# Ling 131A
# Introduction to NLP with Python

## Classifiers

Marc Verhagen, Fall 2018

# Classification

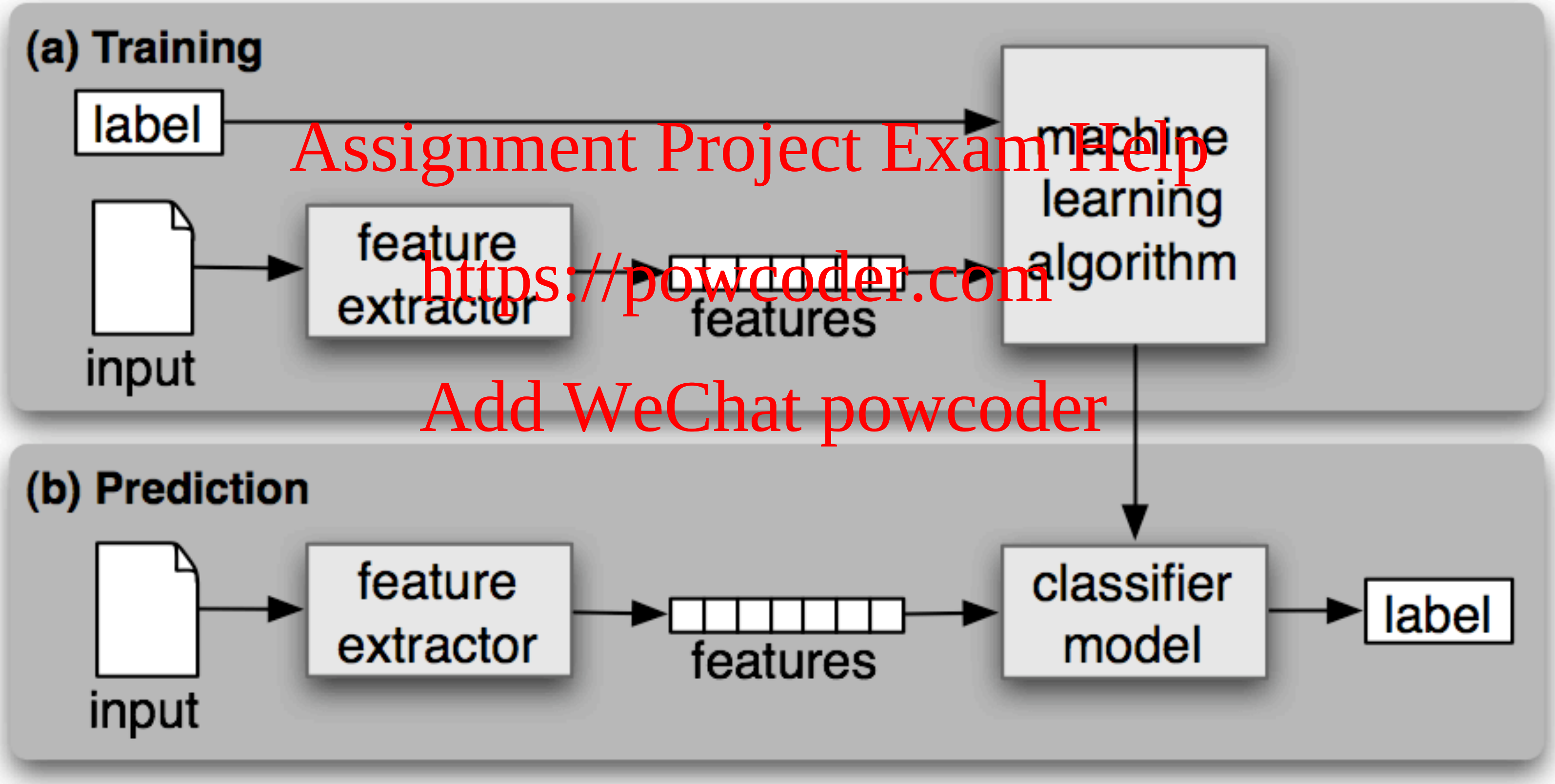
- The task of choosing the correct class label for a given input
  - Is this spam?
  - Is this a positive or a negative review?
  - To what synset does this word belong?
  - Is this a named entity?
  - What POS is this word?
  - What kind of named entity is this?

# Classification

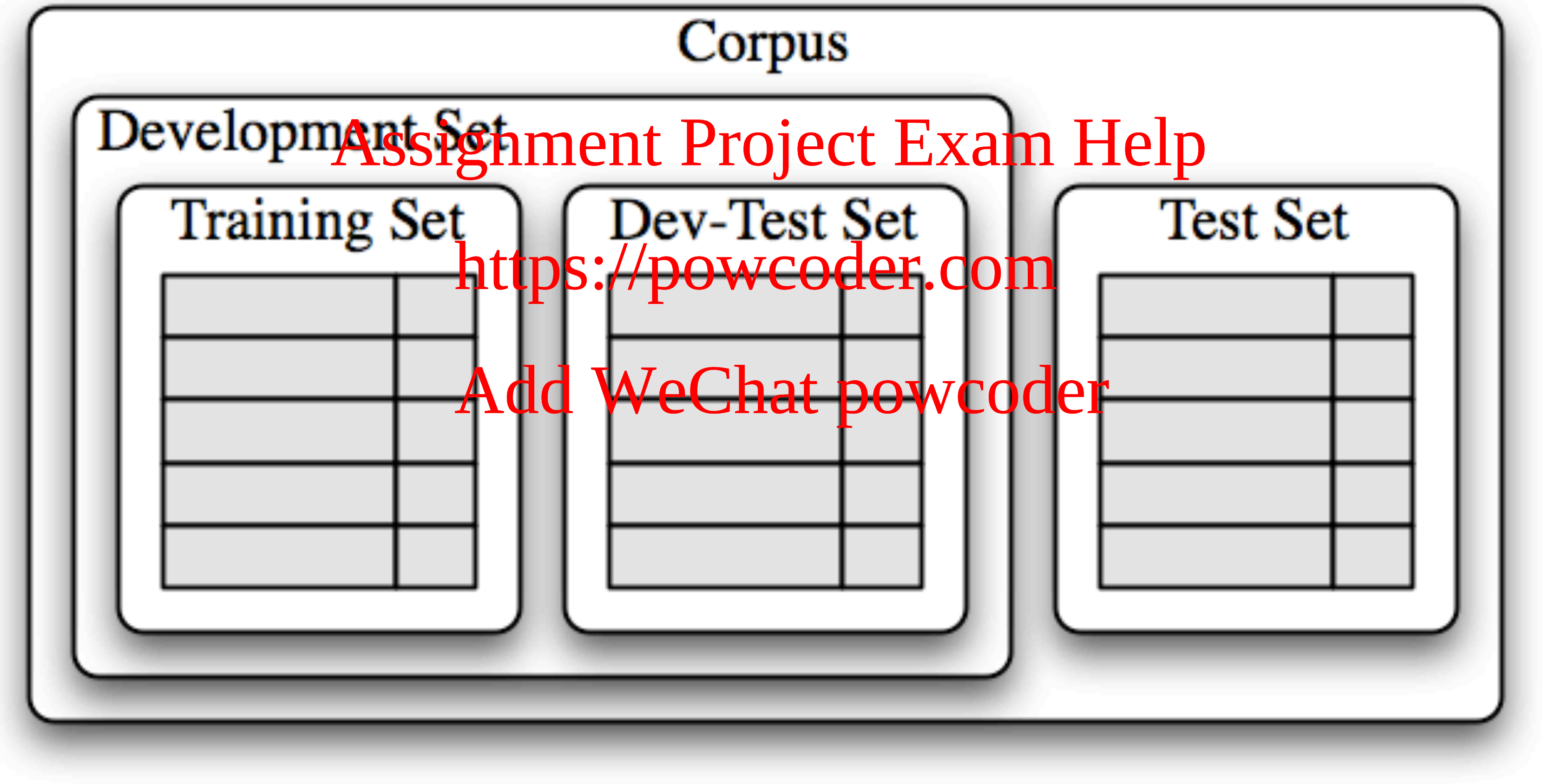
- Classifier can be rule-based
  - Date → Month DayOfMonth Year
- But most classifiers in use are machine learning classifiers
- The ones that we look at are supervised classifiers
  - That is, they use example data

# NLTK Classifier Example

# ML Classifiers



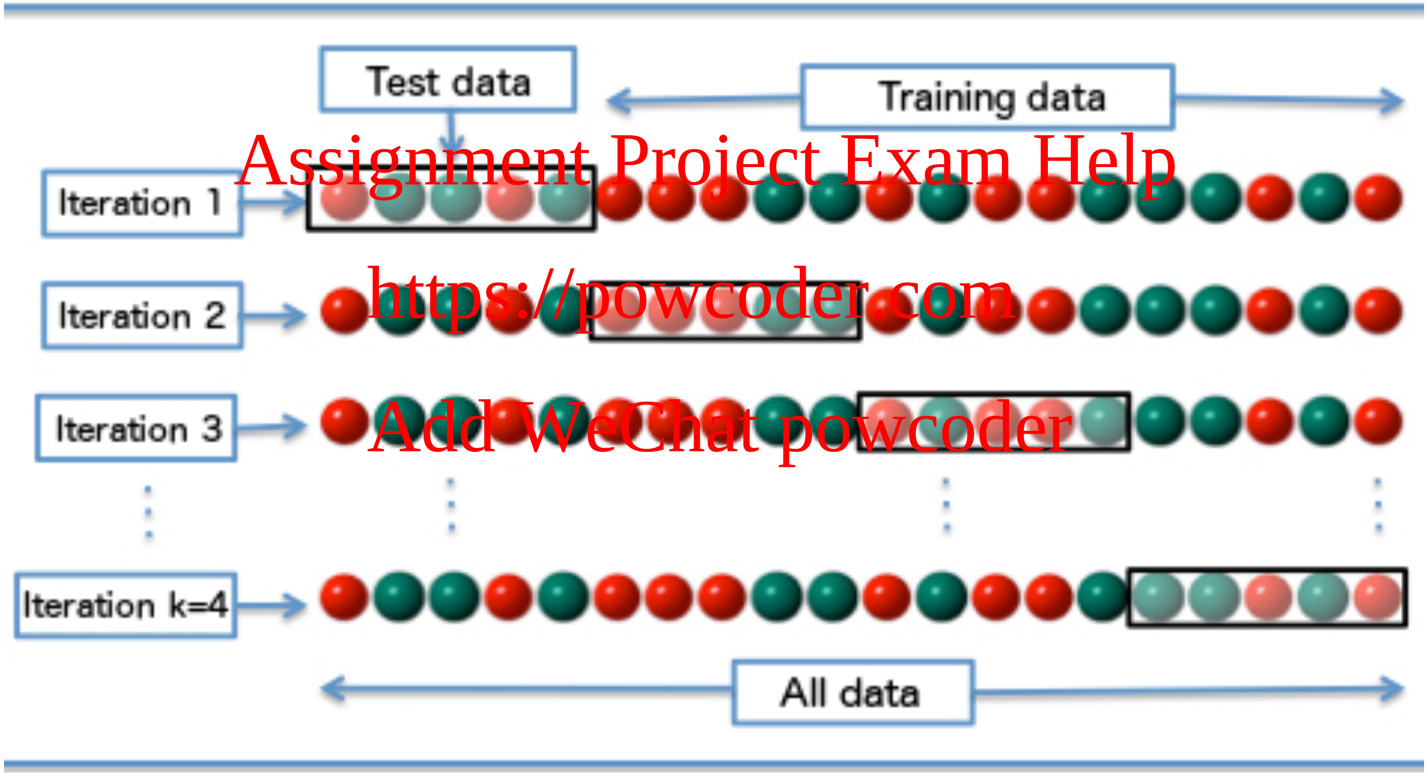
# Training versus testing



# Cross-validation

- Evaluate a model by partitioning the original sample into a training set to train the model, and a test set to evaluate it.
- N-fold cross-validation
  - partition into n equal size subsamples
  - a single subsample is retained as the test data, the remaining n-1 subsamples are used for training
  - Repeat n times with each subsample as the test data

# N-fold cross-validation



Source: wikipedia.org

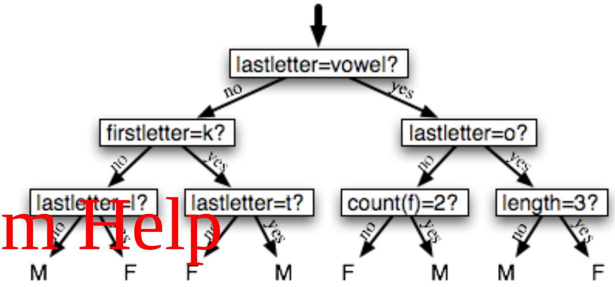
# N-fold cross-validation

- Matters less how the data are divided
- Test and training data need to be taken from the same data set
  - That is, test data and training data have the same characteristics and the overall data are homogeneous
- Avoid human bias
  - Splits have to be random
- Avoid dependencies between folds

# Classifiers

- Decision tree
- Bayesian classifier
- Maximum entropy classifier
- Neural networks and deep learning
  - Maybe towards the end of the course

# Decision Trees



- A flowchart-like structure in the shape of a tree

- Internal nodes are tests

  - Each test is a test for the value of a feature

  - Numbers of outcomes determines the number of branches from that node
- Leaf nodes have final classification
- Traditionally built manually

# Building a decision tree

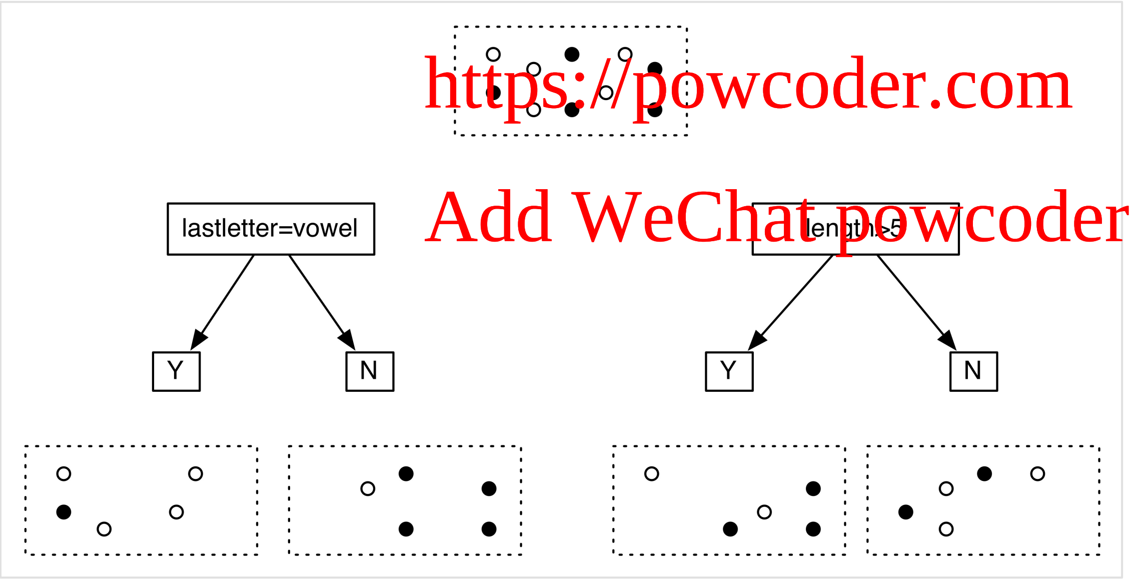
- Create decision stumps
  - A mini tree with just one test and branches depending on how many outcomes the test has
  - We could do one stump for each feature
    - lastletter=vowel
    - length>5
    - count(e)=2

*Note that the algorithm itself does not associate any semantics with these names*

  - If a decision stump implements a test for a binary features, then it splits your training corpus into two partitions (that is, the tree stump has two daughters)

# Building a decision tree

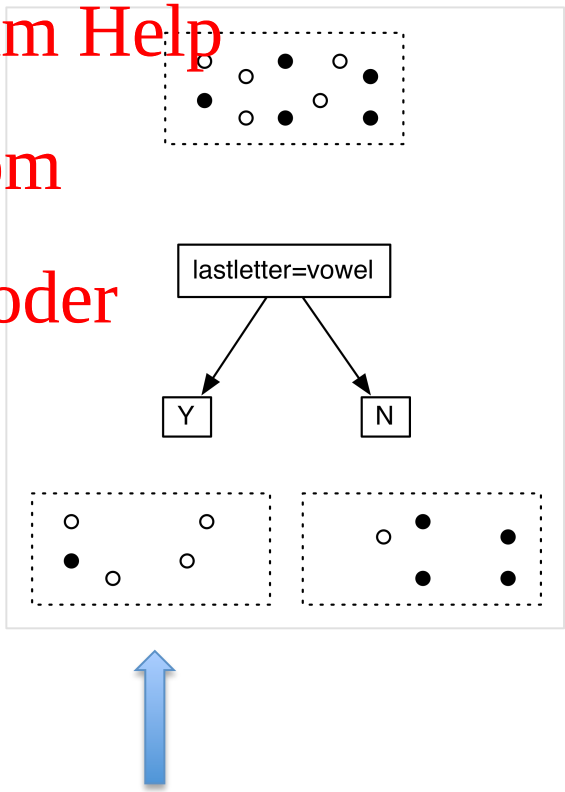
- Select the decision stump that has the highest capacity of predicting the final label



The lastletter=vowel feature performs better than the other feature so we pick it and insert it into the decision tree.

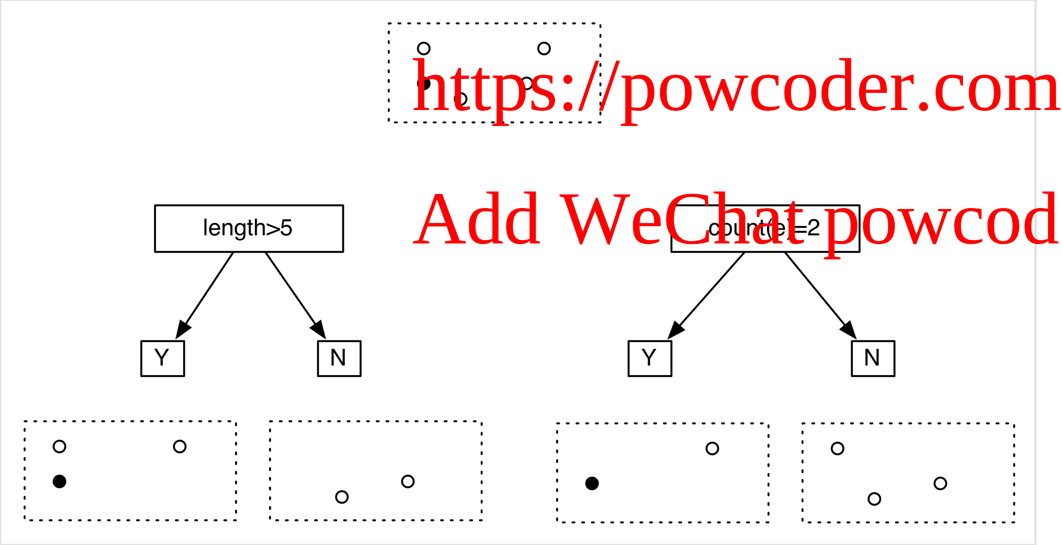
# Building a decision tree

- So we chose lastletter=vowel
- Now let's see if we can do something with that lower left distribution
- We now work with a domain of only five observations



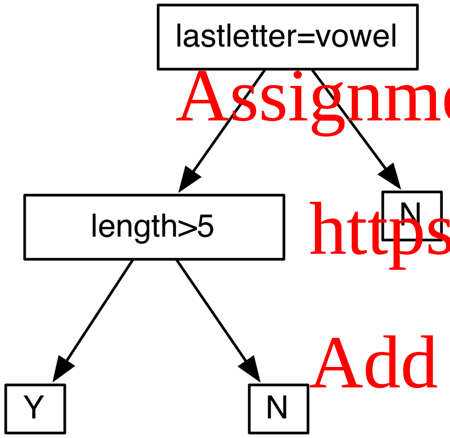
# Building a decision tree

- Again select the decision stump that has the highest capacity of predicting the final label



Hard to say which feature performs better, but let's say that the length>5 feature is better so we pick it and insert it into the decision tree.

# Building a decision tree



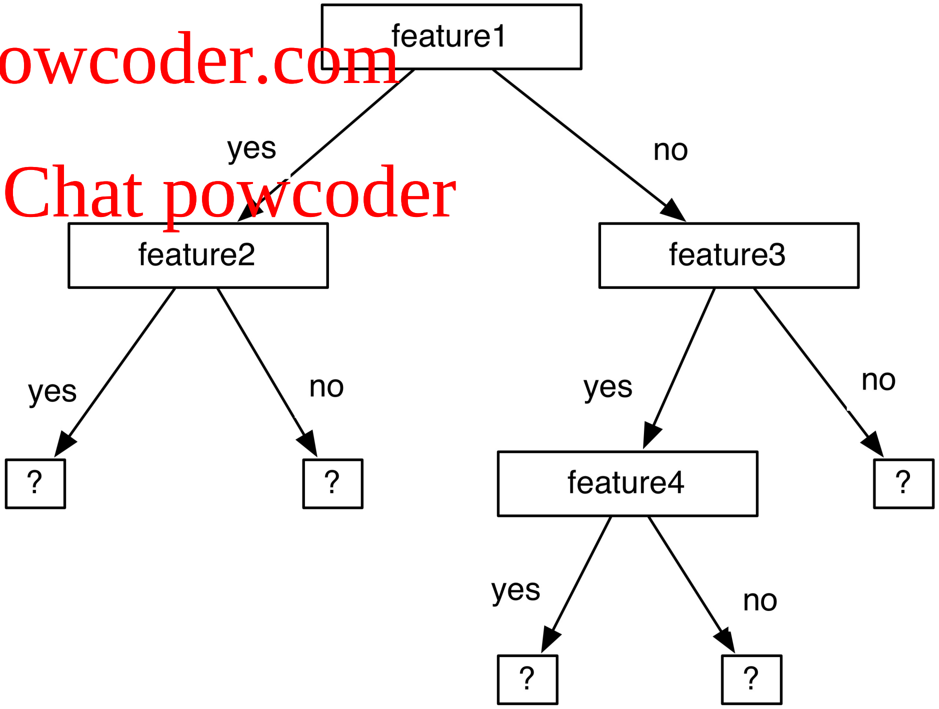
Partial tree after adding two stumps

You keep going till you can add no more rules or till adding rules makes no sense.

Danger of overfitting, that is, the tree gets to be too tailored to the training data. You could avoid this by not adding rules when your domain gets too small or by pruning the tree.

# Building a decision tree

- Iterative process starting at the very top
- At any point you have a partial tree
  - Select a ? node
  - Decide whether there is a good stump that can be added
  - Or replace with label

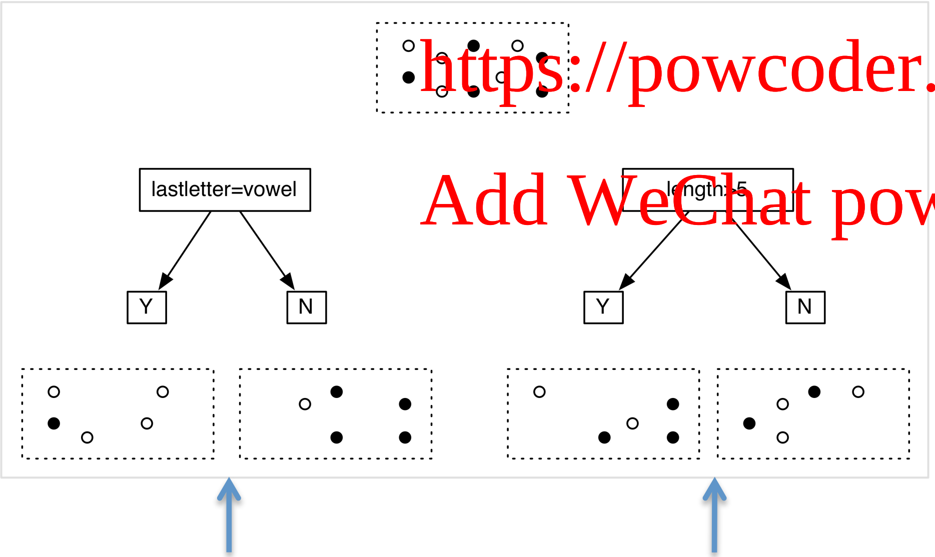


# How to choose the stump?

- Use accuracy
- Use entropy and information gain

# Accuracy in decision trees

- Calculate accuracy of each partitions and take the weighted average



Accuracy = 0.80

Accuracy = 0.40

# Entropy and information gain

- More popular way for choosing a stump
- How much more organized do the values get to be when you partition them with a given feature?
- Entropy is a measure where a low value reflects highly organized input and a high value reflects chaos.
  - Or better: low probability versus high probability
- You gain information if entropy goes down:
  - InfoGain(S1, S2) = Entropy(S1) – Entropy(S2)

# Entropy

$$H = -\sum_{l \in L} P(l) \times log_2 P(l)$$

where

$$\begin{aligned} L &= \text{the set of labels} \\ l &= \text{an element of } L \\ P(l) &= \text{the probability of } l \end{aligned}$$

Zero is lowest possible value and reflects that your output is completely homogeneous

Highest value depends on how many possible labels there are

# Example calculation

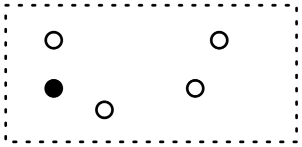
- Original set of observations had 5 white dots (female names) and 5 black dots (male names)

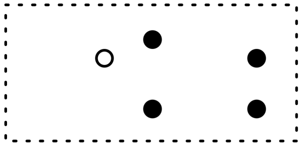
- Entropy:

$$H = -\sum_{l \in L} P(l) \times log_2 P(l)$$

= - (0.5 x log(0.5) + 0.5 x log(0.5))
= - (0.5 x -1.0 + 0.5 x -1.0)
= - (-0.5 + -0.5)
= - (-1)
= 1

# More calculations





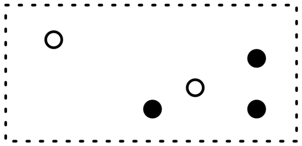
= - (0.2 x log(0.2) + 0.8 x log(0.8))

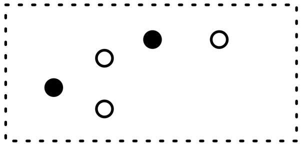
= - (0.2 x -2.322 + 0.8 x -0.322)

= - (-0.464 + -0.258)

= - (-0.722)

= 0.722





= - (0.4 x log(0.4) + 0.6 x log(0.6))

= - (0.4 x -1.322 + 0.6 x -0.737)

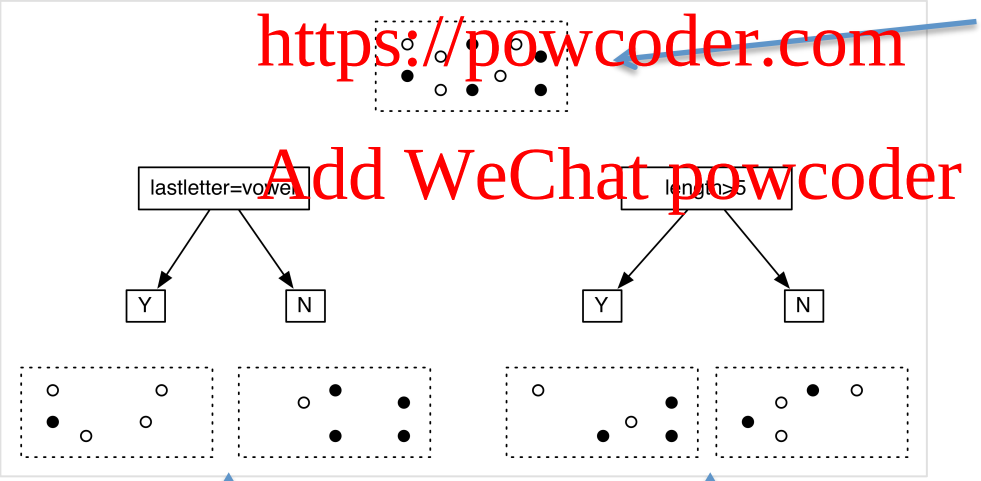
= - (-0.529 + -0.442)

= - (-0.971)

= 0.971

# Entropy in decision trees

- Calculate entropy of original set
- Calculate average entropy of the partitions



H = 1.00

H = 0.72
InfoGain = 1.00 – 0.72 = 0.28

H = 0.97
InfoGain = 1.00 – 0.97 = 0.03

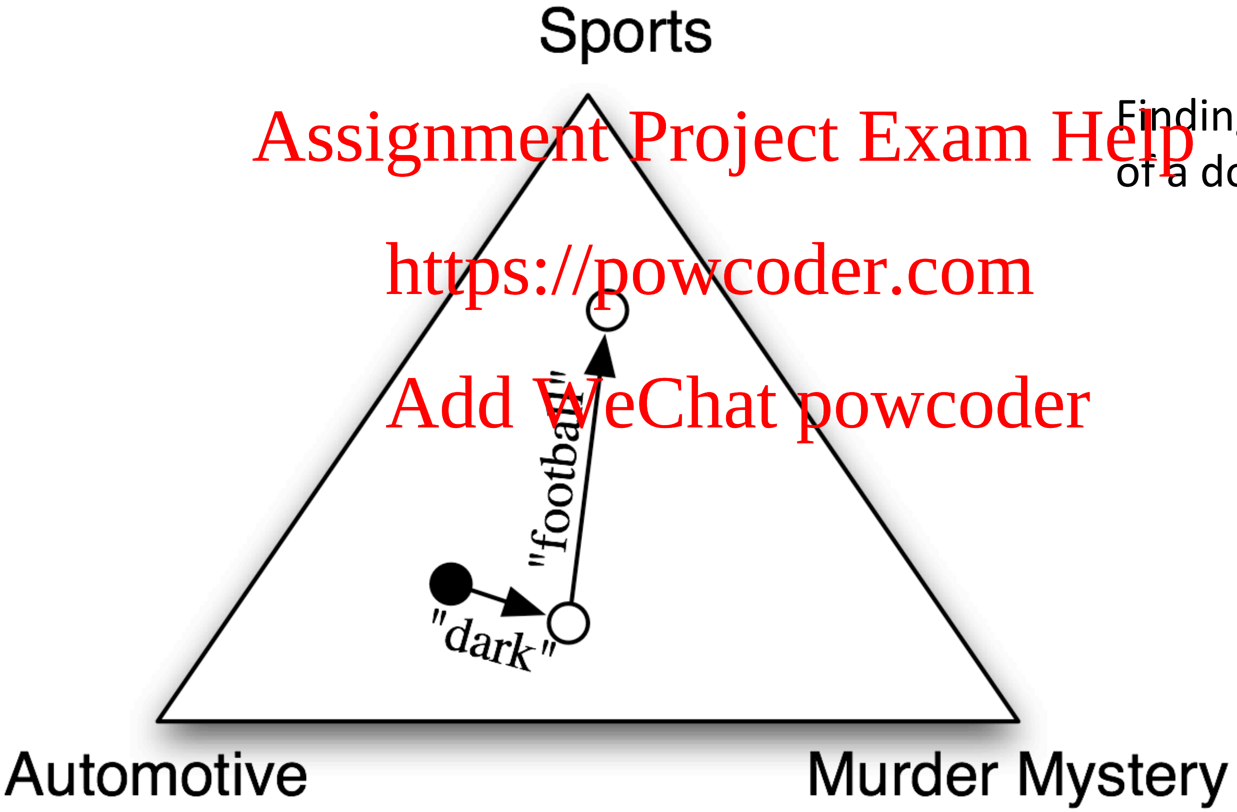
# Decision tree pros and cons

- Advantages
  - Easy to interpret
  - Good match for hierarchical classifications
- Disadvantages
  - Training sets for lower nodes may be too small
  - Features checked in particular order
  - Not enough space at top for “good” features
  - Features that are weak predictors will be ignored

# Naïve Bayes Classifier

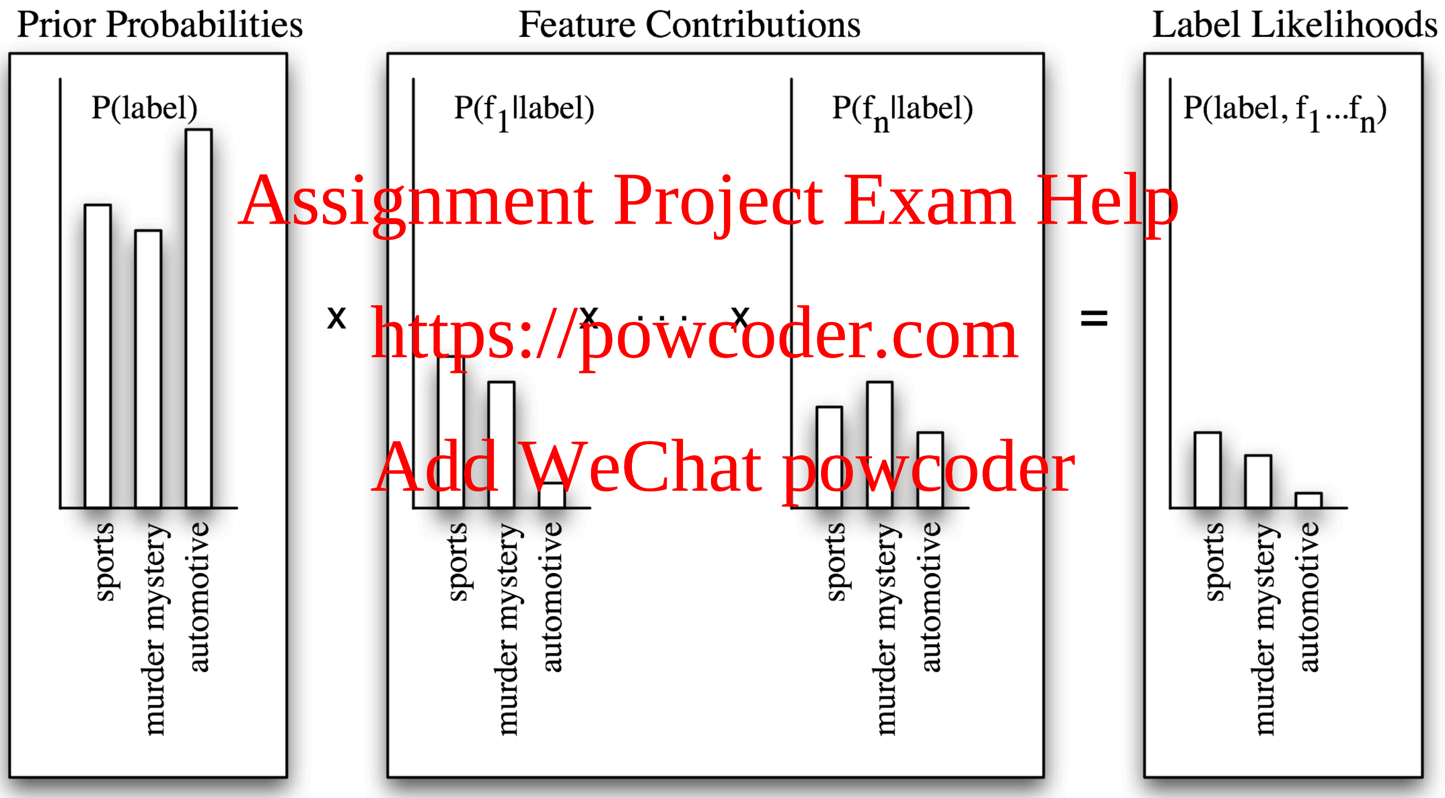
- Every feature counts
- Algorithm
  - Calculate prior probability of a label
    - check frequency in data set
  - For each feature
    - how do features change the probability of labels
    - changes the estimated likelihood of a label given a feature
  - Pick highest likelihood

# Naïve Bayes Classifier



Finding the topic of a document

# Naïve Bayes Classifier



Each feature reduces the likelihood that a label is true, but for some labels the likelihood will be reduced more

# Bayes Rule

$$P(C|F) = \frac{P(F|C) \times P(C)}{P(F)}$$

where

$$\begin{aligned} P(C|F) &= \text{the probability of class } C \text{ given the feature set } F \\ P(F|C) &= \text{the probability of the feature set } F \text{ given class } C \\ P(C) &= \text{the probability of the class } C \\ P(F) &= \text{the probability of the feature set } F \end{aligned}$$

$$P(C|F) = \frac{P(F_1|C) \times P(F_2|C) \times ... \times P(F_n|C) \times P(C)}{P(F_1) \times P(F_2) \times ... \times P(F_n)}$$

# Weather data example

| outlook | temperature | 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 |

from:
Witten & Frank
Data Mining

# Weather data example

What is the chance of play=yes/no with the following conditions

| **outlook** | **temperature** | **humidity** | **windy** | **play** |
|---|---|---|---|---|
| sunny | cool | high | true | ? |

Tabulate individual probabilities

| | yes | no |
|---|---|---|
| outlook=sunny | 2/9 | 3/5 |
| temperature=cool | 3/9 | 1/5 |
| humidity=high | 3/9 | 4/5 |
| windy=true | 3/9 | 3/5 |
| play | 9/14 | 5/14 |

P(outlook=sunny|play=no)
*It is sunny 3/5 times when we do not play*

P(play=no)
*We do not play 5/14 times*

# Weather data example

Calculate P(yes|F) and P(no|F)

| | yes | no |
|---|---|---|
| outlook=sunny | 2/9 | 3/5 |
| temperature=cool | 3/9 | 1/5 |
| humidity=high | 3/9 | 4/5 |
| windy=true | 3/9 | 3/5 |
| play | 9/14 | 5/14 |

$$P(C|F) = \frac{P(F_1|C) \times P(F_2|C) \times \ldots \times P(F_n|C) \times P(C)}{P(F_1) \times P(F_2) \times \ldots \times P(F_n)}$$

P(yes|F) = (2/9 * 3/9 * 3/9 * 3/9 * 9/14) / P(F) = 0.0053 / P(F)
P(no|F)  = (3/5 * 1/5 * 4/5 * 3/5 * 5/14) /P(F) = 0.0206 / P(F)

# What is so naïve about this?

- It is naïve because features are considered independent from each other
  - If features are dependent, then the results will get skewed

- Still, it performs reasonably well for many problems and it is often used as a baseline.

# What about scalar values?

- Say we have the temperature as a feature
- Take potentially infinite range of values and put them in bins
  - Do not use a bin for each value
  - Overfitting
- Or use the scalar values and calculate averages and standard deviations for the values for some label

# What about zero values?

- Example
  - $F_n$ → outlook=overcast<br>
    C → play=yes
  - $P(F_n|C) = Count(F_n+C) / Count(C) = 0 / 5$
- Adjust the formula
  - $P(F_n|C) = Count(F_n+C) + 1 / Count(C) = 1 / 5$
  - similar to Laplace smoothing