Unsupervised Learning - Intermediate Handout

Machine Learning for Smarter Innovation

1 Unsupervised Learning - Intermediate Handout

Target Audience: Practitioners with basic Python knowledge **Duration**: 45 minutes reading + coding **Level**: Intermediate (includes code examples)

1.1 Setup

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.metrics import silhouette_score, calinski_harabasz_score
```

1.2 1. K-Means Clustering

1.2.1 Basic Implementation

```
# Load and scale data
X = df[['feature1', 'feature2', 'feature3']]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Fit K-means
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_scaled)

# Add to dataframe
df['cluster'] = clusters
print(f"Cluster centers:\n{kmeans.cluster_centers_}")
```

1.2.2 Finding Optimal K (Elbow Method)

```
inertias = []
```

```
K_range = range(2, 11)

for k in K_range:
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    km.fit(X_scaled)
    inertias.append(km.inertia_)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(K_range, inertias, 'bo-')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia (Within-cluster SS)')
plt.title('Elbow Method')

# Silhouette scores
silhouettes = []
for k in K_range:
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = km.fit_predict(X_scaled)
    silhouettes.append(silhouette_score(X_scaled, labels))

plt.subplot(1, 2, 2)
plt.plot(K_range, silhouettes, 'ro-')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Analysis')
plt.tight_layout()
plt.show()
```

1.3 2. DBSCAN (Density-Based)

1.3.1 When to Use DBSCAN

- Unknown number of clusters
- Clusters have irregular shapes
- Need to identify outliers/noise
- Data has varying densities

1.3.2 Implementation

```
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors

# Find optimal eps using k-distance graph
neighbors = NearestNeighbors(n_neighbors=5)
neighbors.fit(X_scaled)
distances, _ = neighbors.kneighbors(X_scaled)
distances = np.sort(distances[:, -1])

plt.figure(figsize=(8, 4))
plt.plot(distances)
plt.ylabel('5-NN Distance')
plt.xlabel('Points (sorted)')
plt.title('K-Distance Graph - Look for elbow')
plt.show()

# Apply DBSCAN
```

```
dbscan = DBSCAN(eps=0.5, min_samples=5)
clusters = dbscan.fit_predict(X_scaled)

# Analyze results
n_clusters = len(set(clusters)) - (1 if -1 in clusters else 0)
n_noise = list(clusters).count(-1)
print(f"Clusters found: {n_clusters}")
print(f"Noise points: {n_noise} ({n_noise/len(clusters):.1%})")
```

1.3.3 Parameter Grid Search

```
eps_values = [0.3, 0.5, 0.7, 1.0]
min_samples_values = [3, 5, 10]

results = []
for eps in eps_values:
    for min_samples in min_samples_values:
        db = DBSCAN(eps=eps, min_samples=min_samples)
        labels = db.fit_predict(X_scaled)
        n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
        n_noise = list(labels).count(-1)
        if n_clusters > 1:
            sil = silhouette_score(X_scaled[labels != -1], labels[labels !=
    -1])
        else:
            sil = -1
        results.append({'eps': eps, 'min_samples': min_samples,
                        'n_clusters': n_clusters, 'noise': n_noise, 'silhouette
    ': sil})

pd.DataFrame(results).sort_values('silhouette', ascending=False)
```

1.4 3. Hierarchical Clustering

1.4.1 Dendrogram Analysis

```
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

# Create linkage matrix
linked = linkage(X_scaled, method='ward')

# Plot dendrogram
plt.figure(figsize=(12, 6))
dendrogram(linked, truncate_mode='lastp', p=30,
           leaf_rotation=90, leaf_font_size=10)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index or Cluster Size')
plt.ylabel('Distance (Ward)')
plt.axhline(y=15, color='r', linestyle='--', label='Cut threshold')
plt.legend()
plt.show()

# Cut tree at threshold
clusters = fcluster(linked, t=15, criterion='distance')
# Or cut to get specific number of clusters
clusters = fcluster(linked, t=4, criterion='maxclust')
```

1.4.2 Linkage Methods Comparison

Method	Formula	Best For
Ward	Minimize variance increase	Compact, equal-sized
Complete	Max pairwise distance	Outlier-sensitive
Average	Mean pairwise distance	Balanced
Single	Min pairwise distance	Elongated shapes

1.5 4. Dimensionality Reduction

1.5.1 PCA for Feature Reduction

```
from sklearn.decomposition import PCA

# Determine optimal components
pca_full = PCA()
pca_full.fit(X_scaled)

# Cumulative variance plot
cum_var = np.cumsum(pca_full.explained_variance_ratio_)
plt.figure(figsize=(8, 4))
plt.plot(range(1, len(cum_var)+1), cum_var, 'bo-')
plt.axhline(y=0.95, color='r', linestyle='--', label='95% threshold')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('PCA - Variance Explained')
plt.legend()
plt.show()

# Apply PCA (keep 95% variance)
n_components = np.argmax(cum_var >= 0.95) + 1
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_scaled)
print(f"Reduced from {X_scaled.shape[1]} to {n_components} components")
```

1.5.2 t-SNE for Visualization

```
from sklearn.manifold import TSNE

# t-SNE is for visualization only - never cluster on t-SNE output!
tsne = TSNE(n_components=2, perplexity=30, random_state=42, n_iter=1000)
X_tsne = tsne.fit_transform(X_scaled)

plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=clusters,
                      cmap='viridis', alpha=0.7)
plt.colorbar(scatter, label='Cluster')
plt.title('t-SNE Visualization of Clusters')
plt.xlabel('t-SNE 1')
plt.ylabel('t-SNE 2')
plt.show()
```

1.6 5. Cluster Profiling

1.6.1 Statistical Summary

```
# Add cluster labels to data
df['cluster'] = clusters

# Profile each cluster
profile = df.groupby('cluster').agg({
    'feature1': ['mean', 'std', 'min', 'max'],
    'feature2': ['mean', 'std', 'min', 'max'],
    'feature3': ['mean', 'std', 'min', 'max']
}).round(2)

# Cluster sizes
sizes = df['cluster'].value_counts().sort_index()
print("Cluster sizes:")
print(sizes)
print(f"\nCluster profiles:\n{profile}")
```

1.6.2 Visualization

```
# Parallel coordinates plot
from pandas.plotting import parallel_coordinates

# Normalize for visualization
df_norm = df.copy()
for col in features:
    df_norm[col] = (df[col] - df[col].mean()) / df[col].std()

plt.figure(figsize=(12, 6))
parallel_coordinates(df_norm[features + ['cluster']], 'cluster',
                     colormap='viridis', alpha=0.5)
plt.title('Cluster Profiles - Parallel Coordinates')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

1.7 6. Validation Metrics

```
from sklearn.metrics import (silhouette_score, calinski_harabasz_score,
                             davies_bouldin_score)

# Internal validation (no ground truth needed)
print("Cluster Quality Metrics:")
print(f"  Silhouette Score: {silhouette_score(X_scaled, clusters):.3f}")
print(f"  Calinski-Harabasz: {calinski_harabasz_score(X_scaled, clusters):.1f}
    ")
print(f"  Davies-Bouldin: {davies_bouldin_score(X_scaled, clusters):.3f}")
```

Metric	Range	Interpretation
Silhouette	-1 to 1	Higher = better separation
Calinski-Harabasz	0 to inf	Higher = denser, well-separated

Metric	Range	Interpretation
Davies-Bouldin	0 to inf	Lower = better

1.8 Complete Workflow Template

```
def unsupervised_pipeline(df, features, method='kmeans', n_clusters=None):
    """
    Complete unsupervised learning pipeline.

    Parameters:
    - df: DataFrame with data
    - features: list of column names to use
    - method: 'kmeans', 'dbscan', or 'hierarchical'
    - n_clusters: number of clusters (not needed for DBSCAN)
    """
    # 1. Prepare and scale
    X = df[features].copy()
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # 2. Cluster
    if method == 'kmeans':
        model = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
    elif method == 'dbscan':
        model = DBSCAN(eps=0.5, min_samples=5)
    elif method == 'hierarchical':
        model = AgglomerativeClustering(n_clusters=n_clusters)

    labels = model.fit_predict(X_scaled)

    # 3. Evaluate
    if len(set(labels)) > 1:
        sil = silhouette_score(X_scaled, labels)
        print(f"Silhouette Score: {sil:.3f}")

    # 4. Visualize (PCA to 2D)
    pca = PCA(n_components=2)
    X_2d = pca.fit_transform(X_scaled)

    plt.figure(figsize=(10, 6))
    scatter = plt.scatter(X_2d[:, 0], X_2d[:, 1], c=labels, cmap='viridis')
    plt.colorbar(scatter, label='Cluster')
    plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.1%})')
    plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.1%})')
    plt.title(f'{method.upper()} Clustering Results')
    plt.show()

    return labels, X_scaled

# Usage
labels, X_scaled = unsupervised_pipeline(df, ['col1', 'col2', 'col3'],
                                         method='kmeans', n_clusters=4)
```

1.9 Practice Exercises

1. **Customer Segmentation**: Cluster customers by RFM (Recency, Frequency, Monetary)
2. **Anomaly Detection**: Use DBSCAN to find outliers in transaction data
3. **Document Clustering**: Cluster text embeddings to find topic groups

---

1.10 Next Steps

- Apply to real dataset with business context
- Compare multiple algorithms on same data
- Experiment with different feature combinations
- Read advanced handout for production deployment

The goal is not perfect clusters, but actionable insights. Always validate with domain experts.