Categorization

Jaimie Murdock

IU Cognitive Science Program
810 Eigenmann Hall
jammurdo@indiana.edu

November 15, 2011

"Issues related to concepts and categorization are nearly ubiquitous in psychology because of people's natural tendancy to perceive a thing as something." - Goldstone & Kersten 2003

Different names for the same thing...

- Categorization
- Classification (machine learning)
- Clustering (data mining)
- Partitioning (mathematics)

Different names for the same thing...

- Categorization
- Classification (machine learning)
- Clustering (data mining)
- Partitioning (mathematics)
- Chunking (memory)
- Object Recognition (vision)
- Semantics (linguistics)
- Named Entity Recognition (natural language processing)

What is categorization?

- The assignment of concepts to categories
- "Seeing something as X" - Wittgenstein, *Philosophical Investigations*

What is categorization?

- The assignment of concepts to categories
- "Seeing something as X" - Wittgenstein, *Philosophical Investigations*

- **What is a concept?**
  Whatever psychological state signifies thoughts of X
- **What is a category?**
  All entities that are appropriately categorized as X

Prototypes vs. Exemplars

Prototype Model
Do concepts determine categories?
(Lakoff 1987)

Prototypes vs. Exemplars

Prototype Model
Do concepts determine categories?
(Lakoff 1987)

Exemplar Model
Do categories determine concepts?
(Nosofsky 1984)

Why do we categorize?

- Components of thought

Why do we categorize?

- Components of thought
- Inductive Predictions

Why do we categorize?

- Components of thought
- Inductive Predictions
- Communication

Why do we categorize?

- Components of thought
- Inductive Predictions
- Communication
- Cognitive Economy

Equivilence Classes

Distinguishable stimuli can become treated as the same thing once they are placed in the same category (Sidman 1994)

Equivilence Classes

Distinguishable stimuli can become treated as the same thing once they are placed in the same category (Sidman 1994)

Biology - taxonomy (kingdom, phylum, class, order, family, genus, species)

Equivilence Classes

Distinguishable stimuli can become treated as the same thing once they are placed in the same category (Sidman 1994)

Biology - taxonomy (kingdom, phylum, class, order, family, genus, species)

Things to remove from a burning house - photos, babies, cats

Equivilence Classes

Distinguishable stimuli can become treated as the same thing once they are placed in the same category (Sidman 1994)

Biology - taxonomy (kingdom, phylum, class, order, family, genus, species)

Things to remove from a burning house - photos, babies, cats

Equivilence classes may not be uniquely human - sea lions (Schusterman, Reichmuth, Kastak 2000)

Representations

How are categories represented?

- rules

Representations

How are categories represented?

- rules
- exemplars

Representations

How are categories represented?

- rules
- exemplars
- prototypes

Representations

How are categories represented?

- rules
- exemplars
- prototypes
- boundaries

Algorithms

A process to assign concepts to categories

Algorithms

A process to assign concepts to categories

- k-means Nearest Neighbors (MacQueen 1967)
- QT-clust (Heyer et al. 1999)
- Information Theoretic Clustering (Gokcay & Principé 2002)

Components

Two key decisions in clustering:

distance function

- Euclidean distance
- semantic similarity
- cross-entropy

cluster assignment

- nearest neighbor
- minimal diameter
- maximize cross-cluster distance

k-means Nearest Neighbors

Given n items, place into k groups

Initialize: Pick k centroids
Assign: Assign items to nearest centroid
Update: Recalculate centroids

Repeat until convergence of assignment

Data Structures

```python
class Concept(object):
    # initialize the wrapper
    def __init__(self, value):
        self.value = value
        self._cluster = None
        self._previous_cluster = None

    # Properties to track cluster assignments
    @property
    def cluster(self):
        return self._cluster

    @cluster.setter
    def cluster(self, value):
        """ Track previous cluster assignment auto-magically. """
        self._previous_cluster = self._cluster
        self._cluster = value
```

The Main Loop

```python
def kmeans(k, population, min_delta=0):
    # initialize centroids with random members of the population
    centroids = [random.choice(population).value for i in range(k)]

    # delta is the number of elements that switch cluster
    # since all members will be changing on the first round,
    # we initialize delta to the length of the population
    delta = len(population)

    # test for convergence
    while delta > min_delta:
        # assign population to clusters
        assign_clusters(population, centroids)

        # get number of elements which switched cluster
        delta = len([x for x in population
                          if x.cluster != x.previous_cluster])

        # update centroids
        centroids = update_centroids(population, centroids)
```

Assignment

```python
def assign_clusters(population, centroids):
    for x in population:
        # calculate distance to each centroid
        distances = [distance(x.value, centroid)
                         for centroid in centroids]

        # select the index (cluster id) of the closest cluster
        cluster, min_distance = min(enumerate(distances),
                                    key=itemgetter(1))

        # assign the object to that cluster
        x.cluster = cluster
```

Update

```
def get_centroids(population, k):
    new_centroids = []

    for cluster in range(k):
        # filter out the cluster population
        cluster_values = [x.value for x in population
                              if x.cluster == cluster]
        # generate the new centroid by taking the average of all exemplars
        # comprising the cluster. First, sum the dimensions:
        centroid = map(sum, zip(*cluster_values))

        # then take the average:
        centroid = [dimension / len(cluster_values)
                        for dimension in centroid]

        # add to our list of new centroids
        new_centroids.append(centroid)

    return new_centroids
```

The Main Loop

```
def kmeans(k, population, min_delta=0):
    # initialize centroids with random members of the population
    centroids = [random.choice(population).value for i in range(k)]

    # delta is the number of elements that switch cluster
    # since all members will be changing on the first round,
    # we initialize delta to the length of the population
    delta = len(population)

    # test for convergence
    while delta > min_delta:
        # assign population to clusters
        assign_clusters(population, centroids)

        # get number of elements which switched cluster
        delta = len([x for x in population
                        if x.cluster != x.previous_cluster])

        # update centroids
        centroids = update_centroids(population, centroids)
```

QT–clust

Given n items, place into groups of ϵ diameter

Build: for each $i \in n$, build candidate cluster C_i
Select: pick largest C_i, remove elements from popuation

Repeat until all items are assigned.

Data Structures

```
class Concept(object):
    # initialize the wrapper
    def __init__(self, value):
        self.value = value
        self.cluster = [self]
        self.diameter = 0.0

    def _diameter_append(self, candidate):
        return max([distance(candidate.value, x.value)
                        for x in self.cluster])
```

The Main Loop

```
def qt_clust(thresh, population):
    while population:
        # build candidate clusters
        for x in population:
            x.build_cluster(population, thresh)

        # select the largest candidate cluster
        candidate = max(population, key=lambda x: len(x.cluster))

        # remove elements from the population
        for x in candidate.cluster:
            population.remove(x)

        yield candidate
```

Build

```python
class Concept(object):
    def build_cluster(self, population, thresh):
        # initialize the cluster
        self.cluster = [self]
        population.remove(self)

        while population:
            # find x, such that cluster diameter is minimized
            x = min(population, key=self._diameter_append)
            new_diameter = self._diameter_append(x)

            # if below quality threshold, append to cluster
            if new_diameter < thresh:
                self.cluster.append(x)
                self.diameter = new_diameter
                population.remove(x)
            else:
                # otherwise terminate the loop
                break
```

The Main Loop

```python
def qt_clust(thresh, population):
    while population:
        # build candidate clusters
        for x in population:
            x.build_cluster(population, thresh)

        # select the largest candidate cluster
        candidate = max(population, key=lambda x: len(x.cluster))

        # remove elements from the population
        for x in candidate.cluster:
            population.remove(x)

        yield candidate
```

Given n items, place into k groups, minimizing the value of the cross-entropy function (CEF)

Initialize: Assign all items to random clusters **Group:** for each $i \in n$, build group G_i of size $M = n/k$
Reassign: for each $i \in n$, see if switching G_i reduces CEF, permanently switch cluster assignment for $x \in G_i$ which minimizes CEF

Repeat until CEF reaches minima

Data Structures

```python
class Concept(object):
    # initialize the wrapper
    def __init__(self, value):
        self.value = value
        self.cluster = None
        self.group = [self]

    def _diameter_append(self, candidate):
        return min([distance(candidate.value, x.value)
                       for x in self.group])
```

The Main Loop

```
def info_theory(k, population):
    # assign random clusters
    for x in population:
        x.cluster = random.choice(range(k))

    # create initial group size
    group_size = len(population) / k

    # grow group size exponentially while hill-climbing
    while group_size < len(population):
        # assign groups
        for x in population:
            x.make_group(population, group_size)

        hillclimb(population, group_size, k)
        group_size *= 2
```

Hill Climbing

```
def hillclimb(population, group_size, k):
    orig_CEF = 1.0                     # get baseline CEF
    min_CEF = CEF(population, k)       # initial min CEF

    while orig_CEF != min_CEF:
        for x in population:
            for member in x.group:
                # change cluster assignment
                member.cluster = x.cluster

            group_CEF = CEF(population, k)
            if group_CEF < min_CEF:
                # if CEF decreases, new min!
                min_CEF = group_CEF
            else:
                # restore previous cluster assignment
                for member in x.group:
                    member.cluster = x.previous_cluster

        orig_CEF = min_CEF        # set orig to new minima
```

The Main Loop

```
def info_theory(k, population):
    # assign random clusters
    for x in population:
        x.cluster = random.choice(range(k))

    # create initial group size
    group_size = len(population) / k

    # grow group size exponentially while hill-climbing
    while group_size < len(population):
        # assign groups
        for x in population:
            x.make_group(population, group_size)

        hillclimb(population, group_size, k)
        group_size *= 2
```

Categorization

Jaimie Murdock

IU Cognitive Science Program
810 Eigenmann Hall
jammurdo@indiana.edu

November 15, 2011