Reminder: Clustering

• Or, discovering similarities between objects
  – Individuals, Documents, ...
• Applications:
  – Recommender systems
  – Document organization
Recommending: restaurants

- We have a list of all Wivenhoe restaurants
  - with ratings for some
  - as provided by some Uni Essex students / staff
- Which restaurant(s) should I recommend to you?

### Input

<table>
<thead>
<tr>
<th>Name</th>
<th>Restaurant</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Bakehouse</td>
<td>Yes</td>
</tr>
<tr>
<td>Bob</td>
<td>The Flag</td>
<td>No</td>
</tr>
<tr>
<td>Cindy</td>
<td>Black Buoy</td>
<td>No</td>
</tr>
<tr>
<td>Dave</td>
<td>The Flag</td>
<td>Yes</td>
</tr>
<tr>
<td>Alice</td>
<td>Black Buoy</td>
<td>No</td>
</tr>
<tr>
<td>Estie</td>
<td>The Greyhound</td>
<td>Yes</td>
</tr>
<tr>
<td>Cindy</td>
<td>The Greyhound</td>
<td>No</td>
</tr>
<tr>
<td>Dave</td>
<td>Bengal Spice</td>
<td>No</td>
</tr>
<tr>
<td>Dave</td>
<td>The Greyhound</td>
<td>Yes</td>
</tr>
<tr>
<td>Estie</td>
<td>The Flag</td>
<td>Yes</td>
</tr>
<tr>
<td>Fred</td>
<td>Bengal Spice</td>
<td>No</td>
</tr>
<tr>
<td>Alice</td>
<td>Jardine</td>
<td>No</td>
</tr>
<tr>
<td>Fred</td>
<td>Rose and Crown</td>
<td>No</td>
</tr>
<tr>
<td>Dave</td>
<td>On the Corner</td>
<td>Yes</td>
</tr>
<tr>
<td>Bob</td>
<td>Valentino's</td>
<td>Yes</td>
</tr>
<tr>
<td>Estie</td>
<td>Black Buoy</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Algorithm 0

- Recommend to you the most popular restaurants
  - say # positive votes minus # negative votes
- Ignores your culinary preferences
  - *And* judgements of those with similar preferences
- How can we exploit the wisdom of “like-minded” people?

Another look at the input - a matrix

<table>
<thead>
<tr>
<th></th>
<th>Bengal Spice</th>
<th>Valentina's</th>
<th>Mango's</th>
<th>Bakehouse</th>
<th>The Greyhound</th>
<th>The Flag</th>
<th>Rose and Crown</th>
<th>Black Buoy</th>
<th>On the Corner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Cindy</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Dave</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Estle</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Fred</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Now that we have a matrix

<table>
<thead>
<tr>
<th>Bengal</th>
<th>Valentino's</th>
<th>Jardine</th>
<th>Bakehouse</th>
<th>Greyhound</th>
<th>The Flag</th>
<th>Rose and Crown</th>
<th>The Black Buoy</th>
<th>On the corner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spice</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ab</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dy</td>
<td></td>
<td>1</td>
<td></td>
<td>-1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ve</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>le</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ed</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

View all other entries as zeros for now.

PREFERENCE-DEFINED DATA SPACE
Similarity between two people

• Similarity between their preference vectors.
• Inner products are a good start.
• Dave has similarity 3 with Estie
  – but -2 with Cindy.
• Perhaps recommend Black Buoy to Dave
  – and Bakehouse to Bob, etc.

Algorithm 1.1

• You give me your preferences and I need to give you a recommendation.
• I find the person “most similar” to you in my database and recommend something he likes.
• Aspects to consider:
  – No attempt to discern cuisines, etc.
  – What if you’ve been to all the restaurants he has?
  – Do you want to rely on one person’s opinions?
Algorithm 1.k

• You give me your preferences and I need to give you a recommendation.
• I find the $k$ people “most similar” to you in my database and recommend what’s most popular amongst them.
• Issues:
  – A priori unclear what $k$ should be
  – Risks being influenced by “unlike minds”

Slightly more sophisticated attempt

• Group similar users together into clusters
• You give your preferences and seek a recommendation, then
  – Find the “nearest cluster” (what’s this?)
  – Recommend the restaurants most popular in this cluster
• Features:
  – avoids data sparsity issues
  – still no attempt to discern why you’re recommended what you’re recommended
  – how do you cluster?
CLUSTERS

• Can cluster

How do you cluster?

• Must keep similar people together in a cluster
• Separate dissimilar people
• Factors:
  – Need a notion of similarity/distance
  – Vector space? Normalization?
  – How many clusters?
    • Fixed a priori?
    • Completely data driven?
  – Avoid “trivial” clusters - too large or small
Looking beyond

Clustering people for restaurant recommendations

Clustering other things (documents, web pages)

Other approaches to recommendation

General unsupervised machine learning.

Text clustering

• Search results clustering
• Document clustering
Navigating search results

• Given the results of a search (say *jaguar*), partition into groups of related docs
  – sense disambiguation
• Approach followed by Uni Essex site
  – Kruschwitz / al Bakouri / Lungley
• Other examples: IBM InfoSphere Data Explorer
  – *(was: vivisimo.com)*

Results list clustering example

**Cluster 1:**
- Jaguar Motor Cars' home page
- Mike's XJS resource page
- Vermont Jaguar owners' club

**Cluster 2:**
- Big cats
- My summer safari trip
- Pictures of jaguars, leopards and lions

**Cluster 3:**
- Jacksonville Jaguars' Home Page
- AFC East Football Teams
Search results clustering: example

Why cluster documents?

- For improving recall in search applications
- For speeding up vector space retrieval
- Corpus analysis/navigation
  - Sense disambiguation in search results
Improving search recall

- **Cluster hypothesis** - Documents with similar text are related
- Ergo, to improve search recall:
  - Cluster docs in corpus a priori
  - When a query matches a doc $D$, also return other docs in the cluster containing $D$
- Hope: docs containing *automobile* returned on a query for *car* because
  - clustering grouped together docs containing *car* with those containing *automobile*.

Why might this happen?

Speeding up vector space retrieval

- In vector space retrieval, must find nearest doc vectors to query vector
- This would entail finding the similarity of the query to every doc - slow!
- By clustering docs in corpus a priori
  - find nearest docs in cluster(s) close to query
  - inexact but avoids exhaustive similarity computation

Exercise: Make up a simple example with points on a line in 2 clusters where this inexactness shows up.
Corpus analysis/navigation

• Given a corpus, partition it into groups of related docs
  – Recursively, can induce a tree of topics
  – Allows user to browse through corpus to home in on information
  – Crucial need: meaningful labels for topic nodes.

CLUSTERING DOCUMENTS IN A (VERY) LARGE COLLECTION: GOOGLE NEWS
CLUSTERING DOCUMENTS IN A VERY LARGE COLLECTION: JRC’S NEWS EXPLORER

Setup

- Given “training” docs for each category
  - Theory, AI, NLP, etc.
- Cast them into a decision space
  - generally a vector space with each doc viewed as a bag of words
- Build a classifier that will classify new docs
  - Essentially, partition the decision space
- Given a new doc, figure out which partition it falls into
Supervised vs. unsupervised learning

• This setup is called *supervised learning* in the terminology of Machine Learning
• In the domain of text, various names
  – Text classification, text categorization
  – Document classification/categorization
  – “Automatic” categorization
  – Routing, filtering ...
• In contrast, the earlier setting of clustering is called *unsupervised learning*
  – Presumes no availability of training samples
  – Clusters output may not be thematically unified.

What makes docs “related”?

• Ideal: semantic similarity.
• Practical: statistical similarity
  – We will use cosine similarity.
  – Docs as vectors.
  – For many algorithms, easier to think in terms of a distance (rather than similarity) between docs.
  – We will describe algorithms in terms of cosine similarity.
DOCUMENTS AS BAGS OF WORDS

Doc as vector

- Each doc \( j \) is a vector of \( tfidf \) values, one component for each term.
- Can normalize to unit length.
- So we have a vector space
  - terms are axes - aka \textit{features}
  - \( n \) docs live in this space
  - even with stemming, may have 10000+ dimensions
  - do we really want to use all terms?
TERM WEIGHTING IN VECTOR SPACE MODELS: THE TF.IDF MEASURE

\[ tfidf_{i,k} = f_{i,k} \times \log \left( \frac{N}{df_i} \right) \]

**FREQUENCY** of term \( i \) in document \( k \)

**Number of documents** with term \( i \)

**Intuition**

Postulate: Documents that are “close together” in vector space talk about the same things.
Cosine similarity

Cosine similarity of $D_j, D_k$:

$$\text{sim}(D_j, D_k) = \sum_{i=1}^{m} w_{ij} \times w_{ik}$$

Aka normalized inner product.

Clustering: a bit of terminology

- REPRESENTATIVE
- CENTROID
- OUTLIER
Key notion: *cluster representative*

- In the algorithms to follow, will generally need a notion of a representative point in a cluster
- Representative should be some sort of “typical” or central point in the cluster, e.g.,
  - point inducing smallest radii to docs in cluster
  - smallest squared distances, etc.
  - point that is the “average” of all docs in the cluster
- Need not be a document

Key notion: *cluster centroid*

- **Centroid** of a cluster = component-wise average of vectors in a cluster - is a vector.
  - Need not be a doc.
- Centroid of (1,2,3); (4,5,6); (7,2,6) is (4,3,5).
(Outliers in centroid computation)

- Can ignore outliers when computing centroid.
- What is an outlier?
  - Lots of statistical definitions, e.g.
    - moment of point to centroid > M \( \leq \) some cluster moment.
      
      Say 10.

Clustering algorithms

- Partitional vs. hierarchical
- Agglomerative
- K-means
Partitional Clustering

Original Points

A Partitional Clustering

Hierarchical Clustering

Traditional Hierarchical Clustering

Traditional Dendrogram

Non-traditional Hierarchical Clustering

Non-traditional Dendrogram
A partitional clustering algorithm: 
*K-MEANS*

• Given \( k \) - the number of clusters desired.
• Each cluster associated with a centroid.
• Each point assigned to the cluster with the closest centroid.
• Iterate.

---

**Basic iteration**

• At the start of the iteration, we have \( k \) centroids.
• Each doc assigned to the nearest centroid.
• All docs assigned to the same centroid are averaged to compute a new centroid;
  – thus have \( k \) new centroids.
Iteration example

- Docs
- Current centroids

Iteration example

- Docs
- New centroids
K-means Clustering: the full algorithm

1. Select $K$ points as the initial centroids.
2. repeat
3. Form $K$ clusters by assigning all points to the closest centroid.
4. Recompute the centroid of each cluster.
5. until The centroids don’t change

K-means Clustering – Details

- Initial centroids are often chosen randomly.
  - Clusters produced vary from one run to another.
- The centroid is (typically) the mean of the points in the cluster.
- ‘Closeness’ is measured by Euclidean distance, cosine similarity, correlation, etc.
- K-means will converge for common similarity measures mentioned above.
- Most of the convergence happens in the first few iterations.
  - Often the stopping condition is changed to ‘Until relatively few points change clusters’
- Complexity is $O( n \times K \times I \times d )$
  - $n =$ number of points, $K =$ number of clusters,
    $I =$ number of iterations, $d =$ number of attributes
Effect of initial choice of centroids: Two different K-means Clusterings

Importance of Choosing Initial Centroids
Termination conditions

• Several possibilities, e.g.,
  – A fixed number of iterations.
  – Doc partition unchanged.
  – Centroid positions don’t change.

Convergence

• Why should the \( k \)-means algorithm ever reach a fixed point?
  – A state in which clusters don’t change.

• \( k \)-means is a special case of a general procedure known as the EM algorithm.
  – Under reasonable conditions, known to converge.
  – Number of iterations could be large.
Exercise

• Consider running 2-means clustering on a corpus, each doc of which is from one of two different languages. What are the two clusters we would expect to see?
• Is agglomerative clustering likely to produce different results?

Evaluating K-means Clusters

• Most common measure is Sum of Squared Error (SSE)
  – For each point, the error is the distance to the nearest cluster
  – To get SSE, we square these errors and sum them.

\[
SSE = \sum_{i=1}^{K} \sum_{x \in C_i} \text{dist}^2(m_i, x)
\]

  – \( x \) is a data point in cluster \( C_i \) and \( m_i \) is the representative point for cluster \( C_i \)
  • can show that \( m \) corresponds to the center (mean) of the cluster
  – Given two clusters, we can choose the one with the smallest error
  – One easy way to reduce SSE is to increase \( K \), the number of clusters
    • A good clustering with smaller \( K \) can have a lower SSE than a poor clustering with higher \( K \)
Limitations of K-means

- K-means has problems when clusters are of differing
  - Sizes
  - Densities
  - Non-globular shapes

- K-means has problems when the data contains outliers.

Hierarchical clustering

- As clusters agglomerate, docs likely to fall into a hierarchy of “topics” or concepts.
Hierarchical Clustering

• Produces a set of nested clusters organized as a hierarchical tree
• Can be visualized as a dendrogram
  – A tree like diagram that records the sequences of merges or splits

Strengths of Hierarchical Clustering

• Do not have to assume any particular number of clusters
  – Any desired number of clusters can be obtained by ‘cutting’ the dendogram at the proper level

• They may correspond to meaningful taxonomies
  – Example in biological sciences (e.g., animal kingdom, phylogeny reconstruction, ...)
Hierarchical Clustering

- Two main types of hierarchical clustering
  - Agglomerative:
    - Start with the points as individual clusters
    - At each step, merge the closest pair of clusters until only one cluster (or k clusters) left
  - Divisive:
    - Start with one, all-inclusive cluster
    - At each step, split a cluster until each cluster contains a point (or there are k clusters)

- Traditional hierarchical algorithms use a similarity or distance matrix
  - Merge or split one cluster at a time

Agglomerative clustering

- Given target number of clusters $k$.
- Initially, each doc viewed as a cluster
  - start with $n$ clusters;
- Repeat:
  - while there are > $k$ clusters, find the “closest pair” of clusters and merge them.
“Closest pair” of clusters

- Many variants to defining closest pair of clusters
  - Clusters whose centroids are the most cosine-similar
  - ... whose “closest” points are the most cosine-similar
  - ... whose “furthest” points are the most cosine-similar

Example: $n=6$, $k=3$, closest pair of centroids
Issues

• Have to support finding closest pairs continually
  – compare all pairs?
  – Potentially $n^2$ cosine similarity computations
  – To avoid: use approximations.
  – “points” are changing as centroids change.
• Changes at each step are not localized
  – on a large corpus, memory management an issue
  – sometimes addressed by clustering a sample.

Hierarchical Agglomerative Clustering (HAC)

• More popular hierarchical clustering technique
• Assumes a similarity function for determining the similarity of two instances.
• Starts with all instances in a separate cluster and then repeatedly joins the two clusters that are most similar until there is only one cluster.
• The history of merging forms a binary tree or hierarchy.
Agglomerative Clustering Algorithm

• Basic algorithm is straightforward
  1. Compute the proximity matrix
  2. Let each data point be a cluster
  3. Repeat
     4. Merge the two closest clusters
     5. Update the proximity matrix
     6. Until only a single cluster remains

• Key operation is the computation of the proximity of two clusters
  – Different approaches to defining the distance between clusters distinguish the different algorithms

Starting Situation

• Start with clusters of individual points and a proximity matrix

```
p1  p2  p3  p4  p5  ...
p1  
p2  
p3  
p4  
p5  
      Proximity Matrix
```

```
p1  p2  p3  p4  p5  p6  p10  p11  p12
```

```
p1  
p2  
p3  
p4  
p6  
```
Intermediate Situation

• After some merging steps, we have some clusters

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Proximity Matrix

• We want to merge the two closest clusters (C2 and C5) and update the proximity matrix.

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Proximity Matrix
After Merging

• The question is “How do we update the proximity matrix?”

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>C2 U C5</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>C3</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>C4</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Proximity Matrix

How to Define Inter-Cluster Similarity

- MIN
- MAX
- Group Average
- Distance Between Centroids
- Other methods driven by an objective function
  - Ward's Method uses squared error
How to Define Inter-Cluster Similarity

- MIN
- MAX
- Group Average
- Distance Between Centroids
- Other methods driven by an objective function
  - Ward’s Method uses squared error

Proximity Matrix

- MIN
- MAX
- Group Average
- Distance Between Centroids
- Other methods driven by an objective function
  - Ward’s Method uses squared error
How to Define Inter-Cluster Similarity

- MIN
- MAX
- Group Average
- Distance Between Centroids
- Other methods driven by an objective function
  - Ward's Method uses squared error

Text clustering issues and applications
List of issues/applications covered

• Term vs. document space clustering
• Multi-lingual docs
• Feature selection
• Speeding up scoring
• Building navigation structures
  – “Automatic taxonomy induction”
• Labeling

Term vs. document space

• Thus far, we clustered docs based on their similarities in terms space
• For some applications, e.g., topic analysis for inducing navigation structures, can “dualize”:
  – use docs as axes
  – represent (some) terms as vectors
  – proximity based on co-occurrence of terms in docs
  – now clustering terms, not docs
Term vs. document space

- If terms carefully chosen (say nouns)
  - fixed number of pairs for distance computation
    - independent of corpus size
  - clusters have clean descriptions in terms of noun phrase co-occurrence - easier labeling?
  - left with problem of binding docs to these clusters

Multi-lingual docs

- E.g., News Explorer, Canadian government docs.
- Every doc in English and equivalent French.
  - Must cluster by concepts rather than language
- Simplest: pad docs in one lang with dictionary equivalents in the other
  - thus each doc has a representation in both languages
- Axes are terms in both languages
Feature selection

- Which terms to use as axes for vector space?
- Huge body of (ongoing) research
- IDF is a form of feature selection
  - can exaggerate noise e.g., mis-spellings
- Pseudo-linguistic heuristics, e.g.,
  - drop stop-words
  - stemming/lemmatization
  - use only nouns/noun phrases
- Good clustering should “figure out” some of these

Labelling clusters
Labelling clusters

• After clustering algorithm finds clusters - how can they be useful to the end user?
• Need pithy label for each cluster
  – In search results, say “Football” or “Car” in the *jaguar* example.
  – In topic trees, need navigational cues.
    • Often done by hand, a posteriori.

How to Label Clusters

• Show titles of typical documents
  – Titles are easy to scan
  – Authors create them for quick scanning!
  – But you can only show a few titles which may not fully represent cluster
• Show words/phrases prominent in cluster
  – More likely to fully represent cluster
  – Use distinguishing words/phrases
  – But harder to scan
Labeling

• Common heuristics - list 5-10 most frequent terms in the centroid vector.
  – Drop stop-words; stem.
• Differential labeling by frequent terms
  – Within the cluster “Computers”, child clusters all have the word computer as frequent terms.
  – Discriminant analysis of sub-tree centroids.

The biggest issues in clustering

• How do you compare two alternatives?
• Computation (time/space) is only one metric of performance
• How do you look at the “goodness” of the clustering produced by a method
READINGS

- Ingersoll / Morton / Farris – chapter 6
- Jain et al - Data Clustering: A Review (1999)
  - Available from the module site
  - Also: http://citeseer.nj.nec.com/jain99data.html

Acknowledgments

- Many of these slides borrowed, from
  - Chris Manning’s Stanford module
  - Tan, Steinbach and Kumar, Intro to Data Mining
  - Ray Mooney’s Austin module