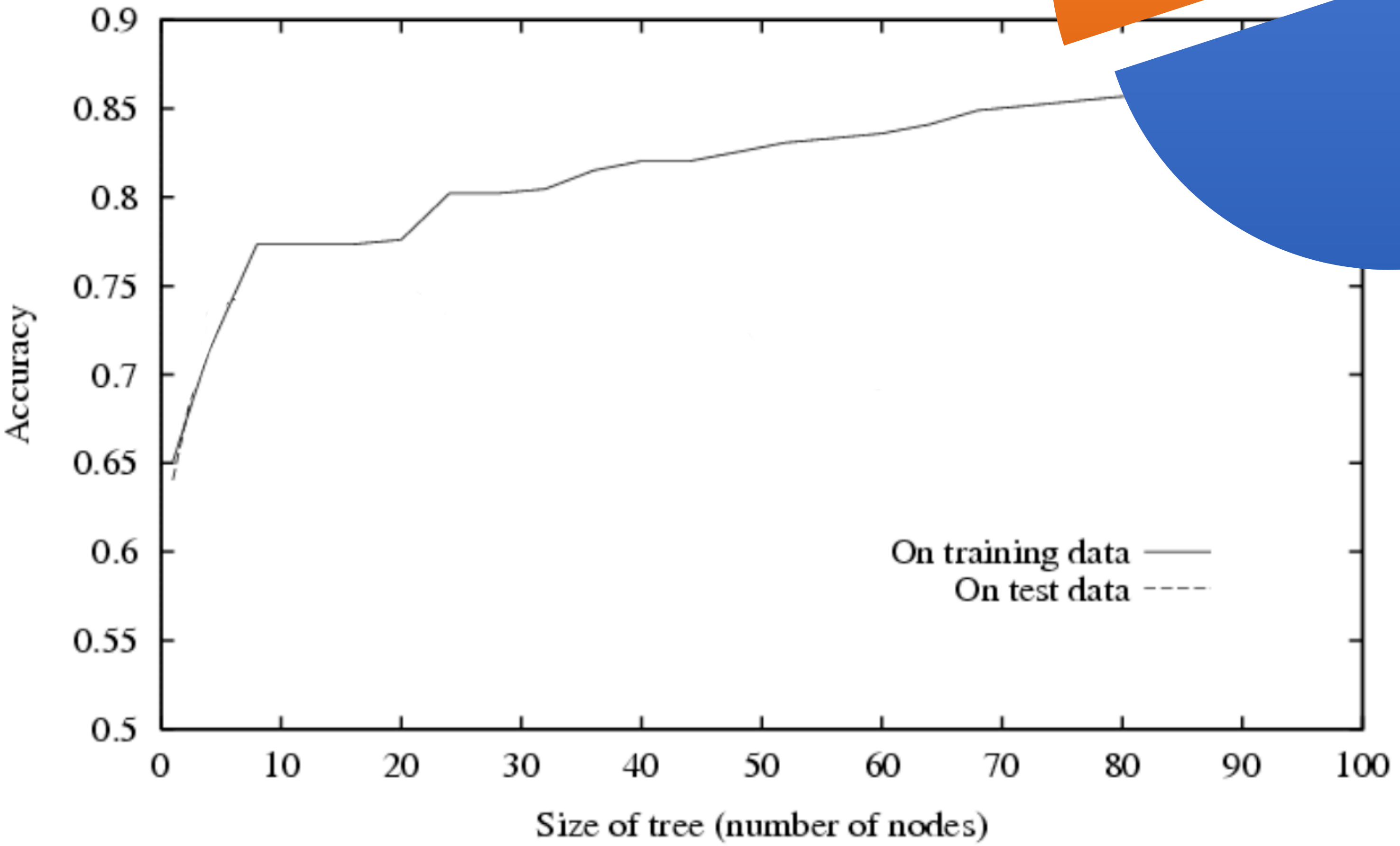
- How can we avoid overfitting?

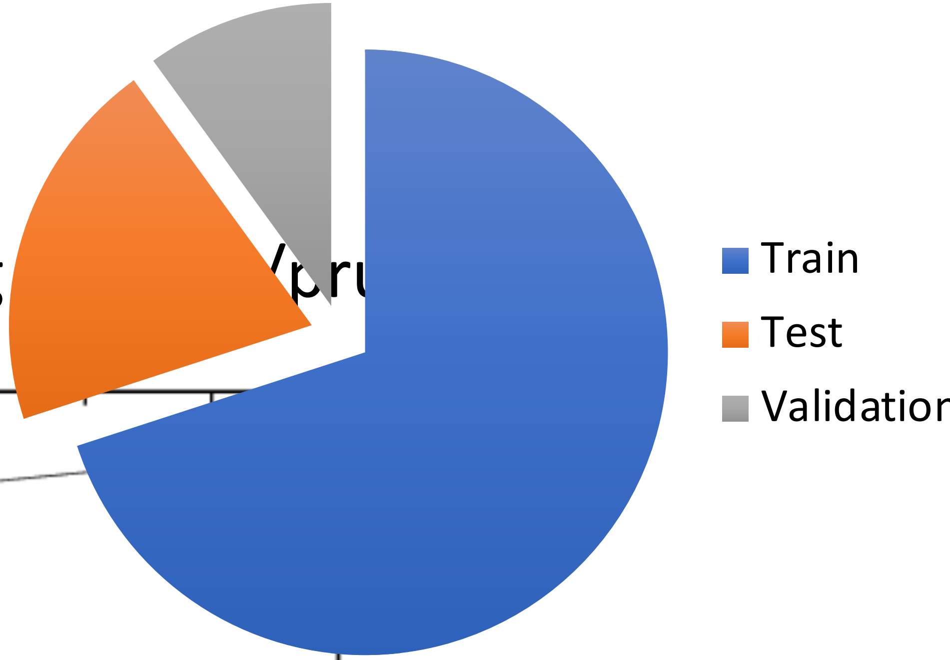
Early Stopping (Pre-prune)

- Stop making splits when
  - average entropy doesn’t change much
  - Predefined # of training instances reach leaf
  - Predefined depth

Post pruning

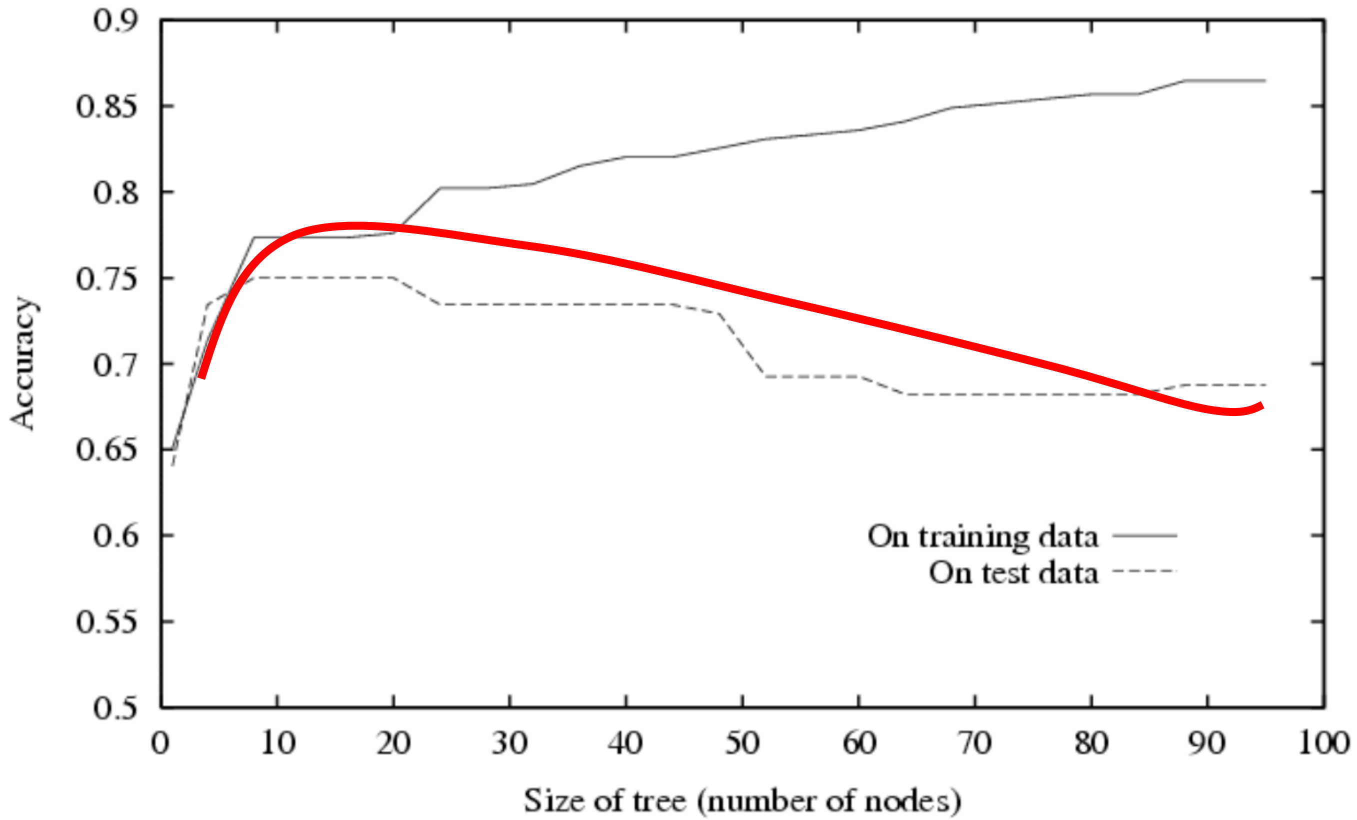
- Build a complex tree, then simplify
- It’s cheating to check test set accuracy during /pru



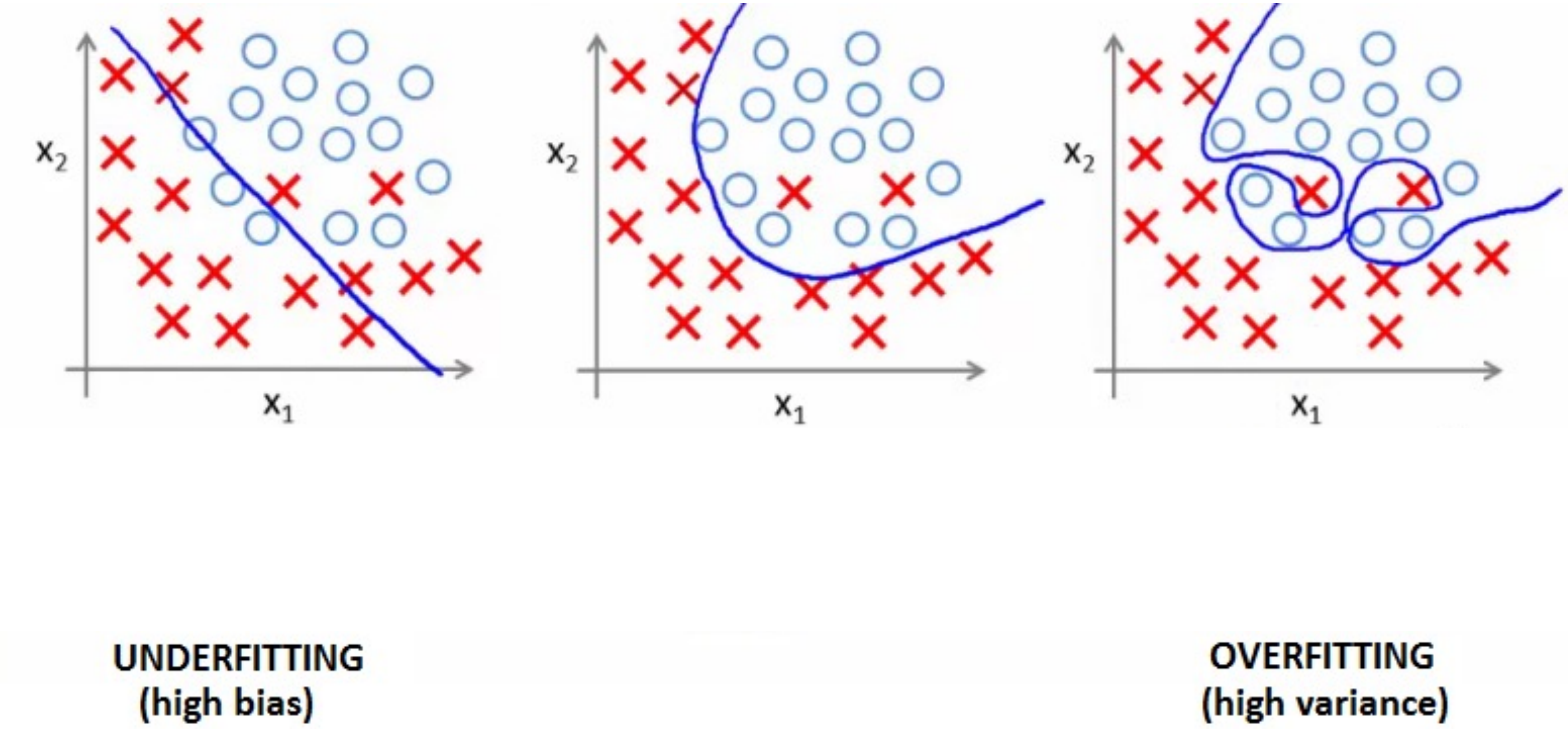


Validation Set

- Select some part of your training data for validation (tuning)
  - Don’t use it to train
  - Choose depth based on validation performance peak
    - (hopefully similar to test data peak)



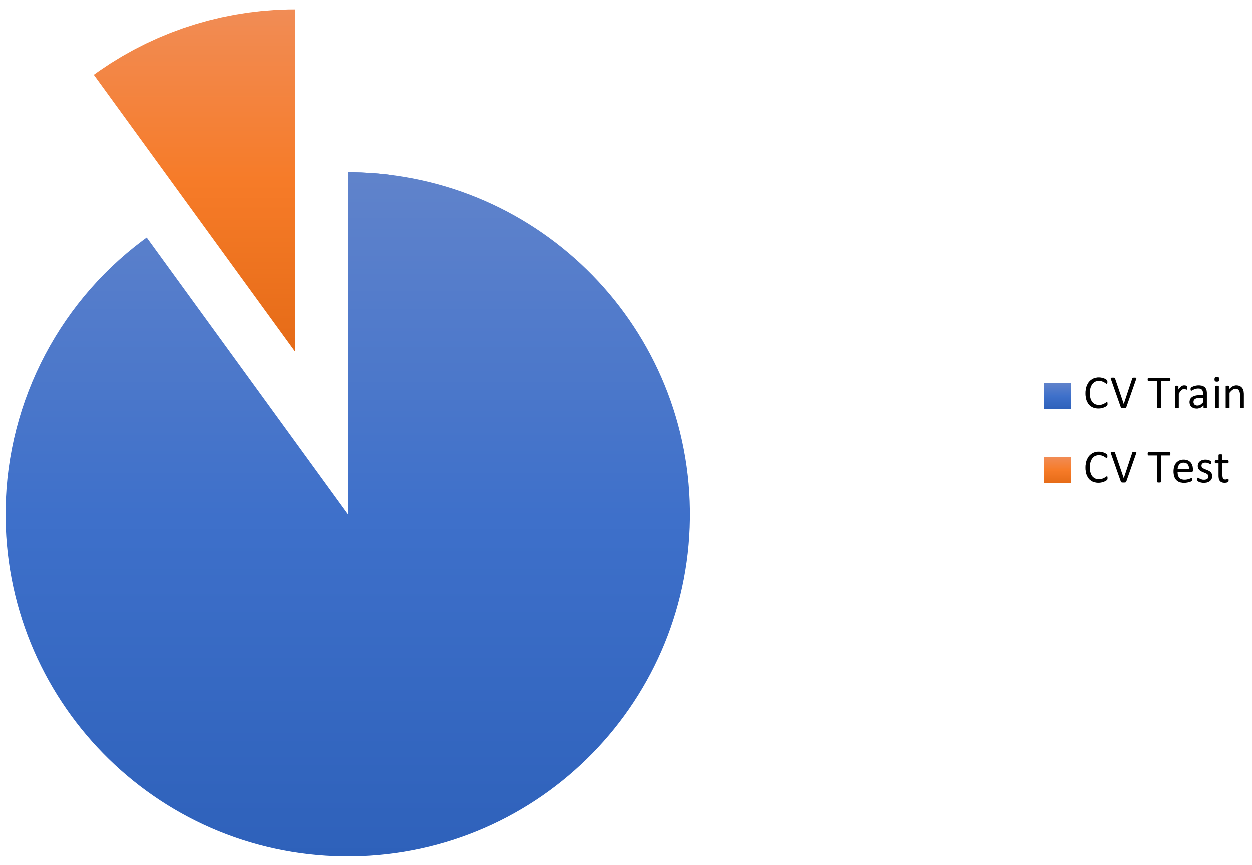
Over vs Under fitting



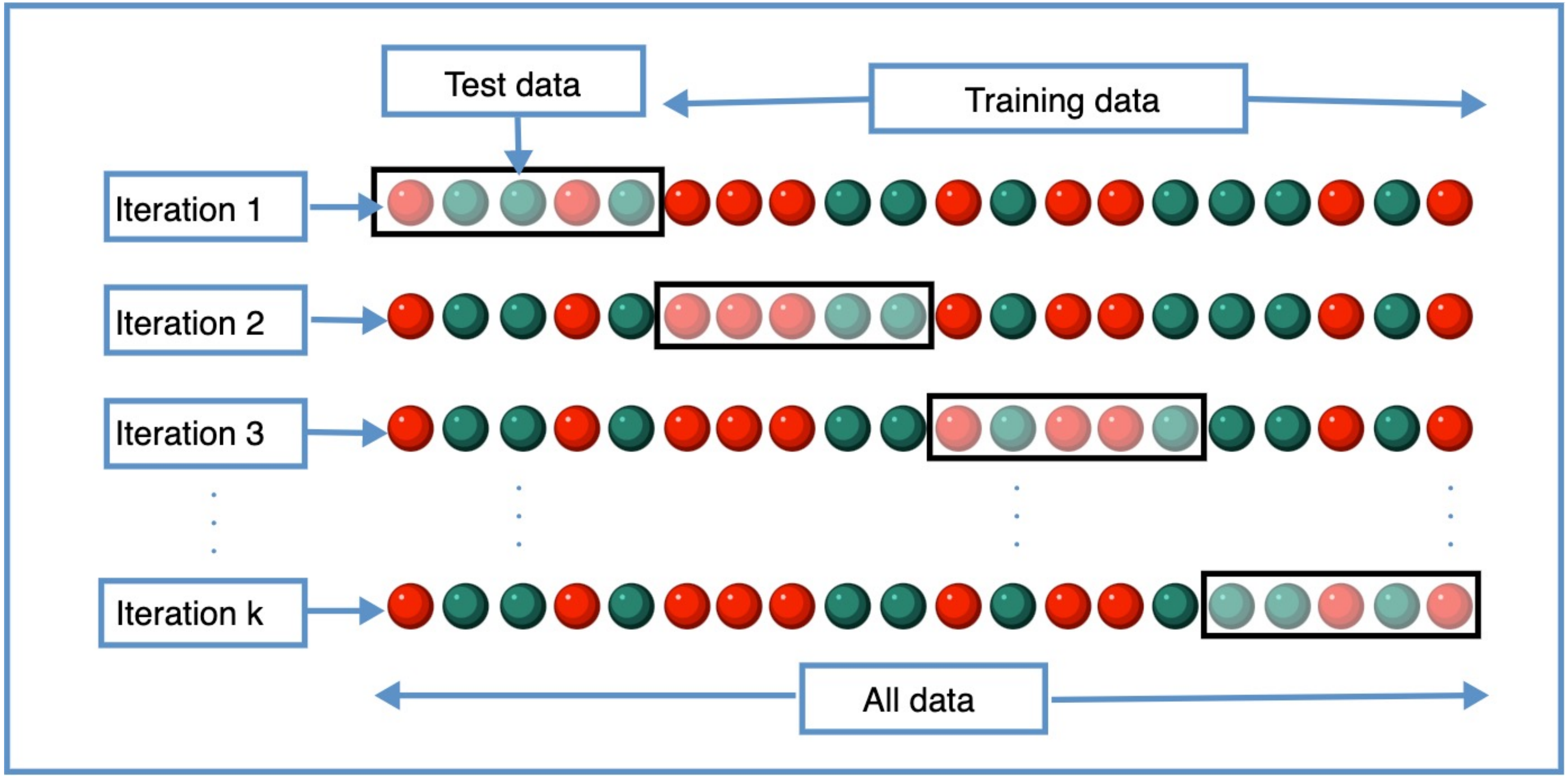
Model Selection

- Suppose we are trying to select among several different models for a learning problem.
- E.g.
  - Full tree vs. tree pruned to depth 5 vs. random forest?

Cross Validation



Cross Validation



Practical issues for CV

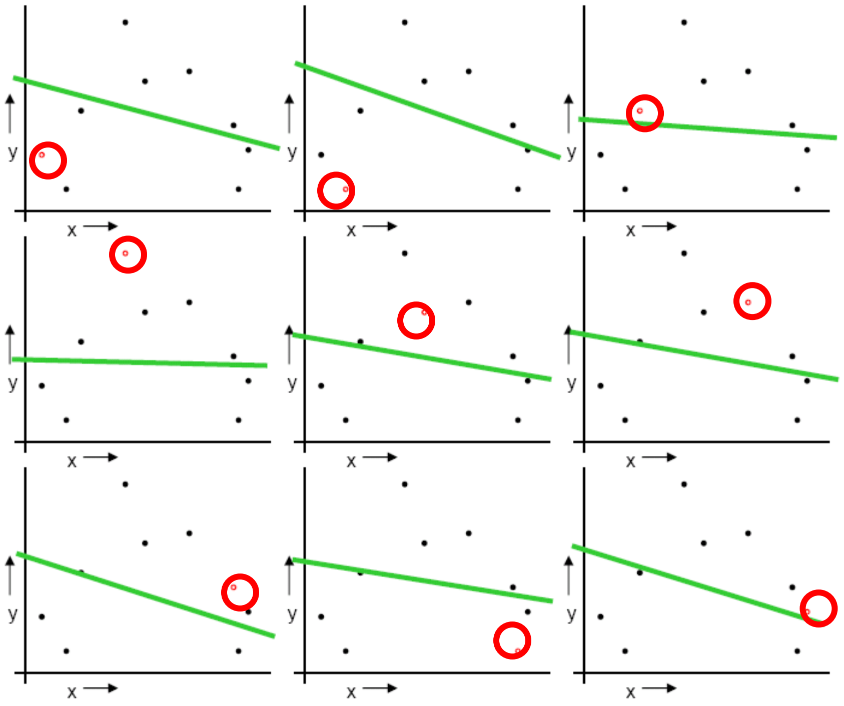
- How to big of a slice of the pie?
  - Commonly used K = 10 folds (thus each fold is 10% of the data)
  - Leave-one-out-cross-validation LOOCV (K=N, number of training instances)

- One important point is that (for a particular fold) the test data is never used for training, because doing so would result in overly (indeed dishonest) optimistic accuracy rates during the testing phase.

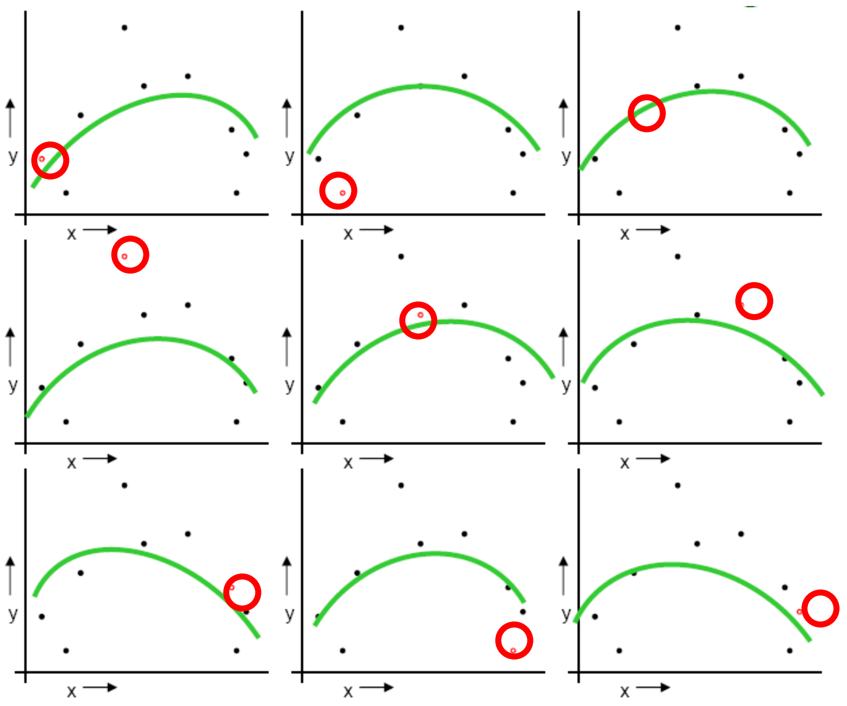
- Stratification – should you balance the classes across the folds?

Example:

- When $k=N$, the algorithm is known as Leave-One-Out-Cross-Validation (LOOCV)



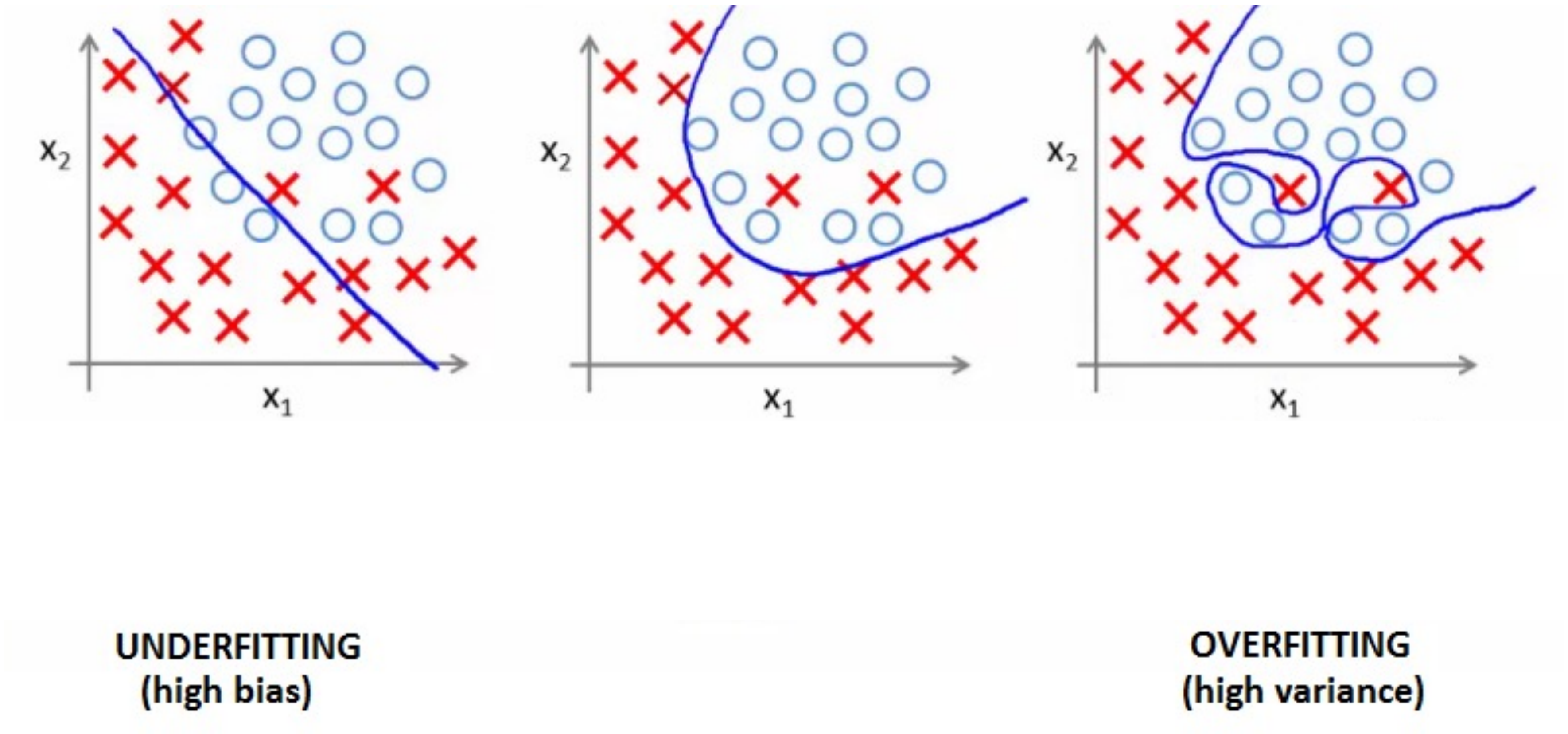
MSELOOCV(M_1)=2.12



MSELOOCV(M_2)=0.962

MSE explained later in these slides

Why is CV so important?



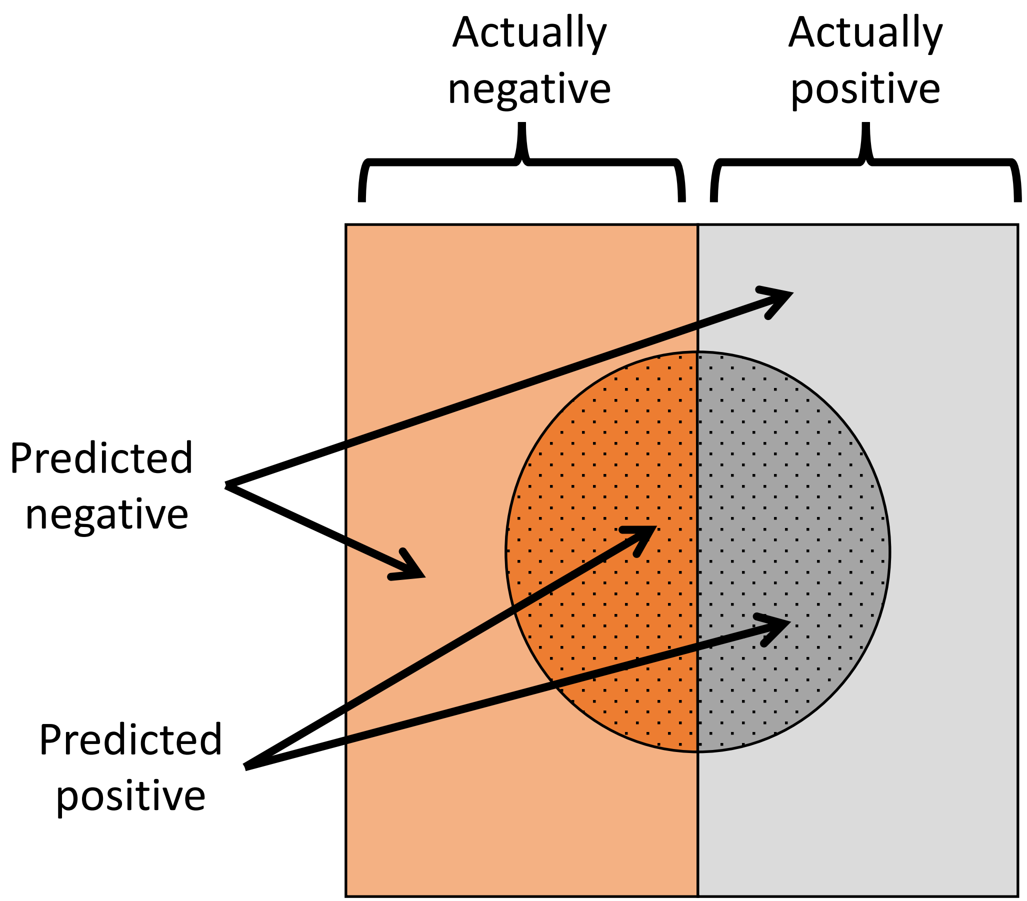
Measuring Performance

- We usually calculate performance on test data
- Calculating performance on training data is called resubstitution and is an *optimistic* measure of performance
  - why?
  - because it can’t detect overfitting

Measuring Performance (Classification)

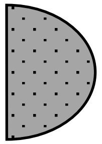
- Accuracy
  - (# test instances correctly labeled)/(# test instances)
- Error
  - 1- accuracy
  - (# test instances incorrectly labeled)/(# test instances)

Measuring Performance (Classification)

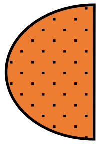


Measuring Performance (Classification)

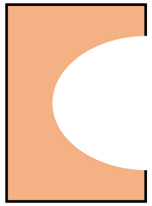
- True positives



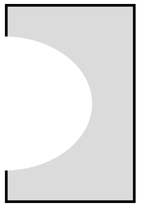
- False positives



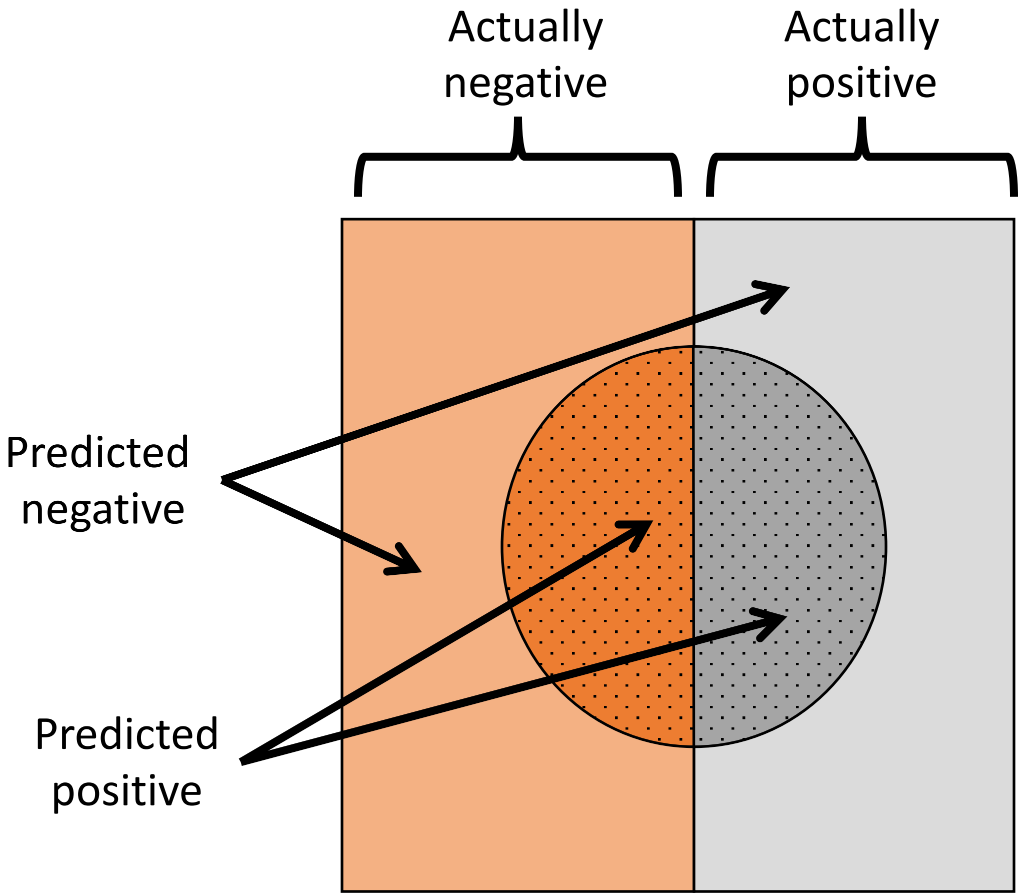
- True neg



- False positive

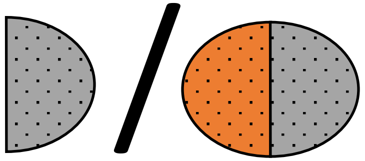


How to remember: the first word is whether the prediction is correct, the second word is what the prediction was.

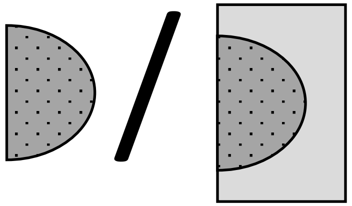


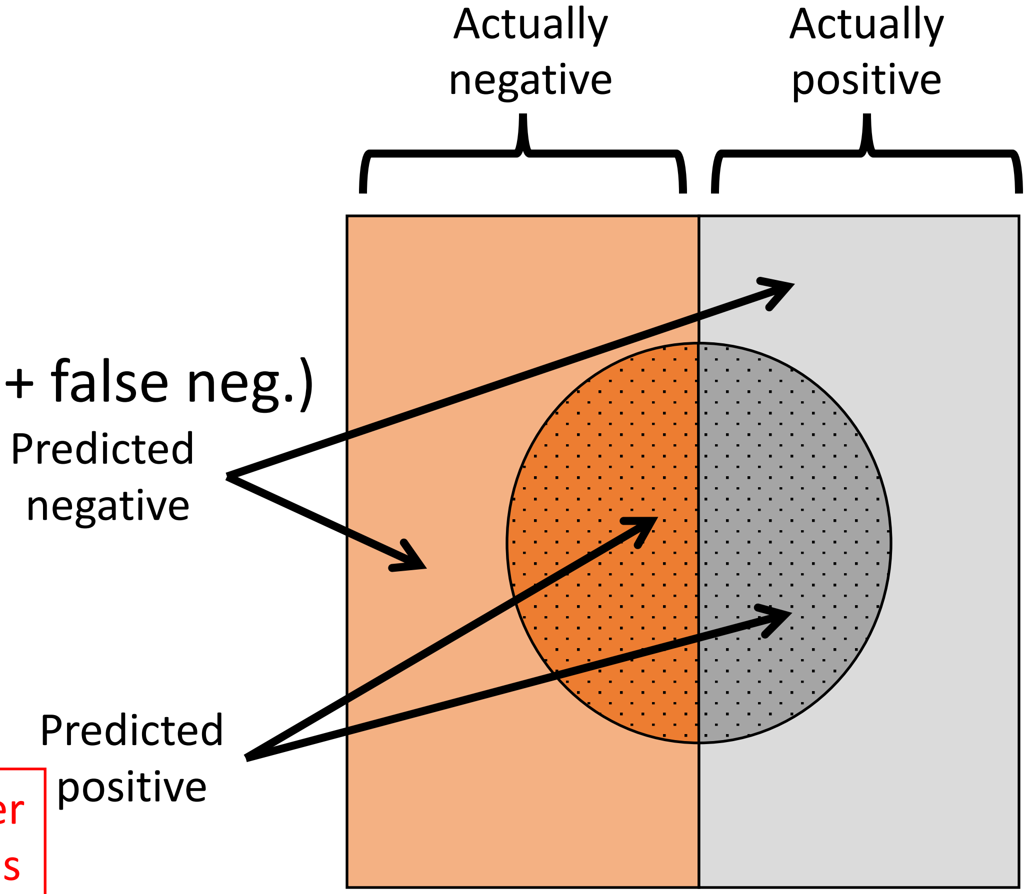
Measuring Performance (Classification)

- Precision
  - true positives/(true pos. + false pos.)



- Recall
  - true positives/(true pos. + false neg.)





How to remember: the first word is whether the prediction is correct, the second word is what the prediction was.

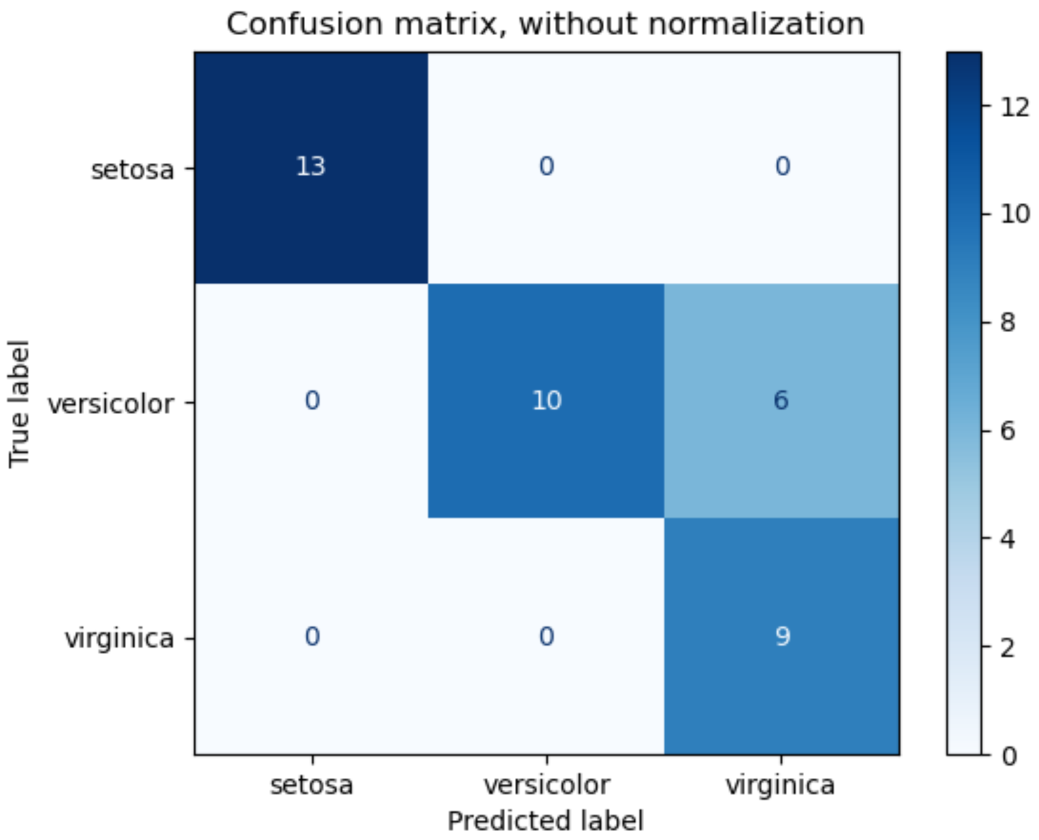
Measuring Performance (Classification)

- F1
  - harmonic mean of precision and recall
  - 2*(p*r)/(p+r)

Evaluating when >1 class

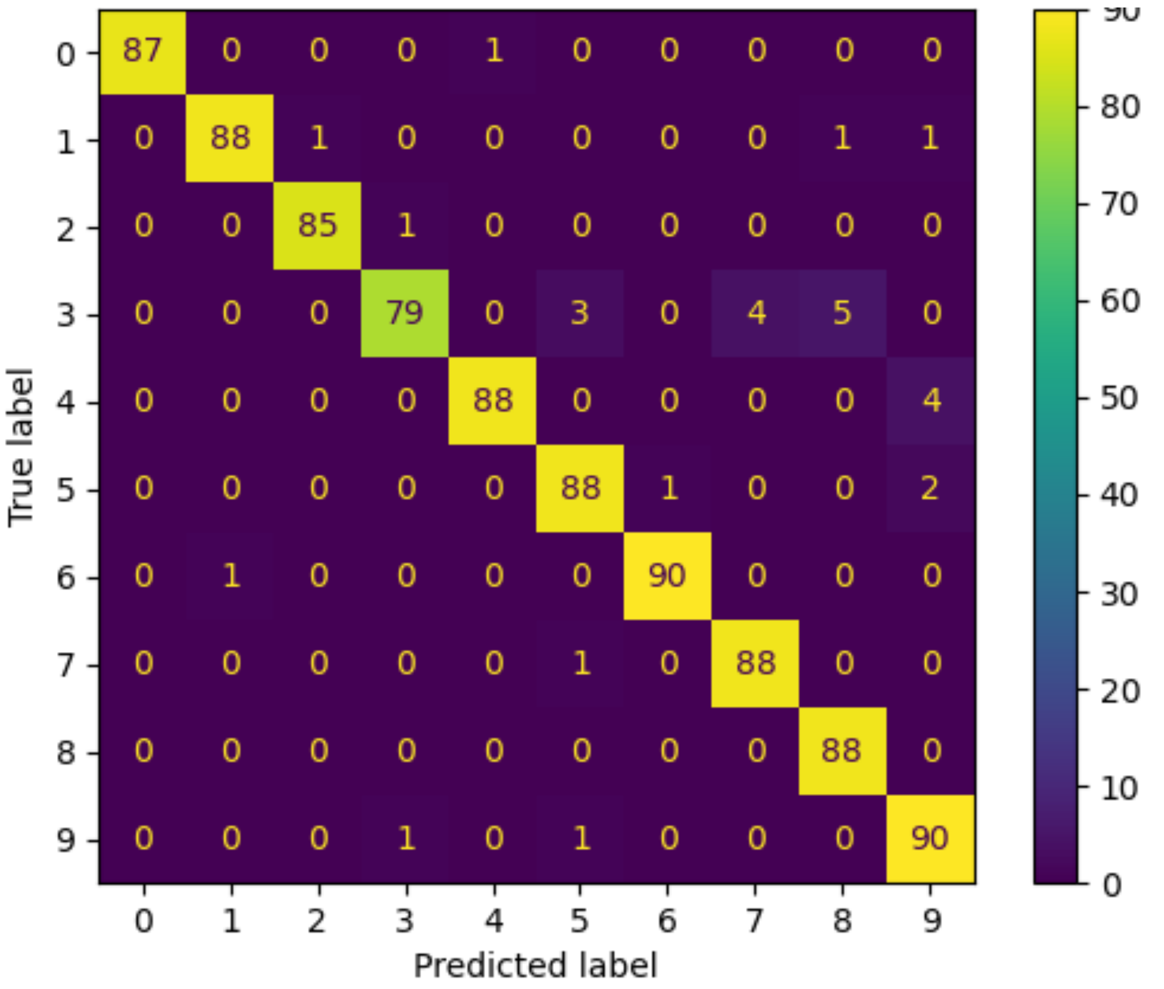
- Can still compute accuracy/error
- Can also compute per-class P, R, F1

Confusion matrix



https://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix

Confusion matrix (handwritten digits)



https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#sphx-glr-auto-examples-classification-plot-digits-classification-py

Measuring Performance (Regression)

- Regression
  - predicting a real number
- Root Mean Squared Error (RMSE)
  - sometimes just MSE (no sqrt)

$$\sqrt{\frac{1}{N}\underbrace{\sum_{i=1}^{N}}_{\text{sum over all N test instances}}\underbrace{(\hat{y}_i - y_i)^2}_{\text{error}}}$$

Performance

- Comparing performance of classifiers
- How do you know if your accuracy number is “high” or error is “low”?

Exercise

Consider the task of building a classifier from random data, where the attribute values are generated randomly irrespective of the class labels. Assume the data set contains records from two classes, “+” and “−.” Half of the data set is used for training while the remaining half is used for testing.

(a) Suppose there are an **equal number** of positive and negative records in the data and the **classifier predicts every test record to be positive**. What is the expected error of the classifier on the test data?

Answer: 50%.

(b) Repeat the previous analysis assuming that the classifier predicts each test record to be **positive class** with probability 0.8 and **negative class** with probability 0.2.

Answer: 50%.

Exercise

Consider the task of building a classifier from random data, where the attribute values are generated randomly irrespective of the class labels. Assume the data set contains records from two classes, “+” and “−.” Half of the data set is used for training while the remaining half is used for testing.

(c) Suppose **2/3** of the data belong to the **positive** class and the remaining **1/3** belong to the **negative** class. What is the **expected** **error** of a classifier that **predicts every test record to be positive**?

Answer: (2/3)*0+(1/3)*1 = 33%.

(d) Repeat the previous analysis assuming that the classifier predicts each test record to be positive class with probability 2/3 and negative class with probability 1/3.

Answer: (2/3)*(1/3)+(1/3)*(2/3) = 44.4%.

Exercise

Consider a classifier X that has **Accuracy = 50%** on a (test) dataset with a class taking 2 possible values (A, B).

The distribution of the instances for each class value is:

A:50,   B:50.

How does X compare to a random classifier Y that outputs A, and B, 50%, 50% of the time, respectively.

Answer:

Y's accuracy:

(50*50/100 + 50*50/100)/100 = 50%

- So, X performs the same (accuracy-wise) as Y.

Exercise

Consider a classifier X that has **Accuracy = 50%** on a (test) dataset with a class taking 4 possible values (A, B, C, and D).

The distribution of the instances for each class value is

A:25, B:25, C:25, and D:25.

How does X compare to a random classifier Y that outputs A, B, C, and D 25%, 25%, 25%, and 25% of the time, respectively.

Answer:

Y's accuracy:

(25*25/100 + 25*25/100 + 25*25/100 + 25*25/100)/100 = 25%

- So, X does twice better than Y (accuracy-wise).

Exercise

The distribution of the instances for each class value is A:25, B:25, C:25, and D:25.

Random classifier Y outputs A, B, C, and D, 25%, 25%, 25%, and 25% of the time, respectively.

Precision and Recall (wrt A)?

Answer:

Y will say 25% of the time “A” and 75% of the time “not A”.

TP = 1/4*1/4, FP = 3/4*1/4, FN = 1/4*3/4

Precision=

TP/(TP+FP) = 25%

Recall=

TP/(TP+FN) = 25%

Exercise

The distribution of the instances for each class value is A:10, B:40, C:25, and D:25.

Random classifier Y outputs A, B, C, and D, 50%, 30%, 10%, and 10% of the time, respectively.

Precision and Recall (wrt A)?

Answer:

Y will say 50% of the time “A” and 50% of the time “not A”.

TP = ? FP = ? FN = ?

Precision=

TP/(TP+FP) = 1/10= 10%

Recall=

TP/(TP+FN) = 1/2= 50%

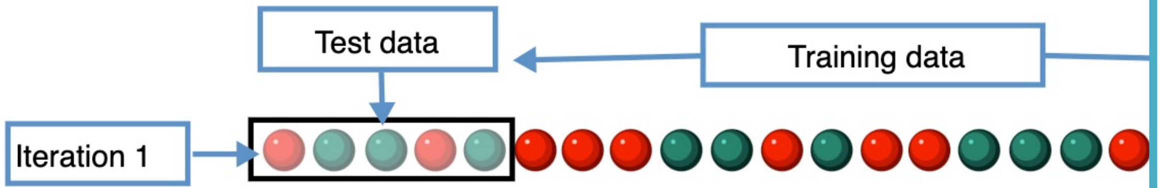
How to handle tuning hyperparameters

- Two regimes:
  1. Validation Set (no cross validation, splits fixed)
     - Split into Train/Validation/Test (e.g. 70,10,20%)
     - Train on Training data, use validation data to set hyperparams or choose model type
     - Report final accuracy by training on all of training data (with your final chosen parameters) and predicting on test data.

- Key idea: data used to tune hyperparams should **never** be used to report accuracy

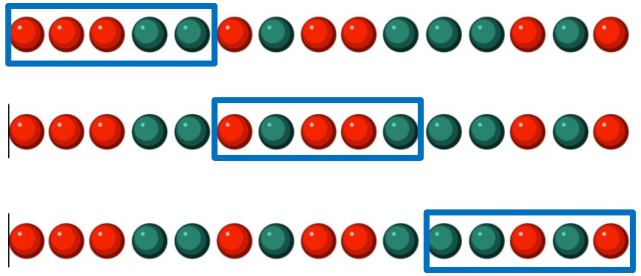
How to handle tuning hyperparameters

- Two regimes:
  2. Nested Cross Validation
     - Partition into k_1 sets and repeat k_1 times:
       I. For each set of $(k_1-1)/k_1$ Train, $1/k_1$ Test (e.g. 90,10%)
          - For each hyperparameter setting h
            - Partition into k_2 sets and repeat k_2 times:
            i. Take your Train set from step I (e.g. 90% of all data) and further split it into $(k_2-1)/k_2$ *sub-Train*, $1/k_2$ *sub-Test* (e.g. 0.9*0.9=81% of all data, 0.1*0.9=9% of all data)
               i. Train on *sub-Train* from step i using hyperparams h
               ii. Test on *sub-Test* from step i
            - Calculate average performance across all k_2 splits for hyperparam h
          - Return hyperparam h' that maximizes performance
       II. Train on all Train data from step I using hyperparam h', test on Test data from step I. Record performance
     - Report average performance across all k_1 folds of Train and Test



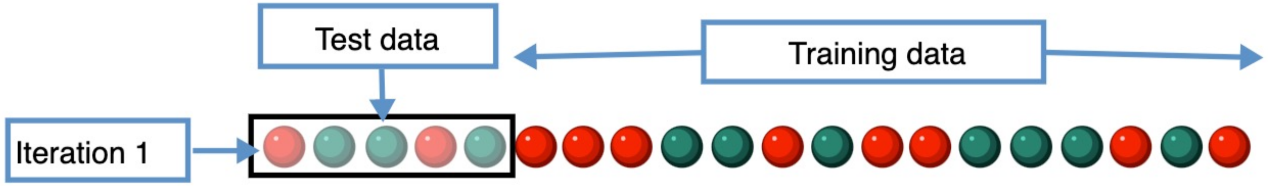
Important: the data I use to choose a hyperparam is **not** used to calculate the **test** accuracy with that hyperparam in the **outer** loop!

Run x-val on this train data, with each hyperparam



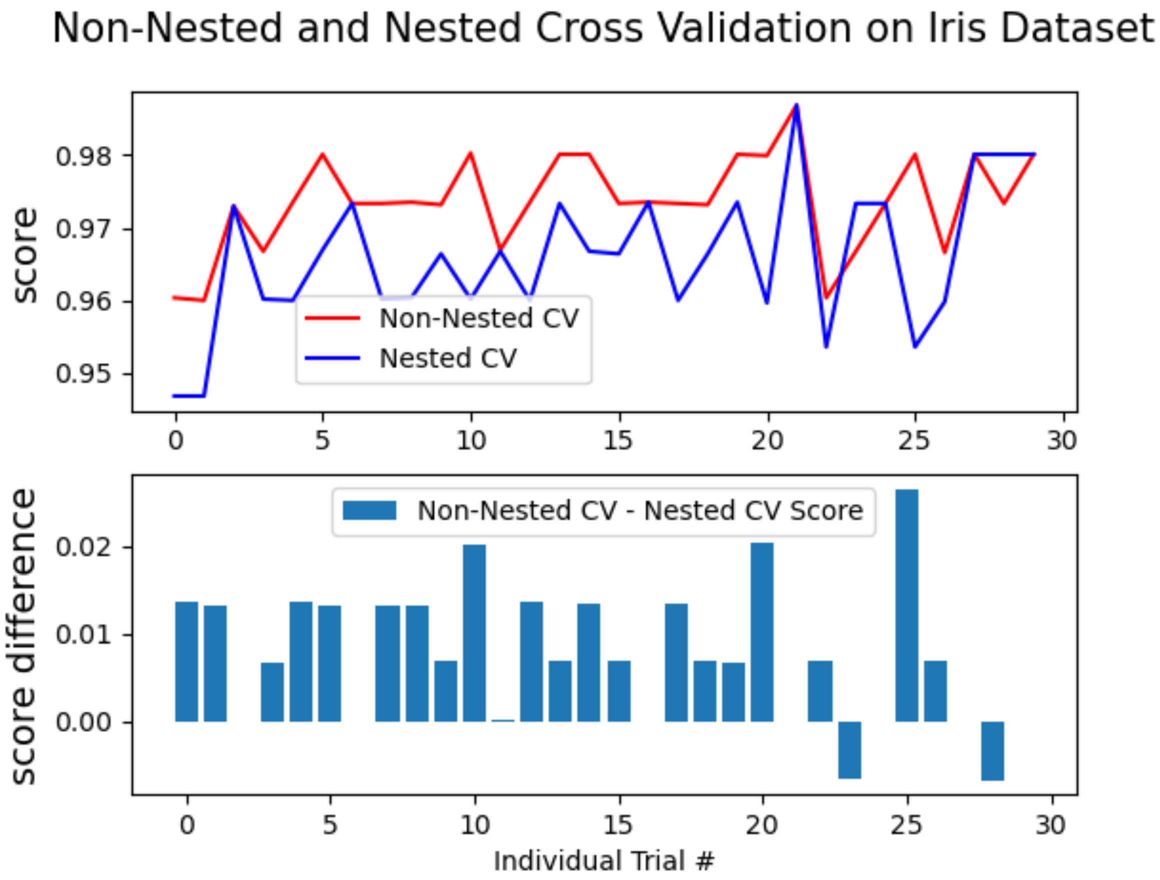
h1: 0.5, h2: 0.7, h3: 0.6, h4: 0.9 -> Best accuracy is with h4

Train on all of the training data using h4, test on test data, and save accuracy for this iteration



Repeat for all outer folds, then report average accuracy

Nested Cross Validation



https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

See also

- https://mlfromscratch.com/nested-cross-validation-python-code/#/

Other Evaluation Methods

- Random subsampling / Monte Carlo cross validation
  - choose a test set randomly and repeatedly, without replacement
  - like cross-validation except test sets need not be disjoint
- Bootstrap
  - choose a test set randomly with replacement
  - like random sampling, but with replacement
  - Pessimistic estimate, corrected with .632 bootstrap estimate

More info: http://web.cs.iastate.edu/~jtian/cs573/Papers/Kohavi-IJCAI-95.pdf

Thanks! Questions?

- 566 students come up if you’re looking for a project group or more members