Week 1: Foundation + Clustering

ML/AI Design Thinking: Empathize Phase

BSc Data Science & Design

2025

The Convergence Flow

How Random Data Points Become Meaningful Groups

[Placeholder for Convergence Flow visualization]

Watch as 1000 users organize themselves into natural groups

Welcome to ML + Design Thinking

Traditional Design Thinking

- Empathize with users
- Define problems
- Ideate solutions
- Prototype ideas
- Test and iterate

+ Machine Learning Power

- Analyze thousands of users
- Find hidden patterns
- Generate insights automatically
- Validate with data
- Scale your understanding

This Week: Using clustering to truly understand your users

The Empathize Phase: Understanding Your Users

What is Empathizing?

- Walking in your users' shoes
- Understanding their needs, wants, fears
- Discovering what they don't tell you
- Finding patterns in behavior

Traditional Methods:

- Interviews (5-20 people)
- Observations (days/weeks)
- Surveys (100s of responses)

ML-Enhanced Methods:

- Analyze millions of interactions
- Find patterns humans miss
- Work 24/7 automatically
- Unbiased grouping

[Placeholder: User empathy visualization]

Why Clustering Helps Us Understand People

From Chaos to Clarity

Before Clustering

[Chaos visualization]

10,000 users = overwhelming

Clustering Process

[Process visualization]

ML finds natural groups

After Clustering

[Segments visualization]

5 clear user types!

- **Power Users**: Heavy usage, all features
- **Casuals**: Weekend usage, basic features
- **Professionals**: Business hours, productivity focus
- **Students**: Evening usage, collaboration features
- **Explorers**: Try everything once

What is Clustering?

Clustering = Finding Natural Groups in Data

Real-World Examples:

- Spotify: Grouping similar listeners
- Netflix: Finding viewer types
- Amazon: Customer segments
- Instagram: Content categories
- Gmail: Organizing emails

[Clustering examples]

Key Idea: Items in the same group are more similar to each other than to items in other groups

No labels needed - the algorithm finds groups automatically!

How Do We Measure "Similar"?

Distance = Difference Between Things

Simple Example: App Usage

- User A: 2 hours/day, 10 features used
- User B: 2.5 hours/day, 12 features used
- User C: 8 hours/day, 50 features used

Who is more similar?

- A and B are close (similar usage)
- C is far from both (power user)

Think of it like: "How different are these users?"

[Distance visualization]

Common Measures:

- Straight line (Euclidean)
- City blocks (Manhattan)
- Correlation-based

K-means: Finding Group Centers

Like Finding the Best Meeting Points for Groups

How K-means Works:

1. Pick K center points randomly
2. Assign each user to nearest center
3. Move centers to group middle
4. Repeat until stable

Real Example:

Finding 3 types of coffee drinkers:

- Morning rushers
- Afternoon socializers
- All-day workers

[K-means animation]

Pros: Fast, simple, scalable

Cons: Need to know K, assumes round clusters

Hierarchical: Building a Family Tree

Bottom-Up Approach:

1. Start: Everyone is separate
2. Find two most similar users
3. Group them together
4. Repeat with groups
5. Stop when all connected

Like Making Friends:

- Best friends first
- Then friend groups
- Then communities
- Finally, everyone connected

Dendrogram: The Family Tree

[Dendrogram example]

Cut the tree at any height to get different numbers of groups!

How Do We Know Our Groups Are Good?

Three Simple Checks

1. Tight Groups	2. Separated Groups	3. Makes Sense
[Tight clusters]	*[Separated clusters]*	*[Meaningful clusters]*
Users in same group should be close together	Different groups should be far apart	Groups should mean something real

Simple Metrics:

- **Elbow Method**: Plot error vs. number of clusters, look for "elbow"
- **Silhouette Score**: -1 (bad) to +1 (perfect), aim for ¿ 0.5
- **Business Sense**: Can you name and use each group?

Preparing Your Data

Getting Data Ready for Clustering

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load your user data
users = pd.read_csv('user_behavior.csv')

# Select features for clustering
features = ['daily_usage_hours', 'features_used',
            'days_active', 'messages_sent']

# Handle missing values
users[features] = users[features].fillna(users[features].mean())

# Normalize: Make all features same scale (0-1)
scaler = StandardScaler()
users_normalized = scaler.fit_transform(users[features])

print("Before:", users[features].iloc[0].values)
# [8.5, 45, 28, 156]
print("After:", users_normalized[0])
# [1.2, 0.8, 1.1, 0.9] - all similar scale!
```

Why normalize? So "hours used" doesn't dominate "features used"

Your First K-means in Python

Just 5 Lines to Find User Groups!

```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Create and fit K-means (let's find 4 groups)
kmeans = KMeans(n_clusters=4, random_state=42)
users['cluster'] = kmeans.fit_predict(users_normalized)

# See the groups
print(users.groupby('cluster')[features].mean())
```

```
Cluster 0: Power Users
  - 8.2 hours/day
  - 52 features used

Cluster 1: Casual Users
  - 1.5 hours/day
  - 8 features used

Cluster 2: Regular Users
  - 4.1 hours/day
  - 25 features used

Cluster 3: New Users
  - 0.8 hours/day
  - 3 features used
```

That's it! You've segmented thousands of users in seconds

How Many Groups? The Elbow Method

Finding the "Just Right" Number of Clusters

```python
# Try different numbers of clusters
inertias = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(users_normalized)
    inertias.append(kmeans.inertia_)

# Plot the elbow curve
plt.plot(K_range, inertias, 'bo-')
plt.xlabel('Number of Clusters')
plt.ylabel('Total Distance')
plt.title('The Elbow Method')
plt.show()
```

Look for the "elbow" - where adding more clusters doesn't help much

[Elbow method chart]

In this example:

- 2-3 clusters: Big improvement
- 4-5 clusters: Good improvement
- 6+ clusters: Diminishing returns
- **Choose: 4 or 5 clusters**

Creating Dendrograms (Tree Diagrams)

Visualizing How Users Group Together

```python
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Create the linkage matrix
linkage_matrix = linkage(users_normalized,
                         method='ward')

# Plot dendrogram
plt.figure(figsize=(10, 6))
dendrogram(linkage_matrix,
           labels=users['user_id'].values,
           leaf_rotation=90)
plt.title('User Clustering Dendrogram')
plt.xlabel('User ID')
plt.ylabel('Distance')
plt.show()

# Cut tree to get 4 clusters
from scipy.cluster.hierarchy import fcluster
users['h_cluster'] = fcluster(linkage_matrix,
                              t=4,
                              criterion='maxclust')
```

[Dendrogram with cut line]

Reading the Tree:

- Bottom: Individual users
- Height: How different groups are
- Branches: Groups forming
- Cut line: Your chosen clusters

Finding Dense Areas with DBSCAN

When Your Groups Aren't Round

```python
from sklearn.cluster import DBSCAN

# DBSCAN: Finds dense regions
dbscan = DBSCAN(eps=0.5,       # neighborhood size
                min_samples=5)  # min points
users['db_cluster'] = dbscan.fit_predict(
    users_normalized)

# Check results
print(f"Found {len(set(users['db_cluster']))-1} clusters")
print(f"Outliers: {sum(users['db_cluster']==-1)}")

# Visualize
colors = ['red','blue','green','yellow','purple']
for i in range(max(users['db_cluster'])+1):
    if i == -1:  # Outliers
        plt.scatter(X[users['db_cluster']==i, 0],
                    X[users['db_cluster']==i, 1],
                    c='gray', marker='x', alpha=0.3)
    else:
        plt.scatter(X[users['db_cluster']==i, 0],
                    X[users['db_cluster']==i, 1],
                    c=colors[i], alpha=0.6)
```

DBSCAN Advantages:

[DBSCAN shape examples]

- Finds any shape clusters
- Identifies outliers (noise)
- No need to specify K
- Great for unusual patterns

Choosing the Right Features

What to Measure for Good Clustering
Good Features for User Clustering:

- Usage frequency
- Feature adoption
- Time patterns
- Interaction types
- Content preferences

Avoid These:

- User ID (unique)
- Registration date (if not relevant)
- Random identifiers
- Highly correlated features

Pro Tip: Start with 3-5 strong features, add more if needed

Feature Engineering Example:

Raw Data	Engineered Feature
Login times	Morning/Evening user
Click events	Clicks per session
Page views	Depth of exploration
Purchase history	Spending tier
Support tickets	Frustration level

[Feature importance chart]

Making Clusters Visible (2D Plots)

Reducing Dimensions to See Patterns

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Reduce to 2D for visualization
pca = PCA(n_components=2)
users_2d = pca.fit_transform(users_normalized)

# Plot with cluster colors
plt.figure(figsize=(10, 8))
colors = ['#e74c3c', '#3498db', '#2ecc71', '#f39c12']
for i in range(4):
    cluster_data = users_2d[users['cluster'] == i]
    plt.scatter(cluster_data[:, 0],
                cluster_data[:, 1],
                c=colors[i],
                label=f'Cluster {i}',
                alpha=0.6, s=50)

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('User Clusters Visualization')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

[PCA visualization]

What PCA Does:

- Combines all features into 2D
- Preserves most important differences
- Makes patterns visible
- Great for presentations!

Working with Categories

Clustering with Non-Numeric Data

```python
# Convert categories to numbers
from sklearn.preprocessing import LabelEncoder

# Example: Device type
le = LabelEncoder()
users['device_encoded'] = le.fit_transform(
    users['device_type'])
# 'mobile' -> 0, 'desktop' -> 1, 'tablet' -> 2

# Better: One-hot encoding for clustering
device_dummies = pd.get_dummies(
    users['device_type'],
    prefix='device')
# Creates: device_mobile, device_desktop, device_tablet

# Combine with numeric features
features_all = pd.concat([
    users[numeric_features],
    device_dummies,
    pd.get_dummies(users['subscription_type'])
], axis=1)

# Now cluster as normal
kmeans = KMeans(n_clusters=4)
users['cluster'] = kmeans.fit_predict(features_all)
```

Category Examples:

- Device type
- Subscription level
- Country/Region
- Product categories
- User role

Rule: Use one-hot encoding for clustering, not label encoding

Real Example: E-commerce Customer Segmentation

Finding Customer Types in Online Shopping Data

```python
# Real e-commerce features
customer_features = [
    'total_spent', 'order_frequency', 'avg_cart_size',
    'categories_browsed', 'return_rate', 'review_count'
]

# Cluster and analyze
kmeans = KMeans(n_clusters=5)
customers['segment'] = kmeans.fit_predict(customers_scaled)

# Results interpretation
for i in range(5):
    segment = customers[customers['segment'] == i]
    print(f"\nSegment {i}: {len(segment)} customers")
    print(segment[customer_features].mean())
```

Discovered Segments:

- **VIP Shoppers**: High spend, low returns
- **Bargain Hunters**: Sale-focused, high cart
- **Window Shoppers**: Browse, rarely buy
- **Loyal Regulars**: Consistent, medium spend
- **One-timers**: Single purchase, dormant

[Customer segments chart]

Interactive Demo: Try It Yourself!

Complete Clustering Pipeline

```python
# Complete working example you can run
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Generate sample data (replace with your data)
np.random.seed(42)
n_users = 1000
data = {
    'usage_hours': np.random.exponential(3, n_users),
    'features_used': np.random.poisson(15, n_users),
    'days_active': np.random.randint(1, 31, n_users)
}
df = pd.DataFrame(data)

# Standardize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df)

# Cluster
kmeans = KMeans(n_clusters=3, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled)

# Visualize and interpret
print(df.groupby('cluster').mean())
# Try changing n_clusters and see what happens!
```

From Data Clusters to User Personas

Turning Numbers into People

Cluster Statistics:

Metric	Cluster 0
Avg. Usage	7.2 hrs/day
Features Used	45/50
Peak Time	9am-5pm
Device	Desktop (85%)
Retention	95%

Persona Created:

"Professional Paula"
Power user, 32, Marketing Manager

Goals: Maximize productivity
Needs: Advanced features, shortcuts
Pain: Slow load times
Quote: "This tool is my office"

Transformation Process:

1. Analyze cluster statistics
2. Identify defining characteristics
3. Create realistic profile
4. Add human elements (name, photo, quote)
5. Validate with real user interviews

Creating Empathy Maps from Data

What Your Clusters Think, Feel, Say, and Do

[Empathy map visualization]

THINK
From search queries and help topics

FEEL
From sentiment in reviews and feedback

SAY
From support tickets and forums

DO
From clickstream and usage data

Each cluster fills the empathy map differently based on their data patterns

Finding User Pain Points

Where Each Cluster Struggles
Pain Point Detection Methods:

- High exit rates at specific features
- Support ticket clustering
- Feature abandonment patterns
- Error message frequency
- Negative sentiment spikes

Cluster-Specific Pain Points:

- **Power Users**: Need bulk operations
- **New Users**: Overwhelming interface
- **Mobile Users**: Desktop-only features
- **Free Users**: Paywall friction

Different clusters = Different problems = Different solutions needed

[Pain points heatmap]

Red = High frustration areas per cluster

Discovering Behavior Patterns

How Different Clusters Use Your Product

[Behavior patterns chart]

Morning Rushers

- 6am-9am peak
- Quick actions
- Mobile-heavy
- Notifications on

Deep Workers

- 2-4 hour sessions
- Complex workflows
- Desktop only
- Focus mode users

Social Butterflies

- Share frequently
- Collaboration tools
- Comments active
- Team features

Design different experiences for different behavior patterns

Building User Journey Maps

Each Cluster's Path Through Your Product
[Journey map visualization]
Cluster-Based Journey Insights:

- **Discoverers**: Long exploration phase
- **Goal-Oriented**: Direct to action
- **Learners**: Heavy documentation use
- **Social**: Share early and often
- **Cautious**: Multiple trial sessions
- **Power**: Skip onboarding

Optimize each touchpoint for each cluster's journey style

Identifying Stakeholder Groups

Who Really Uses Your Product?
Discovered Stakeholders:

- **Decision Makers** (5%)
  - Admin features
  - Billing pages
  - Team management
- **Daily Users** (60%)
  - Core features
  - Regular patterns
  - Productivity focus
- **Influencers** (15%)
  - Share features
  - Invite others
  - Write reviews
- **Evaluators** (20%)
  - Trial users
  - Comparison shoppers
  - Feature testers

[Stakeholder network diagram]

Network shows how different groups interact and influence each other

Visual Persona Cards

Making Clusters Memorable and Actionable

[Persona cards visualization]

Each Card Includes:

- Photo/Avatar
- Name & Role
- Key Stats
- Goals
- Frustrations
- Needs
- Tech savviness
- Usage patterns
- Feature preferences
- Quote
- Design implications
- Priority level

Print and post these cards - keep users visible during design!

From Insights to Action

Cluster-Driven Decisions:

Finding	Design Decision
3 skill levels found	Progressive disclosure UI
Mobile vs Desktop split	Responsive-first design
Power users frustrated	Advanced mode option
New users confused	Better onboarding
Social cluster exists	Add sharing features

Prioritization Framework:

1. Size of cluster (impact)
2. Pain intensity (urgency)
3. Business value (ROI)
4. Implementation cost (feasibility)

Design for your biggest, most valuable, or most struggling clusters first

[Priority matrix chart]

Plot features by cluster importance vs. effort

Other Clustering Methods Overview

Beyond K-means: More Tools in Your Toolkit

Gaussian Mixture Models

- Soft clustering (probability-based)
- Handles overlapping groups
- Good for uncertain boundaries

Mean Shift

- Finds density peaks automatically
- No need to specify K
- Great for image segmentation

Spectral Clustering

- Handles complex shapes
- Uses graph theory
- Good for social networks

OPTICS

- Like DBSCAN but better
- Handles varying densities
- Creates reachability plots

[Methods comparison chart]

How Clustering Powers 75 Million Personalized Playlists

The Challenge:

- 406 million users
- 82 million songs
- Create unique playlists weekly
- Feel personally curated

The Solution:

1. Cluster users by listening history
2. Cluster songs by audio features
3. Find songs your cluster likes that you haven't heard
4. Mix in variety from adjacent clusters
5. Result: 30 new songs every Monday

Clustering at scale: From understanding users to delighting them

[Spotify clustering diagram]

Impact:

- 40% of users engage weekly
- 60% save at least one song
- Billions of streams generated

Week 1: Key Takeaways

What You've Learned

Technical Skills:

- K-means clustering implementation
- Hierarchical clustering with dendrograms
- DBSCAN for density-based groups
- Data preparation and scaling
- Cluster evaluation methods
- Visualization techniques

Design Skills:

- Creating data-driven personas
- Building empathy maps
- Identifying pain points
- Journey mapping
- Stakeholder identification

Key Insights:

- Clustering reveals hidden user groups
- Different algorithms for different data
- Always validate with business sense
- Clusters drive design decisions
- Scale empathy with data

Remember: Clustering is about understanding, not just grouping

Preview: Week 2

Advanced Clustering + Deep Empathy

Next Week You'll Learn:

- Time-series clustering for behavior evolution
- Multi-view clustering (combining data sources)
- Online clustering for real-time segmentation
- Clustering validation techniques
- A/B testing with clusters
- Emotional journey mapping
- Micro-moment identification
- Cluster-based personalization

Practical Project:

Build a complete user segmentation system for a real app

[Week 2 preview visualization]

Homework:

- Practice K-means on your data
- Create one persona from a cluster
- Read: Chapter 2 materials

See you next week for deeper dives into clustering!

Distance Metrics Formulas
Euclidean Distance:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Manhattan Distance:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Cosine Similarity:

$$\text{similarity}(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}}$$

Minkowski Distance:

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

Complete Algorithm Specification
Objective Function (Minimize):

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

where μ_i is the mean of cluster C_i
Algorithm Steps:

1. Initialize: Choose k points as initial centroids
2. Assignment: $C_i = \{x_p : ||x_p - \mu_i||^2 \leq ||x_p - \mu_j||^2 \forall j, 1 \leq j \leq k\}$
3. Update: $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$
4. Repeat until convergence: $||\mu_i^{(t+1)} - \mu_i^{(t)}|| < \epsilon$

Complexity: $O(n \cdot k \cdot d \cdot i)$ where n = points, k = clusters, d = dimensions, i = iterations

Appendix: Silhouette Coefficient Calculation

Cluster Quality Metric

For each point i:

- $a(i)$ = average distance to points in same cluster
- $b(i)$ = minimum average distance to points in different cluster

Silhouette Coefficient:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Interpretation:

- $s(i) \approx 1$: Well clustered
- $s(i) \approx 0$: On border between clusters
- $s(i) \approx -1$: Misclassified

Overall Score:

$$S = \frac{1}{n} \sum_{i=1}^{n} s(i)$$

Appendix: Gaussian Mixture Models

Probabilistic Clustering

Model:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

where:

- π_k = mixing coefficient (prior probability)
- μ_k = mean of component k
- Σ_k = covariance matrix of component k

EM Algorithm:

- **E-step:** Compute responsibilities

$$\gamma_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$

- **M-step:** Update parameters

$$\mu_k = \frac{\sum_n \gamma_{nk} x_n}{\sum_n \gamma_{nk}}$$

Appendix: t-SNE Mathematical Background

t-Distributed Stochastic Neighbor Embedding
High-dimensional similarity:

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

Low-dimensional similarity (Student t-distribution):

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

Objective (KL divergence):

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Gradient:

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}$$

Appendix: Computational Complexity Analysis

Time and Space Complexity of Clustering Algorithms

Algorithm	Time Complexity	Space Complexity
K-means	$O(n \cdot k \cdot d \cdot i)$	$O(n \cdot d + k \cdot d)$
Hierarchical	$O(n^2 \log n)$ to $O(n^3)$	$O(n^2)$
DBSCAN	$O(n \log n)$ average	$O(n)$
GMM	$O(n \cdot k \cdot d^2 \cdot i)$	$O(n \cdot d + k \cdot d^2)$
Spectral	$O(n^3)$	$O(n^2)$
Mean Shift	$O(n^2 \cdot i)$	$O(n \cdot d)$

Legend:

- n = number of data points
- k = number of clusters
- d = dimensionality
- i = number of iterations

Scalability Tips:

- Use Mini-batch K-means for $n > 10,000$
- Consider sampling for hierarchical clustering
- Use approximate nearest neighbors for DBSCAN

Appendix: Further Reading

Key Papers and Resources
Foundational Papers:

- MacQueen, J. (1967). "Some methods for classification and analysis of multivariate observations"
- Ester et al. (1996). "A density-based algorithm for discovering clusters" (DBSCAN)
- Ng et al. (2002). "On spectral clustering: Analysis and an algorithm"

Modern Applications:

- Sculley, D. (2010). "Web-scale k-means clustering" (Google)
- McInnes et al. (2017). "hdbscan: Hierarchical density based clustering"
- Spotify Research. "Understanding Music through Machine Learning"

Online Resources:

- scikit-learn clustering documentation
- Google's Machine Learning Crash Course
- Fast.ai Practical Deep Learning course