Unsupervised Learning - Advanced Handout

Machine Learning for Smarter Innovation

1 Unsupervised Learning - Advanced Handout

Target Audience: Data scientists and ML engineers **Duration**: 60 minutes reading **Level**: Advanced (mathematical foundations, production considerations)

1.1 Mathematical Foundations

1.1.1 K-Means Objective Function

The K-means algorithm minimizes the within-cluster sum of squares (WCSS):

$$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

Where: - K = number of clusters - C_k = set of points in cluster k - μ_k = centroid of cluster k - $\|x_i - \mu_k\|^2$ = squared Euclidean distance

Lloyd's Algorithm: 1. Initialize centroids randomly 2. **Assignment**: $C_k = \{x_i : \|x_i - \mu_k\|^2 \leq \|x_i - \mu_j\|^2 \; \forall j\}$ 3. **Update**: $\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$ 4. Repeat until convergence

Convergence: Guaranteed to converge (monotonically decreasing J), but only to local minimum.

1.1.2 DBSCAN Formal Definitions

Core Point: Point p is a core point if $|N_\epsilon(p)| \geq \text{MinPts}$

Where $N_\epsilon(p) = \{q \in D : d(p, q) \leq \epsilon\}$

Directly Density-Reachable: Point q is directly density-reachable from p if: - p is a core point - $q \in N_\epsilon(p)$

Density-Reachable: Point q is density-reachable from p if there exists a chain $p_1, ..., p_n$ where $p_1 = p$, $p_n = q$, and each p_{i+1} is directly density-reachable from p_i.

Cluster: Maximal set of density-connected points.

Time Complexity: $O(n^2)$ naive, $O(n \log n)$ with spatial index (KD-tree, ball tree).

1.1.3 Hierarchical Clustering Linkage

Single Linkage (nearest neighbor):

$$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y)$$

Complete Linkage (farthest neighbor):

$$d(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y)$$

Average Linkage (UPGMA):

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} d(x, y)$$

Ward's Method (minimize variance):

$$d(C_i, C_j) = \sqrt{\frac{2|C_i||C_j|}{|C_i| + |C_j|}} \|\mu_i - \mu_j\|$$

1.1.4 PCA Mathematical Formulation

Given data matrix $X \in \mathbb{R}^{n \times d}$ (centered):

Covariance Matrix: $\Sigma = \frac{1}{n-1} X^T X$

Eigendecomposition: $\Sigma = V \Lambda V^T$

Where V contains eigenvectors (principal components) and Λ contains eigenvalues.

Projection: $Z = XV_k$ where V_k contains top k eigenvectors.

Variance Explained:

$$\text{VE}_k = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$$

1.2 Advanced Clustering Techniques

1.2.1 Gaussian Mixture Models (GMM)

Probabilistic generalization of K-means:

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

Advantages over K-means: - Soft assignments (probabilities) - Non-spherical clusters - Model selection via BIC/AIC

```
from sklearn.mixture import GaussianMixture

# Fit GMM with model selection
bics = []
for k in range(2, 10):
    gmm = GaussianMixture(n_components=k, random_state=42)
    gmm.fit(X_scaled)
```

```
    bics.append(gmm.bic(X_scaled))

optimal_k = np.argmin(bics) + 2
gmm = GaussianMixture(n_components=optimal_k, random_state=42)
labels = gmm.fit_predict(X_scaled)
probs = gmm.predict_proba(X_scaled)  # Soft assignments
```

1.2.2 Spectral Clustering

For non-convex cluster shapes:

1. Build similarity graph (affinity matrix)
2. Compute graph Laplacian
3. Find eigenvectors of Laplacian
4. Apply K-means to eigenvector representation

$$L = D - W \quad \text{(unnormalized)}$$
$$L_{sym} = I - D^{-1/2} W D^{-1/2} \quad \text{(normalized)}$$

```
from sklearn.cluster import SpectralClustering

spectral = SpectralClustering(n_clusters=4, affinity='rbf',
                              gamma=1.0, random_state=42)
labels = spectral.fit_predict(X_scaled)
```

1.3 Validation Theory

1.3.1 Silhouette Coefficient

For point i:

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

Where: - $a(i)$ = mean intra-cluster distance - $b(i)$ = mean nearest-cluster distance

Range: $[-1, 1]$, higher is better.

1.3.2 Calinski-Harabasz Index

$$CH = \frac{SS_B/(k-1)}{SS_W/(n-k)}$$

Where: - SS_B = between-cluster sum of squares - SS_W = within-cluster sum of squares - k = number of clusters - n = number of points

1.3.3 Gap Statistic

Compare WCSS to null reference distribution:

$$\text{Gap}_k = \mathbb{E}[\log W_k^*] - \log W_k$$

Choose smallest k where $\text{Gap}_k \geq \text{Gap}_{k+1} - s_{k+1}$

```python
from sklearn.cluster import KMeans
from scipy.spatial.distance import pdist, squareform

def gap_statistic(X, k_range, n_refs=10):
    gaps = []
    for k in k_range:
        # Actual clustering
        km = KMeans(n_clusters=k, random_state=42, n_init=10)
        km.fit(X)
        wk = km.inertia_

        # Reference datasets
        ref_wks = []
        for _ in range(n_refs):
            X_ref = np.random.uniform(X.min(axis=0), X.max(axis=0), X.shape)
            km_ref = KMeans(n_clusters=k, random_state=42, n_init=10)
            km_ref.fit(X_ref)
            ref_wks.append(km_ref.inertia_)

        gap = np.mean(np.log(ref_wks)) - np.log(wk)
        gaps.append(gap)

    return gaps
```

1.4 Production Considerations

1.4.1 Scalability

Algorithm	Time	Memory	Scalable Version
K-means	$O(nkdi)$	$O(nd)$	Mini-batch K-means
DBSCAN	$O(n^2)$	$O(n^2)$	HDBSCAN
Hierarchical	$O(n^3)$	$O(n^2)$	BIRCH
GMM	$O(nk^2d^2)$	$O(kd^2)$	Online GMM

1.4.2 Mini-Batch K-Means for Large Data

```python
from sklearn.cluster import MiniBatchKMeans

mbk = MiniBatchKMeans(n_clusters=10, batch_size=1000,
                      random_state=42, n_init=10)

# Incremental fitting for streaming data
for batch in data_generator():
    mbk.partial_fit(batch)

labels = mbk.predict(X_new)
```

1.4.3 Handling High Dimensions

1. **Curse of dimensionality**: Distances become meaningless in high-D
2. **Solutions**:
    - PCA before clustering

- Feature selection
- Use cosine similarity instead of Euclidean
- Subspace clustering

```
# Cosine similarity for text/embeddings
from sklearn.preprocessing import normalize

X_normalized = normalize(X)  # L2 normalize
# Now Euclidean distance = sqrt(2 * (1 - cosine_similarity))
```

1.5 Cluster Stability

1.5.1 Bootstrap Stability Analysis

```
from sklearn.metrics import adjusted_rand_score

def cluster_stability(X, n_clusters, n_bootstrap=100):
    """Measure cluster stability via bootstrap resampling."""
    n = len(X)
    km = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
    base_labels = km.fit_predict(X)

    ari_scores = []
    for _ in range(n_bootstrap):
        # Bootstrap sample
        idx = np.random.choice(n, n, replace=True)
        X_boot = X[idx]

        # Cluster bootstrap sample
        boot_labels = KMeans(n_clusters=n_clusters,
                             random_state=42, n_init=10).fit_predict(X_boot)

        # Compare to base (on common points)
        ari = adjusted_rand_score(base_labels[idx], boot_labels)
        ari_scores.append(ari)

    return np.mean(ari_scores), np.std(ari_scores)

stability, std = cluster_stability(X_scaled, n_clusters=4)
print(f"Stability: {stability:.3f} (+/- {std:.3f})")
```

1.6 Model Persistence and Deployment

```
import joblib

# Save model and scaler
joblib.dump(kmeans, 'models/kmeans_model.pkl')
joblib.dump(scaler, 'models/scaler.pkl')

# Load and predict
kmeans_loaded = joblib.load('models/kmeans_model.pkl')
scaler_loaded = joblib.load('models/scaler.pkl')
```

```
def predict_cluster(new_data):
    """Production prediction function."""
    X_new = scaler_loaded.transform(new_data)
    cluster = kmeans_loaded.predict(X_new)
    distance = kmeans_loaded.transform(X_new).min(axis=1)
    return cluster, distance
```

1.7 Monitoring and Drift Detection

```
def detect_cluster_drift(X_baseline, X_new, threshold=0.1):
    """Detect if new data distribution has drifted from baseline."""
    from scipy.stats import ks_2samp

    drift_detected = False
    for i in range(X_baseline.shape[1]):
        stat, p_value = ks_2samp(X_baseline[:, i], X_new[:, i])
        if p_value < threshold:
            print(f"Feature {i}: Drift detected (p={p_value:.4f})")
            drift_detected = True

    return drift_detected
```

1.8 References

1. MacQueen, J. (1967). “Some methods for classification and analysis of multivariate observations”
2. Ester, M., et al. (1996). “A density-based algorithm for discovering clusters”
3. Ward, J. H. (1963). “Hierarchical grouping to optimize an objective function”
4. Tibshirani, R., et al. (2001). “Estimating the number of clusters via the gap statistic”

In production, cluster quality depends not just on metrics but on business impact. Monitor, iterate, and validate with domain experts.