Entropy and Information Gain

- The entropy (very common in Information Theory) characterizes the (im)purity of an arbitrary collection of examples

- Information Gain is the expected reduction in entropy caused by partitioning the examples according to a given attribute

Entropy Calculations

If we have a set with k different values in it, we can calculate the entropy as follows:

$$entropy(Set) = I(Set) = -\sum_{i=1}^{k} P(value_i) \cdot \log_2(P(value_i))$$

Where $P(value_i)$ is the probability of getting the i^{th} value when randomly selecting one from the set.

So, for the set R = {a,a,a,b,b,b,b,b}

$$entropy(R) = I(R) = -\left[\underbrace{\left(\frac{3}{8}\right)\log_2\left(\frac{3}{8}\right)}_{\text{a-values}} + \underbrace{\left(\frac{5}{8}\right)\log_2\left(\frac{5}{8}\right)}_{\text{b-values}} \right]$$

Looking at some data

Color	Size	Shape	Edible?
Yellow	Small	Round	+
Yellow	Small	Round	-
Green	Small	Irregular	+
Green	Large	Irregular	-
Yellow	Large	Round	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Green	Small	Round	-
Yellow	Large	Round	-
Yellow	Large	Round	+
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Small	Irregular	+
Yellow	Large	Irregular	+

Entropy for our data set

- 16 instances: 9 positive, 7 negative.

$$I(all\_data) = -\left[\left(\frac{9}{16}\right)\log_2\left(\frac{9}{16}\right) + \left(\frac{7}{16}\right)\log_2\left(\frac{7}{16}\right)\right]$$

- This equals: 0.9836
- This makes sense – it's almost a 50/50 split; so, the entropy should be close to 1.

Visualizing Information Gain

Size

Small | Large

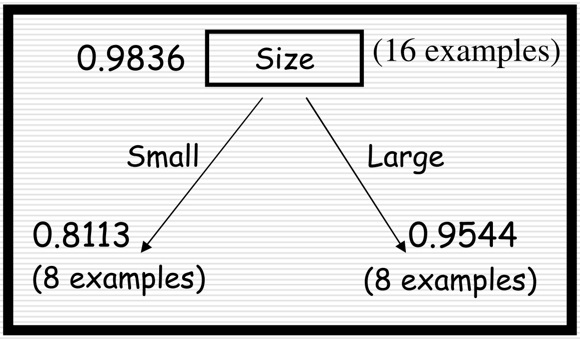
Color	Size	Shape	Edible?
Yellow	Small	Round	+
Yellow	Small	Round	-
Green	Small	Irregular	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Yellow	Small	Round	+
Green	Small	Round	-
Yellow	Small	Irregular	+

Color	Size	Shape	Edible?
Green	Large	Irregular	-
Yellow	Large	Round	+
Yellow	Large	Round	-
Yellow	Large	Round	+
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Large	Round	-
Yellow	Large	Irregular	+

$$G(S,A) = I(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} I(S_v)$$

Visualizing Information Gain

The data set that goes down each branch of the tree has its own entropy value. We can calculate for each possible attribute its **expected entropy**. This is the degree to which the entropy would change if branch on this attribute. You **add** the entropies of the two children, **weighted** by the proportion of examples from the parent node that ended up at that child.



Entropy of left child is 0.8113
I(size=small) = 0.8113

Entropy of right child is 0.9544
I(size=large) = 0.9544

$I(S_{Size})$ = (8/16)*.8113-(8/16)*.9544 = .8828

$G(attrib) = I(parent) - I(attrib)$

We want to calculate the *information gain* (or entropy reduction). This is the reduction in 'uncertainty' when choosing our first branch as 'size'. We will represent information gain as "*G*."

$$G(size) = I(S) - I(S_{Size})$$
$$G(size) = 0.9836 - 0.8828$$
$$G(size) = 0.1008$$

Entropy of all data at parent node = I(parent) = 0.9836

Child's expected entropy for '**size**' split = I(size) = 0.8828

So, we have gained 0.1008 *bits* of information about the dataset by choosing 'size' as the first branch of our decision tree.

Example-based Classifiers

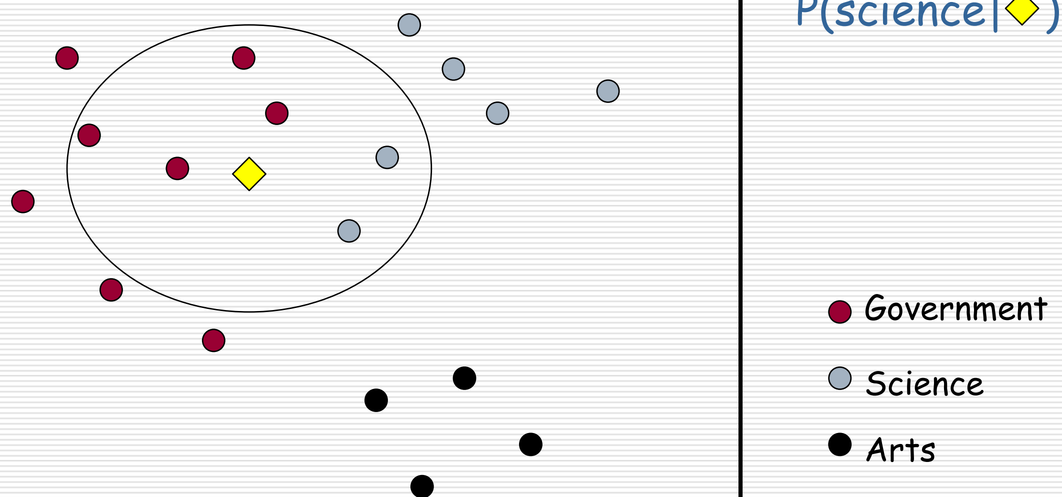
- Example-based classifiers (EBCs) learns from the categories of the training documents similar to the one to be classified
- The most frequently used EBC is the k-NN algorithm

k Nearest Neighbor Classification

1. To classify document *d* into class c
2. Define *k*-neighborhood N as *k* nearest neighbors (according to a given distance or similarity measure) of *d*
3. Count number of documents k_c in N that belong to c
4. Estimate P(c|*d*) as k_c/k
5. Choose as class argmax_c P(c|*d*)   [= majority class]

Example: k=6 (6NN)

P(science|◇)?



Nearest-Neighbor Learning Algorithm

- Learning is just storing the representations of the training examples in *D*.
- Testing instance *x*:
  - Compute similarity between *x* and all examples in *D*.
  - Assign *x* the category of the most similar example in *D*.
- Does not explicitly compute a generalization or category prototypes.
- Also called:
  - Case-based learning
  - Memory-based learning
  - Lazy learning

kNN Is Close to Optimal

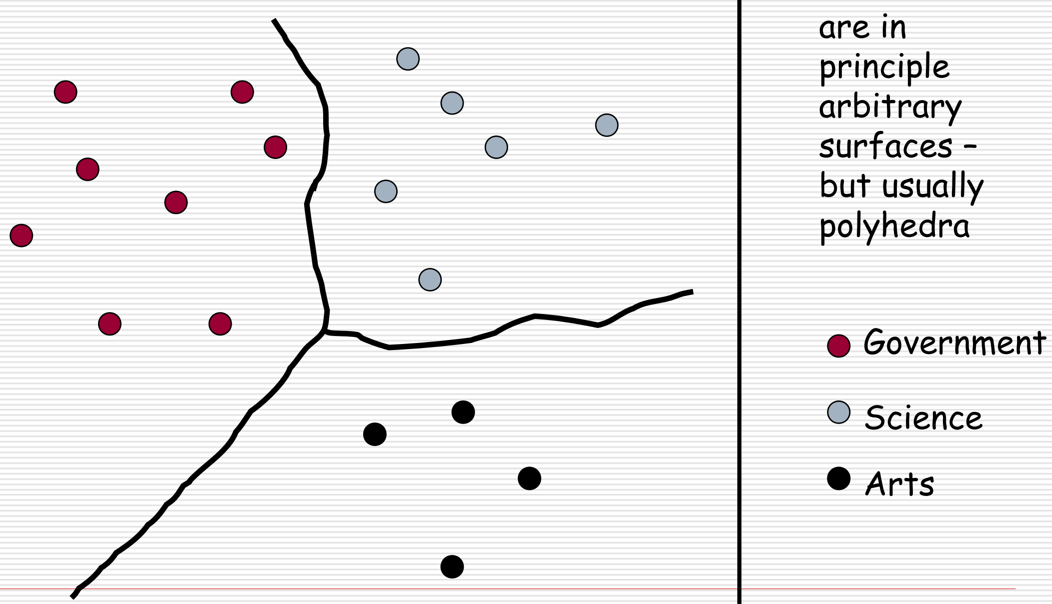
- Asymptotically, the error rate of 1-nearest-neighbor classification is less than twice the Bayes error [error rate of classifier knowing the model that generated data]

- Asymptotic error rate is 0 whenever Bayes error is 0.

K Nearest-Neighbor

- Using only the closest example to determine the categorization is subject to errors due to:
  - A single atypical example.
  - Noise (i.e. error) in the category label of a single training example.
- More robust alternative is to find the *k* most-similar examples and return the majority category of these *k* examples.
- Value of *k* is typically odd to avoid ties;

kNN decision boundaries



Boundaries are in principle arbitrary surfaces – but usually polyhedra

Similarity Metrics

- Nearest neighbor method depends on a similarity (or distance) metric.
- Simplest for continuous *m*-dimensional instance space is *Euclidian distance*.
- Simplest for *m*-dimensional binary instance space is *Hamming distance* (number of feature values that differ).
- For text, cosine similarity of tf.idf weighted vectors is typically most effective.
- .. But any metric can be used!!

Nearest Neighbor with Inverted Index

- Naively finding nearest neighbors requires a linear search through $|D|$ documents in collection
- But determining *k* nearest neighbors is the same as determining the *k* best retrievals using the test document as a query to a database of training documents.
- Use standard vector space inverted index methods to find the *k* nearest neighbors.
- Testing Time: $O(B|V_t|)$ where B is the average number of training documents in which a test-document word appears.
  - Typically $B \ll |D|$

kNN: Discussion

- No feature selection necessary
- Scales well with large number of classes
  - Don't need to train *n* classifiers for *n* classes
- Classes can influence each other
  - Small changes to one class can have ripple effect
- Scores can be hard to convert to probabilities
- No training necessary

The Rocchio Method

- The Rocchio Method is an adaptation to TC of Rocchio's formula for relevance feedback. It computes a profile for c_i by means of the formula

$$w_{ki} = \frac{1}{|POS|} \sum_{d_j \in POS} w_{kj} - \delta \frac{1}{|NEG|} \sum_{d_j \in NEG} w_{kj}$$

  where POS = $\{d_j \in Tr | y_j=+1\}$ NEG = $\{d_j \in Tr | y_j=-1\}$, and δ may be seen as the ratio between γ and β parameters in RF

- In general, Rocchio rewards the similarity of a test document to the centroid of the positive training examples, and its dissimilarity from the centroid of the negative training examples
- Typical choices of the control parameter δ are $0 \leq \delta \leq .25$

Rocchio: a simple case study

When δ=0

- For each category (possibly, more than one), compute a *prototype* vector by summing the vectors of the training documents in the category.
  - Prototype = centroid of members of class
- Assign test documents to the category with the closest prototype vector based on cosine similarity.

Rocchio Time Complexity

- Note: The time to add two sparse vectors is proportional to minimum number of non-zero entries in the two vectors.
- Training Time: $O(|D|(L_d + |V_d|)) = O(|D|\ L_d)$ where L_d is the average length of a document in D and V_d is the average vocabulary size for a document in D.
- Test Time: $O(L_t + |C||V_t|)$ where L_t is the average length of a test document and $|V_t|$ is the average vocabulary size for a test document.
  - Assumes lengths of **centroid** vectors are computed and stored during training, allowing cosSim(**d**, **c**$_i$) to be computed in time proportional to the number of non-zero entries in **d** (i.e. $|V_t|$)