From Supervised Metrics to Unsupervised Clustering:

A Practical Revision of Evaluation Methods and K-Means with Spotify Data

Minor In AI, IIT Ropar

6th June 2025

Welcome, aspiring AI explorers!

Our journey through the world of Artificial Intelligence is like learning to navigate a vast new city. We've already explored some exciting neighborhoods, like building models that learn from examples (that's our supervised learning area!). Today, we're going to revisit how we know if our navigation is good, and then venture into a different kind of discovery: finding patterns hidden in plain sight (that's our unsupervised learning area!).

We'll start with a real-world idea, see how it connects to AI, dive into evaluating models with 'teachers' (supervised), explore models without 'teachers' (unsupervised), and see how we evaluate those too. We'll use some music data from Spotify to make things concrete!

1 Seeing Patterns in the Wild

Imagine you're walking through a forest. What do you see? You might notice groups of trees growing close together, a cluster of mushrooms on a log, a herd of deer grazing, or perhaps a swarm of butterflies fluttering around a flower patch. These are all examples of *clusters* in nature. Things that are similar in some way grouping together.

Figure 1: Clusters of trees, mushrooms, herd of animals, swarm of butterflies.

Even in human society, we see clustering. Think about people who share a common interest. They might start spending time together, form groups, maybe even live in the same neighborhood. Slowly, a community, or a 'cluster', of like-minded individuals emerges.

This natural tendency for similar things to clump together is a fundamental concept that inspires many AI techniques, particularly in the area we call *clustering*. But how do we know if our AI model has found *meaningful* clusters? And how do we know if our supervised models are making *good* predictions? That's where evaluation metrics come in.

2 Learning With and Without a Teacher: A Quick Recap

Before we talk about evaluation, let's quickly remind ourselves of the two main ways AI models learn:

- **Supervised Learning:** This is like learning with a teacher. You have data with clear labels (like "this is an apple," "this is an orange"). The model learns to recognize patterns

in the data to predict these labels for new, unseen data. We saw examples like Linear Regression (predicting a number) or Support Vector Machines (SVM - classifying data into categories based on labeled examples).

- **Unsupervised Learning:** This is like exploring on your own without labels. You have data, but no one tells you what the groups are. The model tries to find hidden patterns, structures, or groupings within the data by itself. Clustering algorithms, which find these natural clumps of data, fall into this category.

Knowing whether your model is supervised or unsupervised is important because you'll use different tools to see how well it's doing!

3 Measuring Our Supervised Success: The Confusion Matrix and Beyond

For supervised learning, where we have the 'correct' labels (the "Ground Truth") to compare against the model's "Predicted" labels, we have powerful tools to measure performance. One of the most fundamental is the **Confusion Matrix**.

Why "Confusion Matrix"? Because it helps us see exactly where our model got 'confused'! It shows us all the possible outcomes when we predict a category:

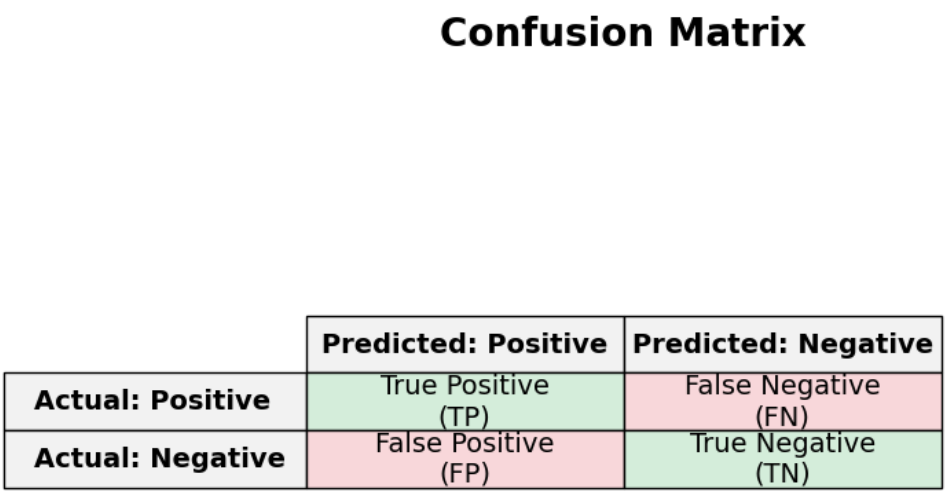

Figure 2: True Positive, True Negative, False Positive, False Negative

Let's break down what those boxes mean using a simple example where we're predicting if something is "Positive" (like 'spam' email) or "Negative" (like 'not spam' email):

- **True Positive (TP):** The actual email *was* spam, and our model correctly *predicted* it as spam. Great!
- **False Negative (FN):** The actual email *was* spam, but our model wrongly *predicted* it as *not* spam. Uh oh, spam is getting through!

- **False Positive (FP):** The actual email *was not* spam, but our model wrongly *predicted* it as spam. Not good, we're marking important emails as spam!
- **True Negative (TN):** The actual email *was not* spam, and our model correctly *predicted* it as *not* spam. Also great!

From these four counts, we can calculate several key metrics:

1. **Accuracy:** How many predictions were correct *overall*?
   - Intuition: (Correct Positives + Correct Negatives) / Total Predictions
   - Formula: $\frac{TP+TN}{TP+TN+FP+FN}$
2. **Precision:** When the model *predicted* something was positive, how often was it *actually* positive? Useful when minimizing False Positives is important (e.g., don't want to flag important emails as spam).
   - Intuition: Correctly Predicted Positives / All Predicted Positives
   - Formula: $\frac{TP}{TP+FP}$
   - *Tip (from lecture):* Think of the "Predicted Positive" column in the Confusion Matrix!
3. **Recall (Sensitivity):** When something was *actually* positive, how often did the model correctly *identify* it? Useful when minimizing False Negatives is important (e.g., don't want to miss actual spam emails).
   - Intuition: Correctly Identified Actual Positives / All Actual Positives
   - Formula: $\frac{TP}{TP+FN}$
   - *Tip (from lecture):* Think of the "Actual Positive" row in the Confusion Matrix!
4. **F1 Score:** A single score that balances both Precision and Recall. Useful when you need a good balance of both.
   - Formula: $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

While understanding the formulas is helpful for intuition, in practice, we use libraries that calculate these for us:

```python
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
import numpy as np

# Example data (like the lecture's simple 0s and 1s)
# Let's say 1 is "Positive" and 0 is "Negative"
ground_truth = np.array([1, 0, 1, 1, 0, 0, 1, 0, 1, 0]) # Actual state
predicted = np.array([1, 0, 1, 0, 0, 1, 1, 0, 1, 0])   # Model's prediction

# Calculate Confusion Matrix
# Output order is usually [[TN, FP], [FN, TP]]
tn, fp, fn, tp = confusion_matrix(ground_truth, predicted).ravel()

print(f"Confusion Matrix:")
print(f"TP: {tp}, FN: {fn}")
print(f"FP: {fp}, TN: {tn}")

# Calculate metrics using scikit-learn APIs
accuracy = accuracy_score(ground_truth, predicted)
```

```python
precision = precision_score(ground_truth, predicted) # for the 'positive' class (1)
recall = recall_score(ground_truth, predicted)     # for the 'positive' class (1)
f1 = f1_score(ground_truth, predicted)           # for the 'positive' class (1)

print(f"\nMetrics (Using APIs):")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

Listing 1: Calculating supervised metrics in Python.

Code Output

```
Confusion Matrix:
TP: 3, FN: 2
FP: 1, TN: 4

Metrics (Using APIs):
Accuracy: 0.70
Precision: 0.75
Recall: 0.60
F1 Score: 0.67
```

As you can see, the APIs give you the results directly. These metrics are essential for understanding how well your supervised model performs on a classification task.

4 Discovering Groups Without Labels: K-Means Clustering

Now, let's switch gears to unsupervised learning and clustering. Imagine you have a massive collection of songs (like from Spotify!) and you want to group them into different genres or moods, but no one has pre-labeled them. This is a job for clustering algorithms like **K-Means**.

K-Means works by finding centers (called *centroids*) for a specified number of clusters (that's the 'K' in K-Means). The core idea is simple:

1. **Choose K:** Decide how many clusters (K) you want to find.
2. **Initialize Centroids:** Randomly pick K points from your data to be the initial centers (centroids) of your clusters.
3. **Assign Points to Clusters:** Look at every data point and calculate its distance (usually Euclidean distance) to *each* of the K centroids. Assign the point to the cluster whose centroid is nearest.
4. **Update Centroids:** Once all points are assigned, recalculate the centroid for each cluster. The new centroid is simply the average (mean) of all the points now assigned to that cluster.
5. **Repeat:** Go back to step 3 and repeat the process (assigning points to the *new* nearest centroids, then recalculating centroids). Keep repeating until the centroids no longer move significantly, or the points stop changing cluster assignments. The algorithm has *converged*.

Let's see this in action with some synthetic (computer-generated) data:

```python
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np

# Generate synthetic data
# _ is a 'throwaway' variable as make_blobs returns two values,
# but we only need the data (X)
X, _ = make_blobs(n_samples=300, centers=4, random_state=42)

# Apply K-Means
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
kmeans.fit(X) # Find the clusters in the data
labels = kmeans.labels_ # Get the cluster assignment for each point
centroids = kmeans.cluster_centers_ # Get final centroid coordinates

# Plot the results
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', s=50, alpha
    =0.6)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=200, label
    ='Centroids')
plt.title('K-Means Clustering on Synthetic Data (K=4)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True)
plt.show()
```

Listing 2: Applying K-Means to synthetic data.

You can experiment by changing `n_clusters` (the K value) in the `KMeans` line. See how K-Means *always* tries to make exactly K clusters, even if the data doesn't naturally have that many distinct groups. This brings us to a crucial question...

5 Finding the Right Number of Groups: Evaluating Unsupervised Clustering

Since unsupervised learning doesn't have ground truth labels, how do we know if the clusters we found are *good*? How do we choose the best value for K in K-Means? We need different evaluation methods!

Two common methods are the **Elbow Method** and the **Silhouette Score**.

1. **Elbow Method:** This method helps find a good value for K by looking at how the "Inertia" (or Within-Cluster Sum of Squares - WCSS) changes as K increases.
    - **Intuition:** As you increase K, each point gets closer to its centroid, so Inertia decreases. But adding more clusters beyond the 'natural' number gives diminishing returns – the drop in Inertia slows down.
    - **How it works:** Calculate Inertia for a range of K values. Plot Inertia vs. K. Look for the "elbow" shape in the plot, where the sharp decrease suddenly bends. The K value at this "elbow" is often a good choice.
2. **Silhouette Score:** This method measures how similar a point is to its own cluster compared to other clusters.
    - **Intuition:** A good cluster is one where points are close to each other (within their cluster) and far from points in other clusters.

- **How it works:** For each point, calculate:
  - `a`: The average distance to all other points *in the same cluster*.
  - `b`: The average distance to all points in the *nearest other cluster*.

  The silhouette score for that point is $\frac{b-a}{\max(a,b)}$.
- **Score Range: -1 to +1.** Scores near +1 are good. Scores near 0 are on the boundary. Scores near -1 are likely misclassified. We want to find the K that maximizes the average silhouette score.

Let's apply this to our Spotify dataset!

```python
# Assumes you have uploaded data.csv to your environment
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings('ignore') # Suppress warnings

# Load the dataset
df = pd.read_csv('data.csv')

# Select features for clustering
features = ['danceability', 'energy', 'loudness', 'speechiness',
            'acousticness', 'instrumentalness', 'liveness', 'valence']
X_spotify = df[features].dropna()

# Standardize the features (important for K-Means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_spotify)

# Calculate Inertia and Silhouette Scores for a range of K values
inertia_values = []
silhouette_scores = []
k_range = range(2, 12) # K from 2 to 11

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia_values.append(kmeans.inertia_)
    score = silhouette_score(X_scaled, kmeans.labels_)
    silhouette_scores.append(score)

# Plot the Elbow Method graph
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(k_range, inertia_values, marker='o')
plt.title('Elbow Method for Spotify Data')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia (WCSS)')
plt.xticks(k_range)
plt.grid(True)

# Plot the Silhouette Scores graph
plt.subplot(1, 2, 2)
plt.plot(k_range, silhouette_scores, marker='o')
plt.title('Silhouette Scores for Spotify Data')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.xticks(k_range)
```

```
plt.grid(True)

plt.tight_layout()
plt.show()
```

Listing 3: Finding the optimal K for Spotify data.

When you run the code above, it will generate two plots. Looking at them:

- **Elbow Method Plot:** We look for where the curve bends like an elbow. Around K=7 or K=8 seems to be where the rate of decrease in Inertia slows down.
- **Silhouette Score Plot:** We look for the peak. K=8 likely has the highest silhouette score among the tested values.

Combining these, **K=8** appears to be a reasonable number of clusters for this Spotify dataset based on these metrics.

6 Beyond Spheres: Density-Based Clustering (DBScan)

K-Means is great for finding spherical, similarly sized clusters. But what if your data has clusters of weird shapes, or contains outliers? This is where **DBScan (Density-Based Spatial Clustering of Applications with Noise)** comes in. It groups points based on how densely they are packed. It discovers clusters based on two parameters:

- `eps` **(epsilon):** The maximum distance between two samples for one to be considered as in the neighborhood of the other.
- `min_samples`**:** The number of samples in a neighborhood for a point to be considered a *core point*.

DBScan is powerful because it can find clusters of arbitrary shapes and naturally identifies outliers (points labeled as noise). Let's see it in action.

```
from sklearn.datasets import make_moons, make_circles
from sklearn.cluster import KMeans, DBSCAN
import matplotlib.pyplot as plt
import numpy as np

# Generate synthetic data that is not globular
X_moons, _ = make_moons(n_samples=200, noise=0.05, random_state=42)

# Apply K-Means
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans_labels = kmeans.fit_predict(X_moons)

# Apply DBScan (requires tuning eps and min_samples)
dbscan = DBSCAN(eps=0.3, min_samples=5) # Example parameters
dbscan_labels = dbscan.fit_predict(X_moons)

# Plot the results
plt.figure(figsize=(12, 5))

# K-Means Plot
plt.subplot(1, 2, 1)
plt.scatter(X_moons[:, 0], X_moons[:, 1], c=kmeans_labels, cmap='viridis')
plt.title('K-Means (K=2) on Moons Data')
plt.grid(True)

# DBScan Plot
```

```
plt.subplot(1, 2, 2)
plt.scatter(X_moons[:, 0], X_moons[:, 1], c=dbscan_labels, cmap='viridis')
plt.title('DBScan (eps=0.3, min_samples=5) on Moons Data')
plt.grid(True)

plt.tight_layout()
plt.show()
```

Listing 4: Comparing K-Means and DBScan on non-spherical data.

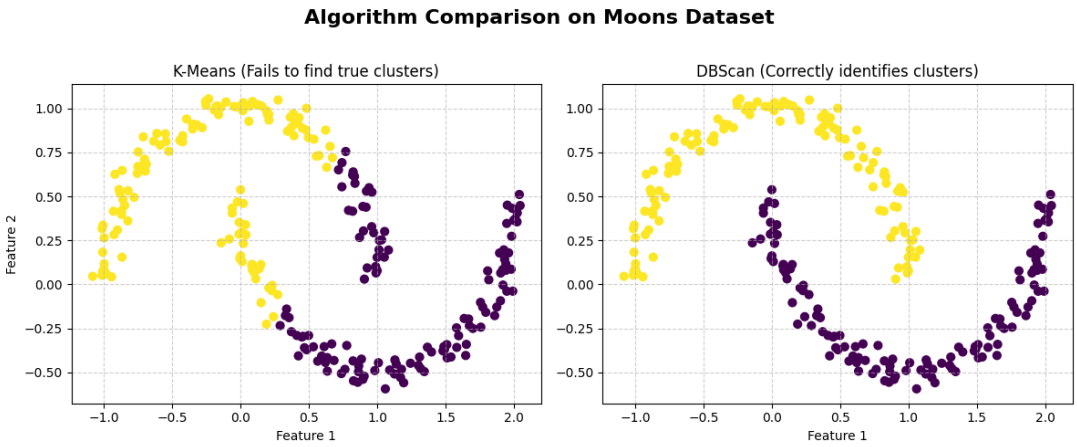

Figure 3: K-Means vs DBScan

See how DBScan (with appropriate parameters) finds the natural, non-spherical clusters while K-Means struggles? The points labeled -1 are considered noise.

7 Putting It All Together & Your Turn!

We've revised supervised and unsupervised evaluation. We saw how Elbow and Silhouette plots can guide your choice of K. As a next step, try applying DBScan to the Spotify data!

Another dataset to explore is `load_digits`. You'll likely want to use **PCA (Principal Component Analysis)**, another unsupervised method, to shrink the many features down to 2 for visualization before clustering.

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

# Load the digits dataset
digits = load_digits()
X_digits = digits.data
y_digits = digits.target

# Apply PCA to reduce to 2 components for visualization
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_digits)

# Now X_pca is a 2D representation of the digits data.
# Plot the PCA-reduced data (colored by actual digit)
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_digits, cmap='tab10', alpha=0.7)
plt.title('Digits Data after PCA to 2D')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Actual Digit')
```

```
plt.grid(True)
plt.show()

# TODO: Apply KMeans or DBScan to X_pca and plot the results!
```

Listing 5: Using PCA to visualize high-dimensional data.

8 Which Technique for the Job? A Scenario Challenge

Finally, let's test your understanding. For each scenario, which technique seems most appropriate?

1. A bank wants to predict if a customer will repay a loan (Yes/No) based on income, credit score, etc. The outcome is binary.
Appropriate Technique: _____

2. You have high-dimensional medical data and want to reduce the number of features while keeping most of the important information.
Appropriate Technique: _____

3. An e-commerce site wants to group its customers into different segments based on purchasing history. You don't know how many segments there should be, and some customers might be outliers.
Appropriate Technique: _____

4. A search engine needs to rank web pages based on their importance by analyzing how pages link to each other.
Appropriate Technique: _____

5. You are building a system to identify different types of objects (like cars, people, trees) within images.
Appropriate Technique: _____

9 Conclusion

Phew! We covered a lot of ground, from supervised evaluation to K-Means and DBScan clustering. We saw how abstract concepts are inspired by the real world and implemented using powerful libraries.

Remember, the best way to learn is by doing. Experiment with the code, change parameters, and see how the results change. Keep exploring, and you'll continue building your intuition and skills in AI!

Good luck with your studies!