Feature Selection

Feature Selection

- Give set of features, some features are more important than others
- We want to select some subset of features to be used by learning algorithms
- Exhaustive approach:
  - Train systems on all $D!$ possible combinations and choose the best one.
  - Pro: It's a globally optimal solution
  - Con: It's often not-feasible

D!

Greedy Feature Selection

D features $\{F_1, F_2, \ldots$

D-1 features

- Greedy heuristic:
  - Start from empty set of features $F = \emptyset$
  - For iteration t, for each remaining feature j
    - Run learning algorithm for features $\mathrm{F} \cup j$
  - Select next best feature $\hat{\jmath}$
    - $F = F \cup \{\hat{\jmath}\}$
  - Continue until converge on locally optimal result
- You could also do backwards feature selection
  - Start with all features, remove worst, etc..

all features

$\{\_ \quad \_ \quad \_\}$

- This would involve training/evaluating $D + (D-1) + (D-2) + 1 = \sum_{i=1}^{D} i = \frac{D(D+1)}{2}$ systems

D system

- Better….

Separability Feature Selection

- What if we were *rank* features somehow out-of-the-gate and then greadily add them one by one
- This would just be D systems
- So how do we rank them?
- If we don't have class labels we could use something like least covariance
    - We want to use features that are unique.
- If we do have class labels maybe we can find a way to measure a feature's effect on class separability…

Feature Selection by L1

Feature Selection by L1

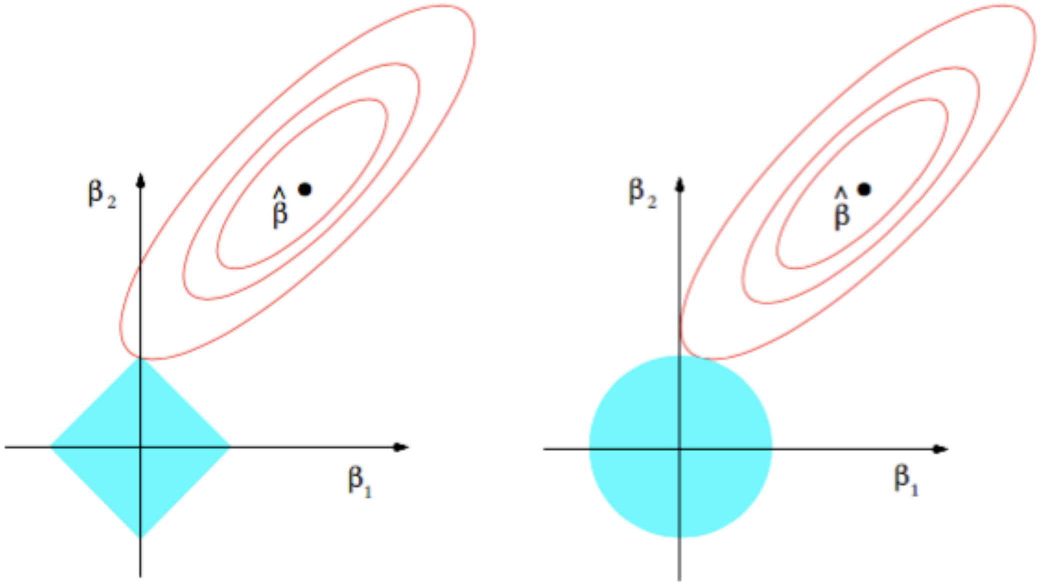
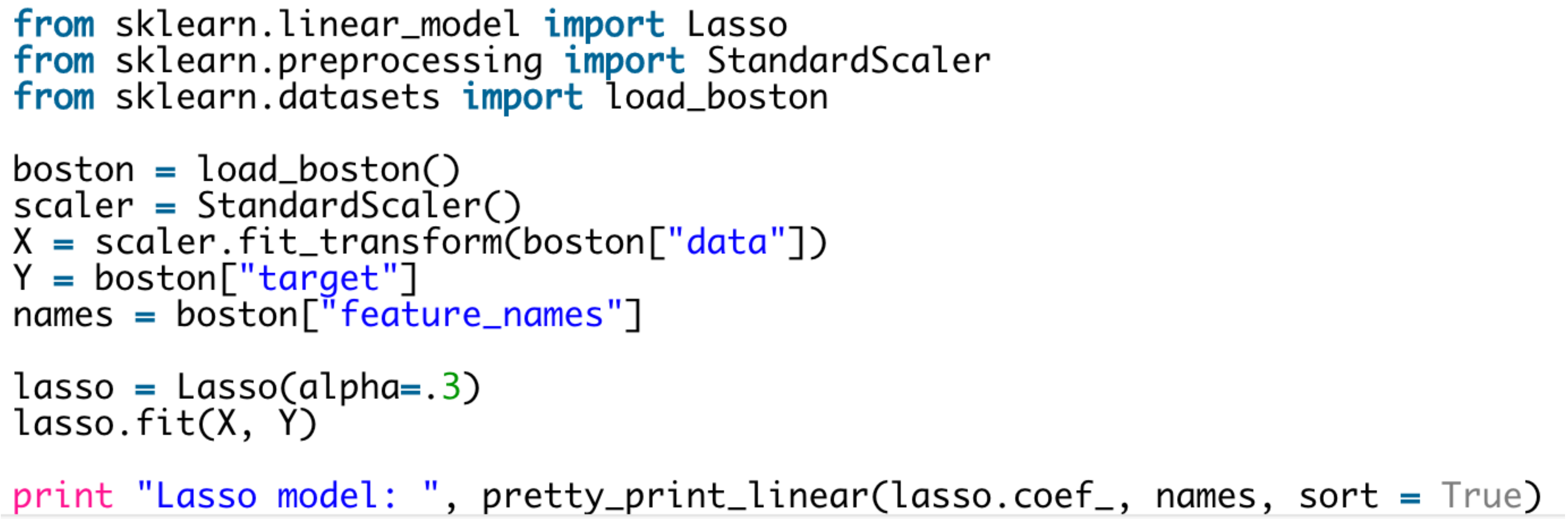


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

Boston Housing Dataset

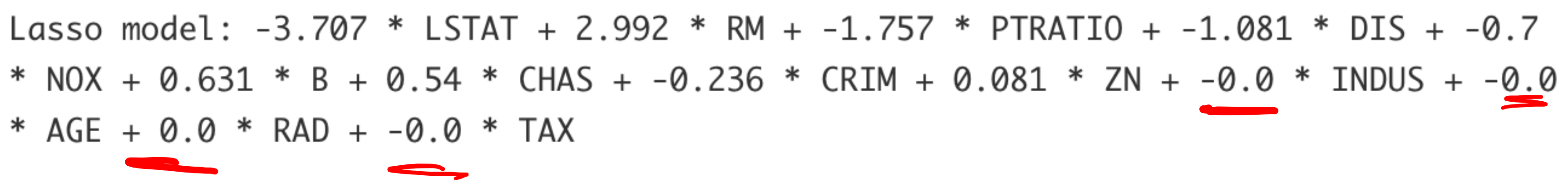


```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_boston

boston = load_boston()
scaler = StandardScaler()
X = scaler.fit_transform(boston["data"])
Y = boston["target"]
names = boston["feature_names"]

lasso = Lasso(alpha=.3)
lasso.fit(X, Y)

print "Lasso model: ", pretty_print_linear(lasso.coef_, names, sort = True)
```



```
Lasso model: -3.707 * LSTAT + 2.992 * RM + -1.757 * PTRATIO + -1.081 * DIS + -0.7
* NOX + 0.631 * B + 0.54 * CHAS + -0.236 * CRIM + 0.081 * ZN + -0.0 * INDUS + -0.0
* AGE + 0.0 * RAD + -0.0 * TAX
```

Feature Selection by Information Gain

$y \ln g(\theta, x)$

$y \log \hat{y}$

$P_{v_i} \log P_{v_i}$

loss

logistic regression

cross-entropy

true false

- Given probability of events $v_1, \ldots, v_n$ as $P(v_1), \ldots, P(v_n)$ we can compute the ***entropy*** as

$$H\big(P(v_1), \ldots, P(v_n)\big) = \sum_{i=1}^{n} \left(-P(v_i) \log_n P(v_i)\right)$$

$-P(\text{true}) \log_2 P(\text{true}) + -P(\text{false}) \log_2 P(\text{false})$

- Entropy measure the randomness of the data
- Example: Tossing a fair coin
  - $v_1 = heads, v_2 = tails,$
  - $P(v_1) = 0.5, P(v_2) = 0.5$
  - $H\left(\frac{1}{2}, \frac{1}{2}\right) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$

$= 1 \in$ entropy random

$0 \rightarrow$ deterministic

Feature Selection via Information Gain

1

0 1

$\boxed{1-0} = 1$

$1-1=0$

- Let us assume we have a dataset with binary labels:

$$Y_i \in \{0,1\}$$

- And that all features are discrete.
- Let a chosen feature $j \in \{1, \ldots, D\}$ have $\overset{2}{k}$ possible values (thus a discretized feature).
- Then the dataset $X = \{X_i\}_{i=1}^{N}$ can be split into subsets $\{E_1, \ldots, E_k\}$ according to each observation's value, $X_{i,j}$
- The intuition is to compute the original entropy of the system and then the average entropy on this split
  - We want to maximize this difference
  - This is our "gain"
- Or, we can say that we want to minimize the average entropy after the split.

Feature Selection via Information Gain

- Let $p = \#\{1\}$,be the number of samples with label one over the entire dataset
- Let $n = \#\{0\}$ be the number of samples with label zero over the entire dataset
- Finally let p_i, n_i be the number samples in subset E_i, with label one and zero, respectively.
- Let's define the average entropy with respect to *A* as:

$$\mathbb{E}(H(A)) = \sum_{i=1}^{k} \frac{p_i + n_i}{p + n} H\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

Feature Selection via Information Gain

- We can now compute information Gain (IG), or reduction in entropy, that would occur if we split on this attribute/feature:

A feature

$$IG(A) = H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \mathbb{E}\left(H(A)\right)$$

- We should choose the attribute with the largest IG!
    - I.e. the one with the smallest average entropy.

Example

- For reference:
  - $H\big(P(v_1), \dots, P(v_n)\big) = \sum_{i=1}^{n}(-P(v_i)\log_2 P(v_i))$
  - $\mathbb{E}\big(H(A)\big) = \sum_{i=1}^{k}\frac{p_i+n_i}{p+n} H\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$
- Class 1 Samples
  - $\{(1,1)(1,3), (2,2)\}$
- Class 0 Samples
  - $\{(1,2), (3,2), (2,2)\}$
- Let’s figure out which feature provides the highest information gain!

Example

Class 1 Samples

{(1,1)(1,3), (2,2)}

Class 0 Samples

{(1,2), (3,2), (2,2)}

$$X \Rightarrow \begin{array}{cc} f_1 & f_2 \\ \hline 1 & 1 \\ 1 & 3 \\ 2 & 2 \\ 1 & 2 \\ 3 & 2 \\ 2 & 2 \end{array} \qquad Y \Rightarrow \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$E_{f_1} \begin{bmatrix} 1 & 1 & : & 1 \\ 1 & 3 & : & 1 \\ 1 & 2 & : & 0 \end{bmatrix} \rightarrow \left(-\frac{1}{3}\log\frac{1}{3} + -\frac{2}{3}\log\frac{2}{3}\right)$$

$$E_{f_2} \begin{bmatrix} 2 & 2 & \cdot & 1 \\ 2 & 2 & \cdot & 0 \end{bmatrix} \rightarrow 1$$

$$E_{f_3} \begin{bmatrix} 3 & 2 & \cdot & 0 \end{bmatrix} \rightarrow 0$$

$$\underbrace{\frac{3}{6}\left(-\frac{1}{3}\log\frac{1}{3} - \frac{2}{3}\log\frac{2}{3}\right)}_{E_{f_1}} + \underbrace{\frac{2}{6}(1)}_{E_{f_2}} + \underbrace{\frac{1}{6}(0)}_{E_{f_3}} \Rightarrow \text{total entropy of } f_1$$

Example

- $H\big(P(v_1),\ldots,P(v_n)\big) = \sum_{i=1}^{n}(-P(v_i)\log_2 P(v_i))$
- $\mathbb{E}\big(H(A)\big) = \sum_{i=1}^{k}\frac{p_i+n_i}{p+n} H\left(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i}\right)$
- Positive Samples
  - {(1,1)(1,3),(2,2)}
- Negative Samples
  - {(1,2),(3,2),(2,2)}
- Feature 1
  - p_1=2, n_1=1
  - p_2=1, n_2=1
  - p_3=0, n_3=1 $\frac{3}{6} E f_1 + \frac{2}{6} E f_2 + \frac{1}{6} E f_2$
  - $\mathbb{E}\big(H(1)\big)$=
    (2+1)/(3+3)*(-2/3*$\log_2$(2/3)+-1/3*$\log_2$(1/3))
    + (1+1)/(3+3)*(-1/2*log(1/2)+-1/2log(1/2)
    + (0+1)/(3+3)*(0log(0)-1log(1)) = 0.7925
  - IG(1)=(-3/6log(3/6)-3/6log(3/6))-0.7925 = 0.2075

$1 - 0.7925 = \boxed{0.2075}$

Example

- Positive Samples
  - {(1,1)(1,3),(2,2)}
- Negative Samples
  - {(1,2),(3,2),(2,2)}
- Feature 2
  - $p_1=1$, $n_1=0$
  - $p_2=1$, $n_2=3$
  - $p_3=1$, $n_3=0$
  - $\mathbb{E}(H(2))=$
    $(1+0)/(3+3)*(-1/1*\log_2(1/1)+-0*\log_2(0))$
    $+ (1+3)/(3+3)*(-1/4*\log(1/4)+-3/4\log(3/4)$
    $+ (1+0)/(3+3)*(-1/1\log(1/2)-0/1\log(0/1)) = 0.5409$
  - $IG(2)=(-3/6\log(3/6)-3/6\log(3/6))-0.5409 = 0.4591$
- Recall IG(1)=0.2075
- So we should prioritized feature 2!

More than 2 classes?

- Think about how you can change the equations for work for more than two class…

$$H\big(P(v_1),\ldots,P(v_n)\big) = \sum_{i=1}^{n}(-P(v_i)\log_n P(v_i))$$

$$\mathbb{E}\big(H(A)\big) = \sum_{i=1}^{k}\frac{p_i+n_i}{p+n}H\left(P(v_1),\ldots,P(v_n)\right)$$

$$IG(A) = H\left(\frac{p}{p+n},\frac{n}{p+n}\right) - \mathbb{E}(H(A))$$

IG for real-valued features

0.1
0.3
0.8
0.2
0.4

μ
σ

$\frac{x-\mu}{\sigma}$

zero

if $x_i > 0$ $y_i=1$

$x_i < 0$ $y_i=0$

- The examples we showed work for categorical and/or enumerated features
- This could be expanded to work on real-valued features
- Two approaches:
  1. Break up the range of possible values into enumerated/discrete regions
  2. Assume some distribution (say Gaussian) μ σ
     1. Separate the data by class
     2. Find the parameters for ~~Gaussians~~ for each feature within in subset.
     3. For each sample compute the probability of class k given j^{th} attribute value $X_{i,j}$, $P(j = X_{i,j}|Y_{\ i} = C_k)$ and use these to compute the entropy.

        Initial entropy: $H\left(\frac{p}{p+n}, \frac{n}{p+n}\right)$

        Average Entropy after Split:

$$\mathbb{E} = \frac{1}{N}\sum_{i=1}^{N} H\left(P\left(j = X_{i,j}|Y_i = 0\right), P\left(x_j = X_{i,j}|Y_i = 1\right)\right)$$

- Don't worry, we'll explore the use of standard distributions more soon!

Choosing an Attribute

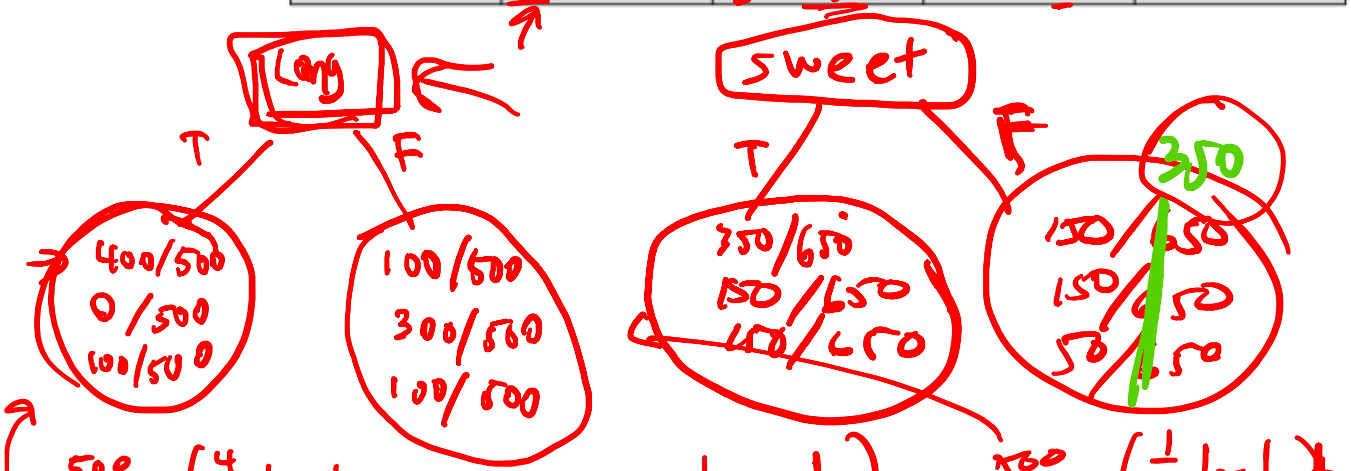
- While there's many ideas on how to choose the next attribute, let's use something we already talked about!
- Information gain/entropy!
- Recall the formula for entropy:

$$H\big(P(v_1), \dots, P(v_n)\big) = \sum_{i=1}^{n} -P(v_i) \log_n P(v_i)$$

Choosing an Attribute

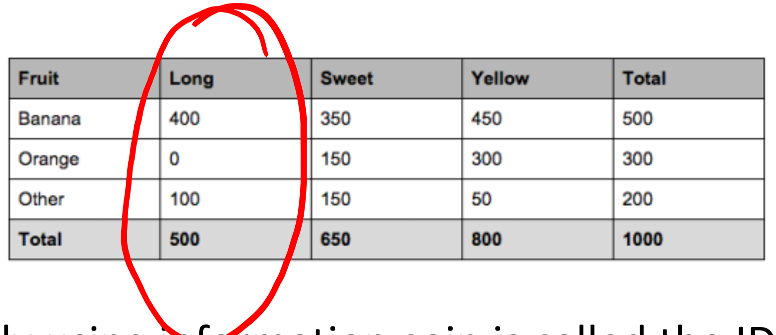
- Idea: A good attribute splits the examples into subsets that contain (ideally) observations from just one class.

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	**500** 500	**650** 350	**800** 200	**1000**



$$\frac{500}{1000}\left(\frac{4}{5}\log_2\frac{4}{5} + -\frac{0}{5}\log_2\frac{0}{5} + -\frac{1}{5}\log_2\frac{1}{5}\right) + \frac{500}{1000}\left(-\frac{1}{5}\log_2\frac{1}{5} + -\frac{3}{5}\log_2\frac{3}{5} + -\frac{1}{5}\log_2\frac{1}{5}\right)$$

Example



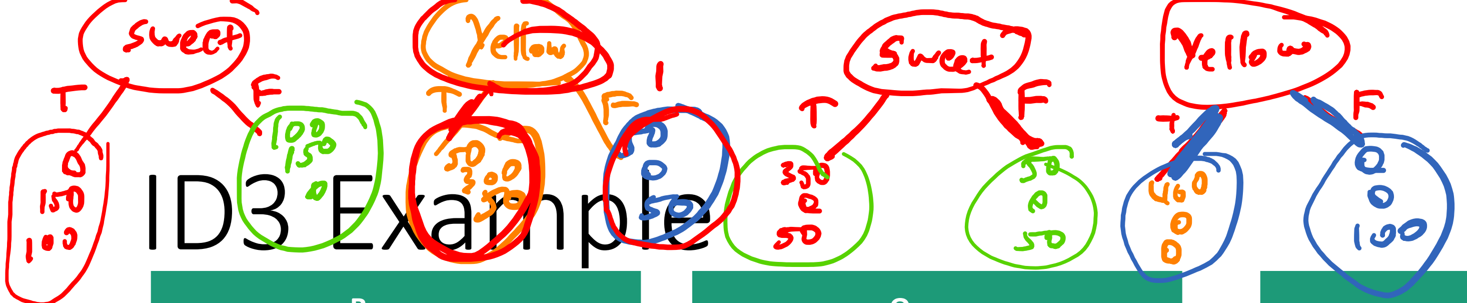
Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	**500**	**650**	**800**	**1000**

- Building a tree completely using information gain is called the ID3 decision tree algorithm
- Let's build this ID3 DT all the way out!
- To do this we'll need a little more information

Banana			
Long	*Sweet*	*Yellow*	*Count*
F	F	F	50
F	F	T	50
F	T	F	0
F	T	T	0
T	F	F	0
T	F	T	50
T	T	F	0
T	T	T	350

Orange			
Long	*Sweet*	*Yellow*	*Count*
F	F	F	0
F	F	T	150
F	T	F	0
F	T	T	150
T	F	F	0
T	F	T	0
T	T	F	0
T	T	T	0

Other			
Long	*Sweet*	*Yellow*	*Count*
F	F	F	0
F	F	T	0
F	T	F	50
F	T	T	50
T	F	F	50
T	F	T	0
T	T	F	50
T	T	T	0



ID3 Example

Drexel UNIVERSITY

E_LF

E_LT

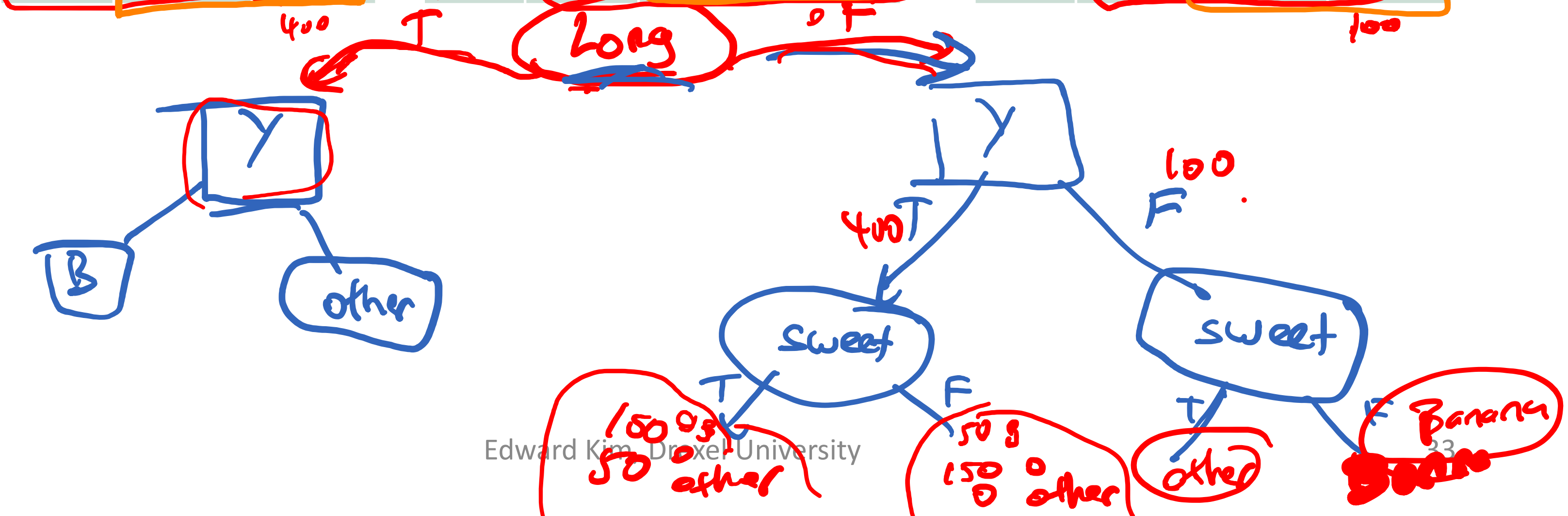
Banana			
~~Long~~	Sweet	Yellow	Count
F	F	F	50
F	F	T	50
F	T	F	0
F	T	T	0
T	F	F	0
T	F	T	50
T	T	F	0
T	T	T	350

400

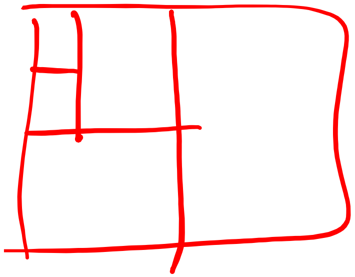
Orange			
Long	Sweet	Yellow	Count
F	F	F	0
F	F	T	150
F	T	F	0
F	T	T	150
T	F	F	0
T	F	T	0
T	T	F	0
T	T	T	0

Other			
Long	Sweet	Yellow	Count
F	F	F	0
F	F	T	0
F	T	F	50
F	T	T	50
T	F	F	50
T	F	T	0
T	T	F	50
T	T	T	0

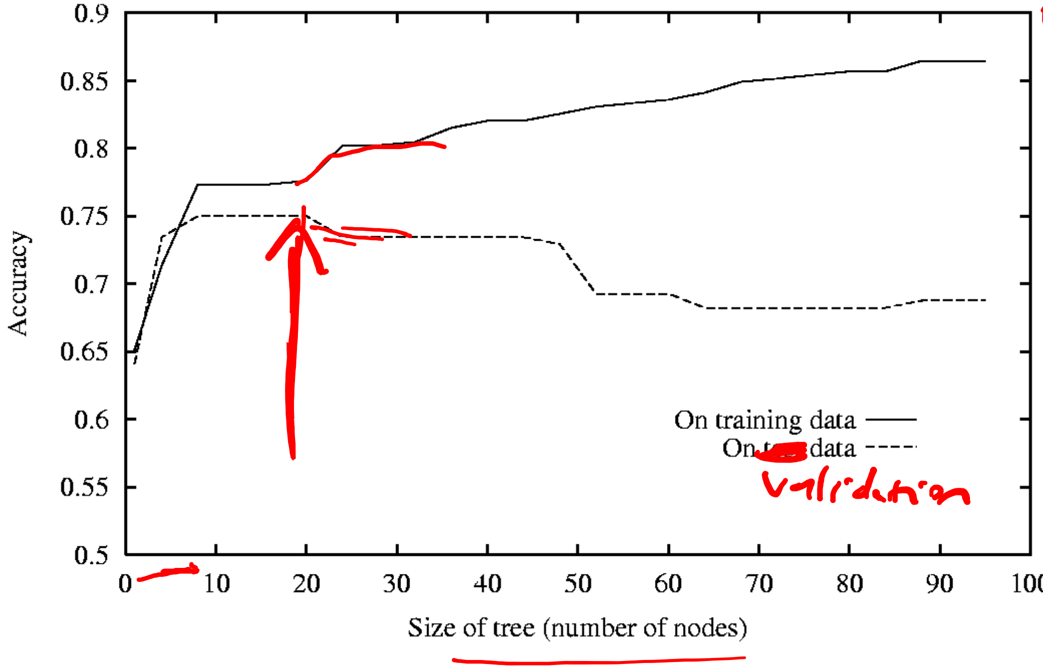
100

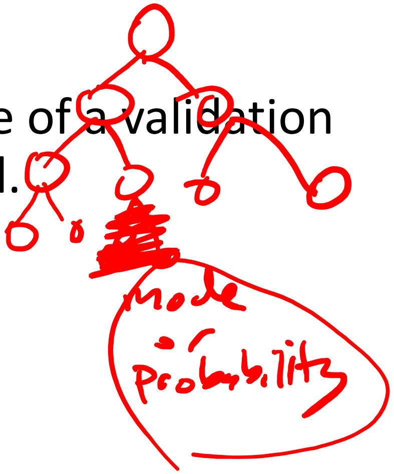


Overfitting



- What's the problem with fitting our data as closely as possible?
  - We may overfit the data!
- Since this is an iterative (or recursive) algorithm, the use of a validation set to decide between different versions is quite natural.





http://jmvidal.cse.sc.edu/talks/decisiontrees/allslides.html

Reduced Error Approaches

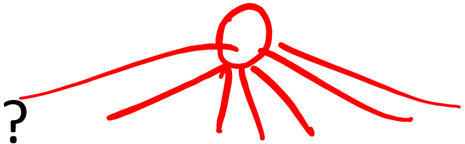
- How can we use the validation set?
- Evaluate it during the growing process.
  - At each *incomplete* branch, just assign the **mode** label of the data going down it.
- Find the level of growth that maximizes the accuracy of the validation set (or conversely minimizes the error).
- Then report test accuracy for that version.
- Reduced Error Growing
  - When you look to split a node based on training data, evaluating after using *validation set*
  - If things get worse, stop
- Reduced Error Pruning
  - Evaluate impact on *validation* set of pruning each possible node (plus those below it)
  - Greedily remove the one that most improve *validation* set accuracy
  - Continue until none help

Other Ideas

- There are other ideas….
- Pruning
  - Try removing each of the splits that happen before leaf nodes and evaluate the validation set on each of them.
  - As long as one of them improves the validation evaluation, remove the one that improves things the most
  - Keep doing this until no removal improves validation evaluation.
- Or maybe we could somehow use statistics to halt the growing process
  - If the information gain provided by a split is above some threshold, split, otherwise don’t

Continuous Valued Inputs

- What if we have features that have continuous values?
    - One branch for each value?
        - Bad idea!!!! (impossible?)



$x_i > 0$ $y_i = 1$

$x_i < 0$ $y_i = 0$

- If we know the range of our values and we set how many branches the node should make then
    - Divide range evenly?
    - Somehow do it more intelligently?