Summary so far

• Supervised learning – learn to predict
  – Continuous target – regression; Categorical target – classification
• Linear Regression
• Classification
  – Discriminative models
    • Perceptron (linear)
    • Logistic regression (linear)
    • Decision tree (nonlinear)
    • Nearest Neighbor (nonlinear)
  – Generative models
    • Bayes classifier (not really practical)
    • Naïve bayes classifier (conditional independence between features given class label)
What to come

• There will be a few more topics we will cover on supervised learning
  – Ensemble methods
  – Neural nets (perhaps)
• But we will delay this for a couple of weeks
• We will talk about unsupervised learning such that you are not limited to supervised learning when thinking about your project
Unsupervised learning, Clustering

CS434
Unsupervised learning and pattern discovery

So far, our data has been in this form:

| \( x_1^1, x_2^1, x_3^1, \ldots, x_m^1 \) | \( y^1 \) |
| \( x_1^2, x_2^2, x_3^2, \ldots, x_m^2 \) | \( y^2 \) |
| \( \cdots \) | \( \cdots \) |
| \( x_1^n, x_2^n, x_3^n, \ldots, x_m^n \) | \( y^n \) |

We will be looking at **unlabeled data**:

| \( x_1^1, x_2^1, x_3^1, \ldots, x_m^1 \) |
| \( x_1^2, x_2^2, x_3^2, \ldots, x_m^2 \) |
| \( \cdots \) |
| \( x_1^n, x_2^n, x_3^n, \ldots, x_m^n \) |

What do we expect to learn from such data?

I have tons of data (web documents, gene data, etc) and need to:

- organize it better – e.g., find subgroups
- understand it better – e.g., understand interrelationships
- find regular trends in it – e.g., If A and B, then C
Hierarchical clustering as a way to organize web pages
Finding association patterns in data …
Dimension reduction
clustering
A hypothetical clustering example

Each point represents a person in the database
A hypothetical clustering example

This suggests there maybe three distinct spending patterns among the group of people in our database.
What is Clustering

• In general clustering can be viewed as an exploratory procedure for finding interesting subgroups in given data

• A large portion of the efforts focus on a special kind of clustering:
  – Group all given examples (i.e., exhaustive) into disjoint clusters (i.e., partitional), such that:
    • Examples within a cluster are (very) similar
    • Examples in different clusters are (very) different
Some example applications

- Information retrieval: cluster retrieved documents to present more organized and understandable results
- Consumer market analysis: cluster consumers into different interest groups so that marketing plans can be specifically designed for each individual group
- Image segmentation: decompose an image into regions with coherent color and texture
- Vector quantization for data (i.e., image) compression: group vectors into similar groups, and use group mean to represent group members
- Computational biology: group gene into co-expressed families based on their expression profile using different tissue samples and different experimental conditions
Image compression: Vector quantization

Group all pixels into self-similar groups, instead of storing all pixel values, store the means of each group

701,554 bytes → 127,292 bytes
Important components in clustering

• Distance/similarity measure
  – How to measure the similarity between two objects?

• Clustering algorithm
  – How to find the clusters based on the distance/similarity measures

• Evaluation of results
  – How do we know that our clustering result is good
Distance/similarity Measures

• One of the most important question in unsupervised learning, often more important than the choice of clustering algorithms

• What is similarity?
  – Similarity is hard to define, but we know it when we see it
  – More pragmatically, there are many mathematical definitions of distance/similarity
Common distance/similarity measures

- **Euclidean distance**
  \[ L_2(x, x')^2 = \sum_{i=1}^{d} (x_i - x'_i)^2 \]
  
  Straight line distance \( \geq 0 \)

- **Manhattan distance**
  \[ L_1(x, x') = \sum_{i=1}^{d} |x_i - x'_i| \]
  
  City block distance \( \geq 0 \)

- **Cosine similarity**
  \[ \cos(x, x') = \frac{\langle x \cdot x' \rangle}{|x| \cdot |x'|} \]
  
  Angle between two vectors, commonly used for measuring document similarity

- **More flexible measures:**
  \[ D(x, x') = \sqrt{\sum_{i=1}^{d} w_i (x_i - x'_i)^2} \]
  
  Scale each feature differently using \( w_i \)'s

  one can learn the appropriate weights given user guidance

Note: We can always transform between distance and similarity using a monotonically decreasing function, for example \( e^{-\alpha D(x_1, x_2)^2} \)
How to decide which to use?

• It is application dependent
• Consider your application domain, you need to ask questions such as:
  – What does it mean for two consumers to be similar to each other? Or for two genes to be similar to each other?
• For example, for text domain, we typically use cosine similarity
• This may or may not give you the answer you want depends on your existing knowledge of the domain
  – When domain knowledge is not enough, one could consider learning based approach
A learning approach

• Ideally we’d like to learn a distance function from user’s inputs
  – Ask users to provide things like object A is similar to object B, dissimilar to object C
  – For example, if a user want to group the marbles based on their pattern
  – Learn a distance function to correctly reflect these relationships
    • E.g. weigh pattern-related features more than color features
  – This is a more advanced topic that we will not cover in this class, but nonetheless very important
• When we can not afford to learn a distance measure, and don’t have a clue about which distance function is appropriate, what should we do?

• Clustering is an exploratory procedure and it is important to explore – i.e., try different options
Clustering

• Now we have seen different ways that one can use to measure distances or similarities among a set of unlabeled objects, how can we use them to group the objects into clusters?

• People have looked at many different approaches and they can be categorized into two distinct types
Hierarchical and non-hierarchical Clustering

- **Hierarchical clustering** builds a tree-based hierarchical taxonomy (*dendrogram*)

  - animal
    - vertebrate
      - fish
      - reptile
      - …
      - mammal
    - invertebrate
      - worm
      - …
      - insect

- **A non-hierarchical (flat) clustering** produces a single partition of the unlabeled data
  - Generates a single partition of the data without resorting to the hierarchical procedure
  - Representative example for flat clustering: K-means clustering
Goal of Clustering

• Given a data set $D = \{x_1, x_2, ..., x_n\}$
• We need to partition $D$ into $k$ disjoint clusters, such that
  – Examples are self-similar in the same cluster
• How do we quantify this?
Objective: Sum of Squared Errors

- Given a partition of the data into $k$ clusters, we can compute the center (i.e., mean, center of mass) of each cluster

$$\mu_i = \frac{1}{n_i} \sum_{x \in C_i} x$$

- For a well formed cluster, its points should be close to its center. We measure this with sum of squared error (SSE):

$$J_e = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

- Our objective is thus to find a partition $C^*$:

$$C^* = \arg\min_{C=\{C_1, \ldots, C_k\}} \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

- This is a combinatorial optimization problem
  - NP-hard
  - Kmeans finds a local optimal solution using an iterative approach
Basic idea

- We assume that the number of desired clusters, $k$, is given.
- Randomly choose $k$ examples as *seeds*, one per cluster.
- Form initial clusters based on these seeds.
- Iterate by repeatedly reallocating instances to different clusters to improve the overall clustering.
- Stop when clustering converges or after a fixed number of iterations.
The Standard K-means algorithm

**Input:** $D = \{x_1, x_2, \ldots, x_n\}$ and desired number of clusters $k$

**Output:** $c_1, \ldots, c_k$ such that $D = c_1 \cup c_2 \cup \cdots \cup c_k$

Let $d$ be the distance function between examples

1. Select $k$ random samples from $D$ as centers $\{\mu_1, \ldots, \mu_k\}$ // **Initialization**
2. Do
3. for each example $x_i$,
4. assign $x_i$ to $c_j$ such that $d(\mu_j, x_i)$ is minimized // **the Assignment step**
5. for each cluster $j$, update its cluster center
6. $\mu_j = \frac{1}{|c_j|} \sum_{x \in c_j} x$ // **the update step**
7. Until convergence
K-Means Example (K=2)
K-Means Example (K=2)

Pick seeds
K-Means Example (K=2)

Pick seeds
Reassign clusters
K-Means Example (K=2)

Pick seeds
Reassign clusters
Compute centroids
K-Means Example (K=2)

- Pick seeds
- Reassign clusters
- Compute centroids
- Reassign clusters
K-Means Example (K=2)

- Pick seeds
- Reassign clusters
- Compute centroids
- Reassign clusters
- Compute centroids
K-Means Example (K=2)

Pick seeds
Reassign clusters
Compute centroids
Reassign clusters
Compute centroids
Reassign clusters
K-Means Example (K=2)

Pick seeds
Reassign clusters
Compute centroids
Reassign clusters
Compute centroids
Reassign clusters
Converged!
Monotonicity of K-means

- **Monotonicity Property:** Each iteration of K-means strictly decreases the SSE until convergence.

- The following lemma is key to the proof:
  - **Lemma:** Given a finite set $C$ of data points, the value of $\mu$ that minimizes the SSE:

    $$ J = \sum_{x \in C} ||x - \mu||^2 $$

    is:
    $$ \mu = \frac{1}{|C|} \sum_{x \in C} x $$
Proof of monotonicity

• Given a current set of clusters with their means, the SSE is given by:
  \[ J_e = \sum_{i=1}^{k} \sum_{x \in c_i} \|x - \mu_i\|^2 \]

• Consider the assignment step:
  – Since each point is only reassigned if it is closer to some other cluster than its current cluster, so we know the reassignment step will only decrease SSE

• Consider the re-center step:
  – From our lemma we know the updated \( \mu_i \) minimizes the SSE of \( c_i \), which implies that the resulting SSE again is decreased.

• Combine the above two, we know K-means always decreases the SSE
K-means properties

- K-means always converges in a finite number of steps
  - Typically converges very fast (in fewer iterations than $n = |D|$)
- Time complexity:
  - Assume computing distance between two instances is $O(d)$ where $d$ is the dimensionality of the vectors.
  - Reassigning clusters: $O(kn)$ distance computations, or $O(knd)$.
  - Computing centers: Each instance vector gets added once to some center: $O(nd)$.
  - Assume these two steps are each done once for $l$ iterations: $O(lknd)$.
  - Linear in all relevant factors, assuming a fixed number of iterations
More Comments

- Highly sensitive to the initial seeds

- This is because SSE has many local minimal solutions, i.e., solutions that can not be improved by local reassignments of any particular points.
Solutions

• Run multiple trials and choose the one with the best SSE
  – This is typically done in practice
• Heuristics: try to choose initial centers to be far apart
  – Using furthest-first traversal
    • Start with a random initial center
    • Set the second center to be furthest from the first center
    • The third center to be furthest from the first two centers
    • ...

Even more comments

• K-Means is exhaustive:
  – Cluster every data point, no notion of outlier

• Outliers may cause problems, why?
K-medoids

• Outliers strongly impact the cluster centers
• K-medoids – use the medoid for each cluster, i.e., the data point that is closest to other points in the cluster

\[
\mu = \frac{1}{|C|} \sum_{x \in C} x
\]

\[
\mu = \arg\min_{x \in C} \sum_{z \in C} \|x - z\|^2
\]

• K-medoids is computationally more expensive but more robust to outliers
Deciding \( k \) – a model selection problem

- What if we don’t know how many clusters there are in the data?
- Can we use SSE to decide \( k \) by choosing \( k \) that gives the smallest SSE?
  - We will always favor larger \( k \) values
- Any quick solutions?
  - Find the knee(s)