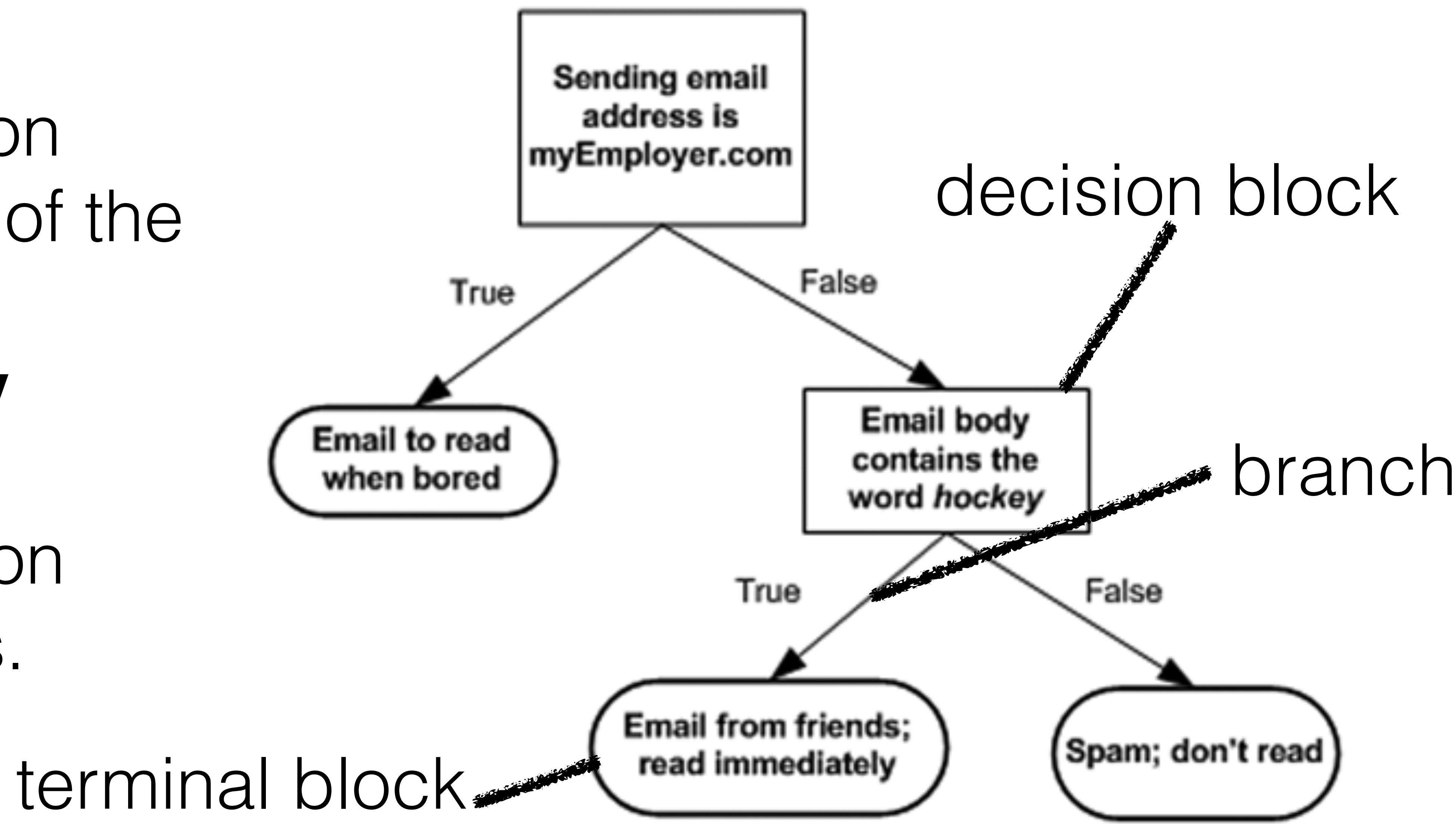
Splitting Datasets One Feature at a Time: Decision Trees

Harvey Alférez, Ph.D.

Introduction

The decision tree is one of the **most commonly used** classification techniques.



Introduction (Cont.)

- The **kNN algorithm** does a **great job of classifying**, but **it didn't lead to any major insights about the data**.

- **One of the best things about decision trees is that humans can easily understand the data!**

Introduction (Cont.)

- The **decision tree** does a **great job** of **distilling data** into **knowledge**.
  - With this, you can take a set of unfamiliar data and extract a set of **rules**.
    - The machine learning will take place as the machine creates these rules from the dataset.
  - The results obtained by using decision trees are often comparable to those from a **human expert** with decades of experience in a given field!

Tree Construction

CreateBranch() - Recursive

Check if every item in the dataset is in the same class:
    If so return the class label
    Else
        find the best feature to split the data
        split the dataset
        create a branch node
            for each split
                call createBranch and add the result to the branch node
        return branch node

Tree Construction (Cont.)

General approach to decision trees

1. Collect: Any method.
2. Prepare: This tree-building algorithm works only on nominal values, so any continuous values will need to be quantized.
3. Analyze: Any method. You should visually inspect the tree after it is built.
4. Train: Construct a tree data structure.
5. Test: Calculate the error rate with the learned tree.
6. Use: This can be used in any supervised learning task. Often, trees are used to better understand the data.

Tree Construction (Cont.)

We're also going to split on one and only one feature at a time. If our training set has 20 features, how do we choose which one to use first?

	Can survive without coming to surface?	Has flippers?	Fish?
1	Yes	Yes	Yes
2	Yes	Yes	Yes
3	Yes	No	No
4	No	Yes	No
5	No	Yes	No

2 Features   2 Classes

Tree Construction (Cont.)

- For our classifier algorithm to work, you need to:
  - Measure the **entropy** (expected value of information)
  - **Split** the **dataset**
  - **Measure** the **entropy** on the **split sets**
  - See **if splitting was the right thing to do**
- You'll do this for **all of the features** to determine the best feature to split on.

Tree Construction (Cont.)

- **1. Measure the Entropy**
  - We choose to split our dataset in a way that makes our unorganized data **more organized**.
    - One way to organize this messiness is to measure the information.
      - Using **information theory**, you can measure the information before and after the split.

Tree Construction (Cont.)

- **1. Measure the Entropy (Cont.)**
  - The change in information before and after the split is known as the **information gain**.
    - When you know how to calculate the **information gain**, you can split your data across every feature **to see which split gives you the highest information gain**.
      - **The split with the highest information gain is your best option.**

Tree Construction (Cont.)

- **1. Measure the Entropy (Cont.)**
  - In trees.py:

```
def createDataSet():
        dataSet = [[1, 1, 'yes'],
                   [1, 1, 'yes'],
                   [1, 0, 'no'],
                   [0, 1, 'no'],
                   [0, 1, 'no']]
labels = ['no surfacing','flippers']
return dataSet, labels
```

Tree Construction (Cont.)

- **1. Measure the Entropy (Cont.)**

```
>>> import trees
>>> myDat,labels=trees.createDataSet()
>>> myDat
[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0,
1, 'no']]
>>> trees.calcShannonEnt(myDat)
0.9709505944546686
```

Tree Construction (Cont.)

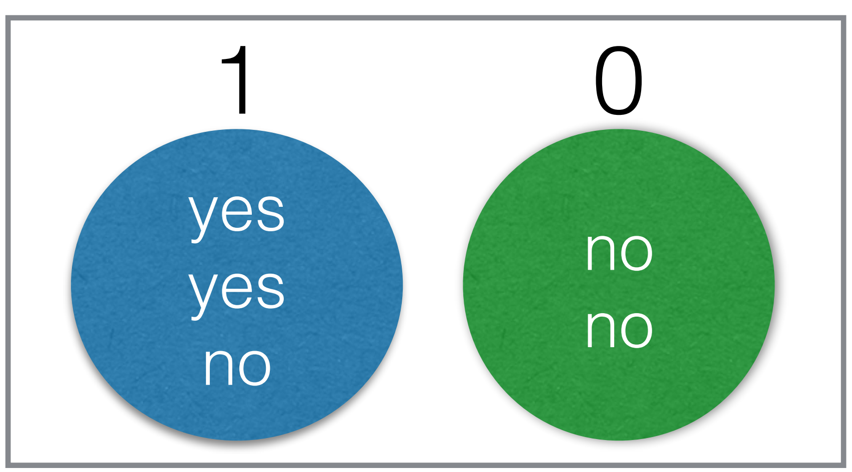
- **1. Measure the Entropy (Cont.)**
  - *The higher the entropy, the more mixed up the data is:*

```
>>> myDat[0][-1]
>>> myDat[0][-1]='maybe'
>>> myDat
[[1, 1, 'maybe'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]
>>> trees.calcShannonEnt(myDat)
1.3709505944546687
```

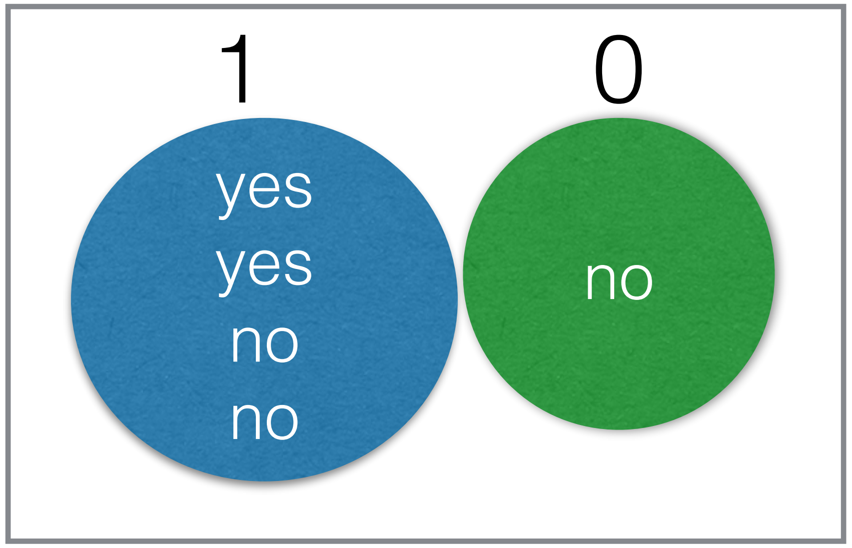
Tree Construction (Cont.)

- **2. Split the Dataset** (Shannon Entropy is calculated in the whole dataset before and after splitting)

```
>>> reload(trees)
<module 'trees' from 'trees.py'>
>>> myDat,labels=trees.createDataSet()
>>> trees.chooseBestFeatureToSplit(myDat)
0
>>> myDat
[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]
```



Split on Feature 0

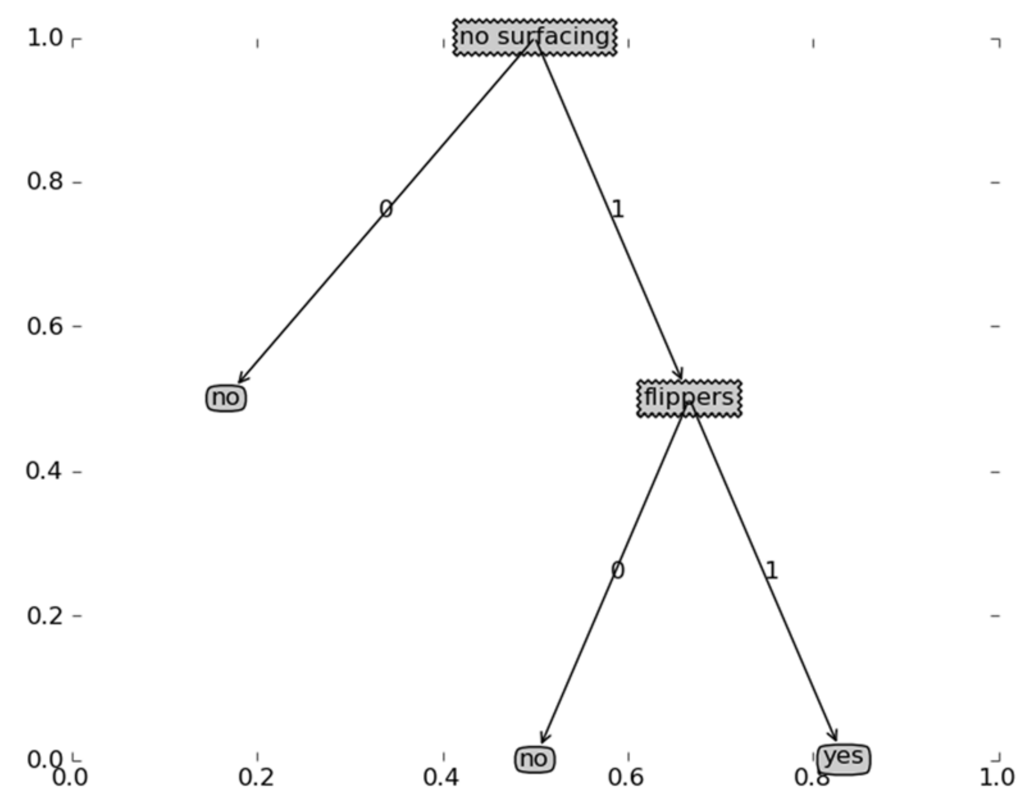


Split on Feature 1

Tree Construction (Cont.)

- **3. Recursively Building the Tree**

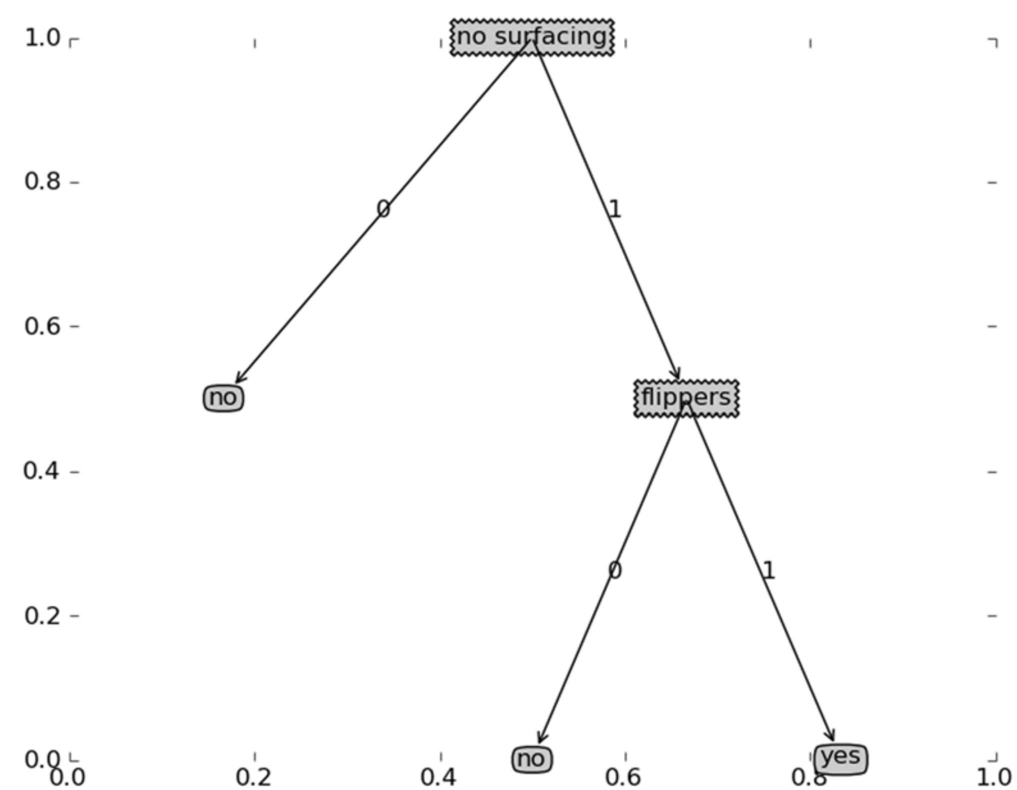
```
>>> reload(trees)
<module 'trees' from 'trees.pyc'>
>>> myDat,labels=trees.createDataSet()
>>> myTree = trees.createTree(myDat,labels)
>>> myTree
{'no surfacing': {0: 'no', 1: {'flippers':
{0: 'no', 1: 'yes'}}}}
```



Test: Using the Tree for Classification

```
>>> myDat,labels=trees.createDataSet()
>>> labels
['no surfacing', 'flippers']
>>> myTree=treePlotter.retrieveTree (0)
>>> myTree
{'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: ‘yes'}}}}
>>> trees.classify(myTree,labels,[1,0])
        'no'
>>> trees.classify(myTree,labels,[1,1])
        'yes'
```

```
import treePlotter
treePlotter.createPlot(myTree)
```



Using Decision Trees to Predict Contact Lens Type

- The Lenses dataset3 is one of the most famous datasets.
- It’s a number of observations based on patients’ eye conditions and the type of contact lenses the doctor prescribed.
- The classes are **hard**, **soft**, and **no contact lenses**.

Using Decision Trees to Predict Contact Lens Type (Cont.)

- You can load the data by typing the following into your Python shell:

```
>>> fr=open('lenses.txt’)
>>> lenses=[inst.strip().split('\t') for inst in fr.readlines()]
>>> lensesLabels=['age', 'prescript', 'astigmatic', 'tearRate']
>>> lensesTree = trees.createTree(lenses,lensesLabels)
>>> lensesTree
{'tearRate': {'reduced': 'no lenses', 'normal': {'astigmatic': {'yes':
        {'prescript': {'hyper': {'age': {'pre': 'no lenses', 'presbyopic':
        'no lenses', 'young':'hard'}}, 'myope': 'hard'}}, 'no': {'age': {'pre':
        'soft', 'presbyopic': {'prescript': {'hyper': 'soft', 'myope':
        'no lenses'}}, 'young': 'soft'}}}}}}
>>> treePlotter.createPlot(lensesTree)
```

Using Decision Trees to Predict Contact Lens Type (Cont.)