Clustering and Naive Bayes

Introduction

Welcome to this module of our AI course! Today we're going to learn about two powerful machine learning techniques: **Clustering and Naive Bayes**. These techniques help computers **find patterns and make predictions** without being explicitly programmed to do so.

By the end of this module, you'll understand how computers can **group similar things together** (**clustering**) and how they can make predictions based on probability (Naive Bayes).

1. What is Clustering?

Clustering is a technique that helps us **find natural groupings in data**. It's like sorting a pile of mixed fruits into separate baskets based on what type of fruit they are - without being told in advance what the different types are!

Why Do We Use Clustering?

Clustering helps us:

- Discover **hidden patterns** in data
- **Group similar items** together
- Identify **unusual data points** (outliers)
- Reduce **complex data into simpler** categories

Real-Life Examples

Clustering is used in many places:

- Customer segmentation in marketing (grouping similar customers)
- Image compression (grouping similar colors)
- Document categorization (grouping similar articles or papers)
- Identifying groups of genes that behave similarly

💡 **Interactive Question #1**: A streaming service wants to group its movies into categories without using pre-defined genres. This is an example of:

A) Classification

B) Regression

C) Clustering

D) Dimensionality reduction

AMEYA

2. Types of Clustering Algorithms

There are several ways to group data. Let's look at the most common ones:

2.1 K-Means Clustering

K-Means is one of the simplest and most popular clustering methods. Here's how it works:

1. **Choose K** (the number of clusters you want)
2. Randomly place K **"centroids"** (central points) in your data
3. Assign each data point to the **nearest centroid**
4. Move each centroid to the **average position of all points** assigned to it
5. **Repeat steps 3-4** until the centroids stop moving significantly

K-means is like playing a game where you place K flags in a field, then everyone runs to the nearest flag. After everyone is standing by a flag, you move each flag to the center of its group. Then everyone runs again to the nearest flag. This continues until nobody needs to run anymore.

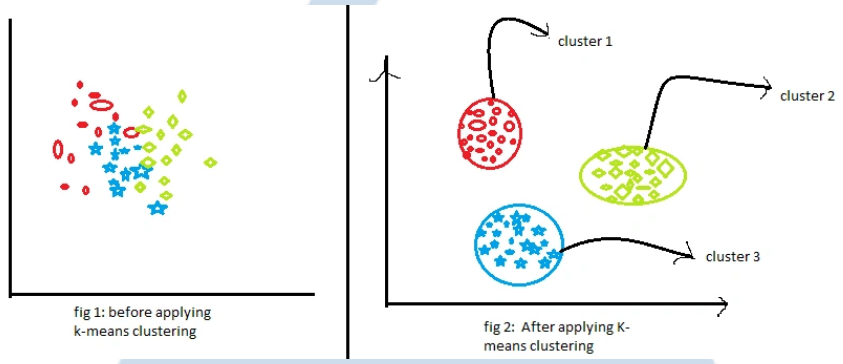

Image Source: https://www.analyticsvidhya.com/blog/2020/10/a-simple-explanation-of-k-means-clustering/

Advantages:

- Simple to understand and implement
- Works well for evenly-sized, well-separated clusters
- Fast for large datasets

Limitations:

- You need to specify K (number of clusters) in advance
- Assumes clusters are circular/spherical
- Struggles with clusters of different sizes and densities

Let's see a simple Python example:

No Copy-Paste, Only Typing

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

# Generate some sample data with 3 natural clusters
np.random.seed(42)
x1 = np.random.normal(0, 1, 100)
y1 = np.random.normal(0, 1, 100)
x2 = np.random.normal(5, 1, 100)
y2 = np.random.normal(5, 1, 100)
x3 = np.random.normal(3, 1, 100)
y3 = np.random.normal(0, 1, 100)

X = np.column_stack([np.concatenate([x1, x2, x3]),
                     np.concatenate([y1, y2, y3])])

# Apply K-means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X)

# Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='viridis', alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            c='red', marker='X', s=100)
plt.title('K-Means Clustering Results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

print(f"Cluster centers: {kmeans.cluster_centers_}")
```

No Copy-Paste, Only Typing

💡 **Interactive Question #2**: In K-means clustering, what does the "K" represent?

A) The number of iterations to run the algorithm

B) The number of clusters you want to create

C) The number of data points in your dataset

D) The maximum distance between points in a cluster

2.2 Hierarchical Clustering

Unlike K-means, hierarchical clustering doesn't require us to specify the number of clusters in advance. It creates a tree-like structure showing how data points can be grouped at different levels.

There are two main approaches:

- **Agglomerative** (bottom-up): Start with each point as its own cluster, then gradually merge the closest ones
- **Divisive** (top-down): Start with all points in one cluster, then gradually split them

The results are usually shown as a dendrogram - a tree diagram showing how clusters merge or split.

2.3 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN **groups points** based on **how densely packed** they are. It's great for finding clusters of **irregular shapes and identifying outliers**.

Here's how it works:

1. For each point, check how many other points are within a certain distance (ε)
2. If a point has at least a minimum number of neighbors (minPts), it's a "core point"
3. Connect core points that are close to each other to form clusters
4. Points that connect to a cluster but don't have enough neighbors are "border points"
5. Points that don't belong to any cluster are "noise points" (outliers)

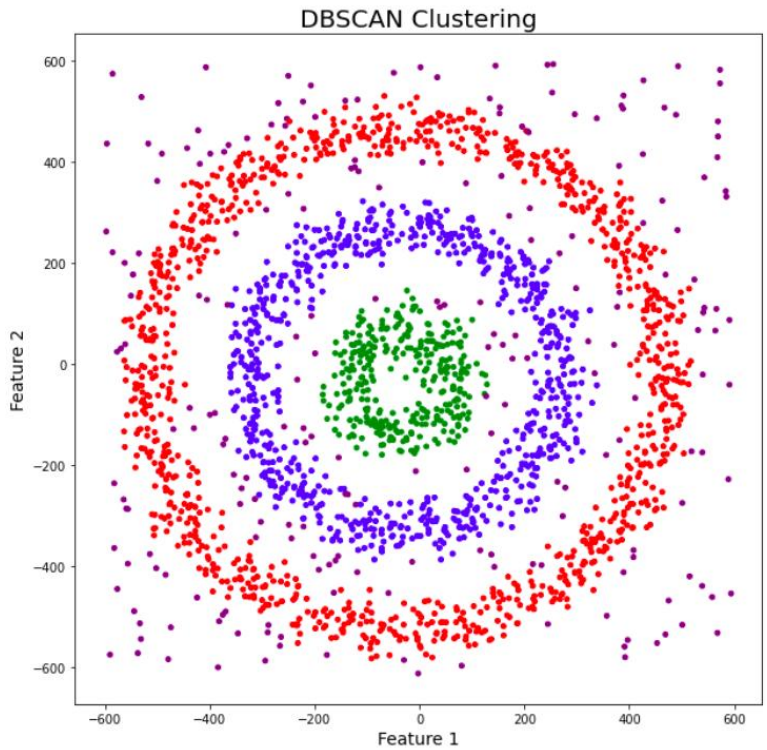

Image Source: https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/

💡 **Interactive Question #3**: Which clustering algorithm would be best for finding oddly-shaped clusters and automatically identifying outliers?

A) K-means

B) Hierarchical clustering

C) DBSCAN

D) Principal Component Analysis

3. How to Choose the Right Number of Clusters

One of the challenges in clustering is deciding how many clusters to look for. Here are some methods to help:

3.1 Elbow Method

1. Run K-means with **different values of K**
2. For each K, calculate the **"inertia"** (**sum of squared distances of each point to its closest centroid**)
3. Plot K vs. inertia
4. Look for an "elbow" in the plot - a point where adding more clusters doesn't reduce inertia much

Think of it like this: When you bend your arm, your elbow is where the direction changes most clearly. Similarly, on our graph, the **"elbow"** is where adding more clusters stops making a **big difference**.

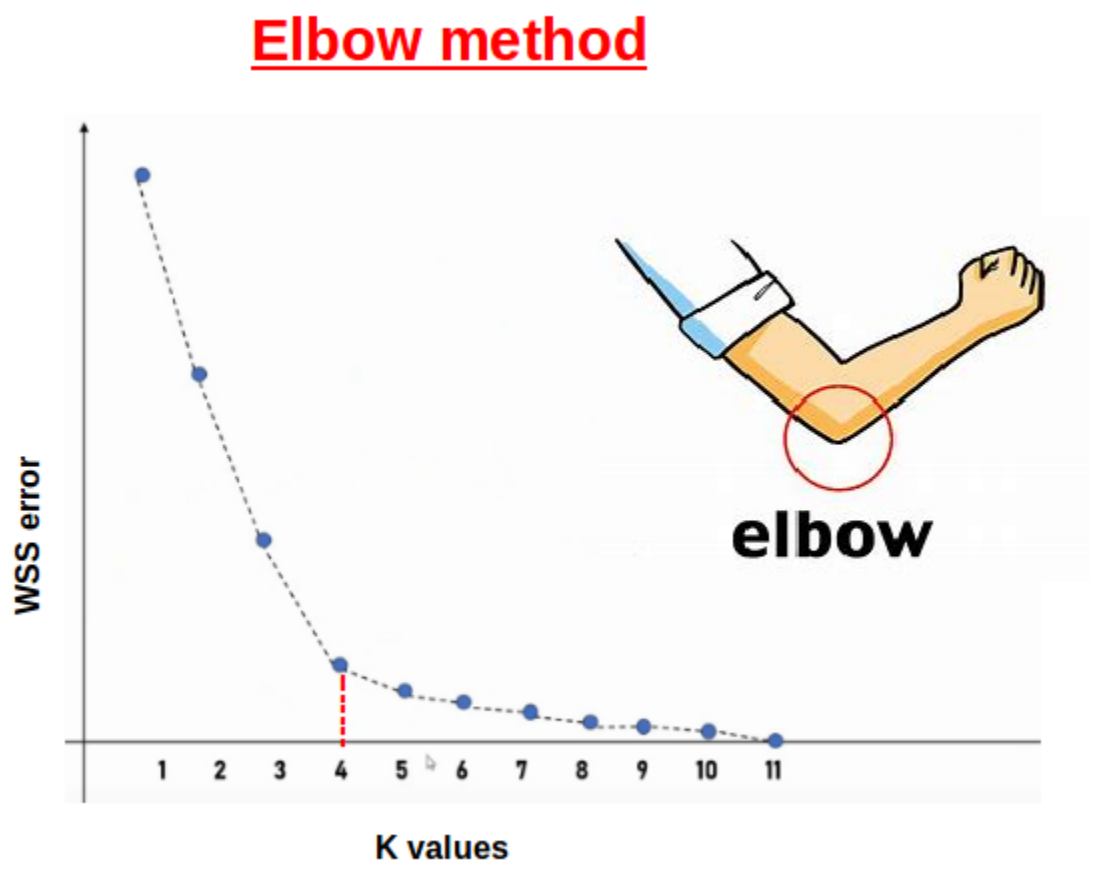

Image Source: https://medium.com/@zalarushirajsinh07/the-elbow-method-finding-the-optimal-number-of-clusters-d297f5aeb189

No Copy-Paste, Only Typing

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

# Try different numbers of clusters
scores = []  # Will store how well each K value performs
K_values = range(1, 10)  # Try K from 1 to 9

# Calculate score for each K
for k in K_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    scores.append(kmeans.inertia_)  # Lower score = better clusters

# Plot the results
plt.figure(figsize=(8, 5))
plt.plot(K_values, scores, 'o-')
plt.xlabel('Number of clusters (K)')
plt.ylabel('Score (lower is better)')
plt.title('Elbow Method: Finding the Best K')
plt.annotate('Elbow point', xy=(3, scores[2]),
             xytext=(4, scores[2]+300),
             arrowprops=dict(arrowstyle='->'))
plt.grid(True)
plt.show()
```

No Copy-Paste, Only Typing

AMEYA

💡 **Interactive Question #4**: You're using the elbow method to determine the optimal number of clusters. The inertia values for K=1 through K=5 are: [1000, 500, 300, 250, 220]. What is likely the optimal number of clusters?

A) 1

B) 2

C) 3

D) 5

4. Naive Bayes Classification

Now, let's switch gears and learn about Naive Bayes - a **classification algorithm based on probability**.

4.1 What is Naive Bayes?

Naive Bayes is a family of probabilistic classifiers based on **Bayes' theorem**. Despite its simplicity, it often performs surprisingly well, especially for text classification.

The "naive" part comes from assuming that all features are independent of each other, which simplifies the calculations (even though this assumption is rarely true in real life).

4.2 Bayes' Theorem

Bayes' theorem is a **mathematical formula** that shows how to **update the probability** of a hypothesis when **given evidence**. The formula is:

$P(A|B) = P(B|A) \times P(A) / P(B)$

Where:

- $P(A|B)$ is the probability of A given B (posterior)
- $P(B|A)$ is the probability of B given A (likelihood)
- $P(A)$ is the probability of A (prior)
- $P(B)$ is the probability of B (evidence)

For classification, we're trying to find P(class|features) - the probability of a class given the features.

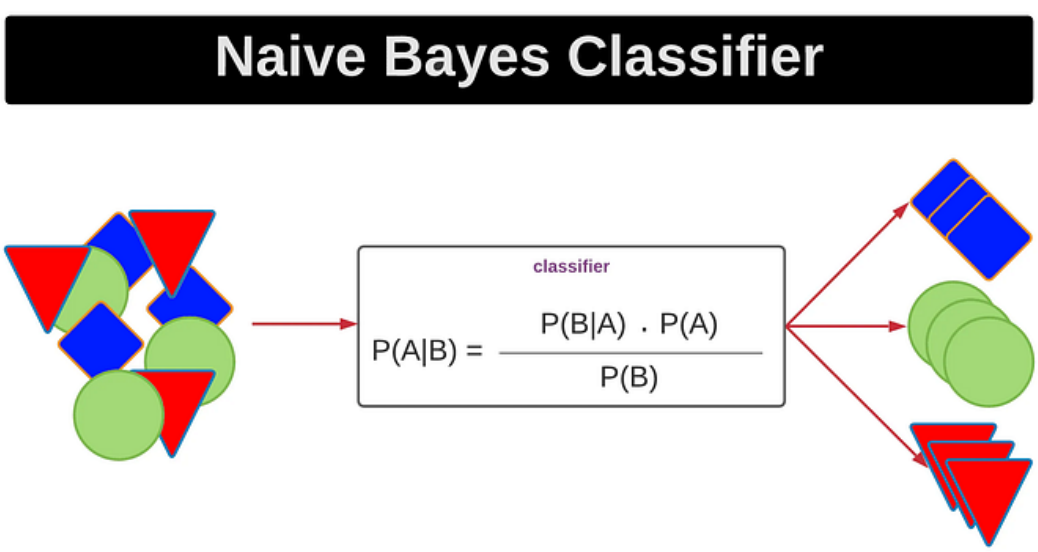

Image Source: https://medium.com/@patwariraghottam/mastering-naive-bayes-a-comprehensive-python-guide-to-probabilistic-classification-b7fe67c6763f

4.3 How Naive Bayes Works

For a given set of features ($X_1, X_2, ..., X_n$) and a class label Y:

1. We want to find the most likely class: $P(Y|X_1, X_2, ..., X_n)$
2. Using Bayes' theorem: $P(Y|X_1, X_2, ..., X_n) = P(X_1, X_2, ..., X_n|Y) \times P(Y) / P(X_1, X_2, ..., X_n)$
3. The "naive" assumption: features are independent given the class
4. This lets us simplify: $P(X_1, X_2, ..., X_n|Y) = P(X_1|Y) \times P(X_2|Y) \times ... \times P(X_n|Y)$
5. We calculate the probability for each class and pick the one with the highest probability

💡 **Interactive Question #5**: What does the "naive" in Naive Bayes refer to?

A) The algorithm is simple compared to other methods

B) It assumes all features are independent of each other

C) It makes predictions based on limited knowledge

D) It was created by someone named Naive Bayes

4.4 Types of Naive Bayes

There are three main types of Naive Bayes classifiers:

4.4.1 Gaussian Naive Bayes

- Used for continuous data
- Assumes features follow a normal (Gaussian) distribution
- Good for values like height, weight, or test scores

4.4.2 Multinomial Naive Bayes

- Used for discrete counts (like word counts)
- Perfect for text classification
- Often used for spam detection or document categorization

4.4.3 Bernoulli Naive Bayes

- Used for binary/boolean features (true/false)
- Common for text classification when we only care if a word appears or not
- Good for short texts like tweets or reviews

AMEYA

4.5 Example: Text Classification with Naive Bayes

Let's see a simple example of using Naive Bayes for spam detection:

No Copy-Paste, Only Typing

```python
import numpy as np
from sklearn.feature_extraction.text import CountVectorize
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Sample emails
emails = [
    # Spam emails
    "Get rich quick! Guaranteed money",
    "Win a free iPhone! Click here now",
    "Congratulations! You've won a prize",
    "Amazing weight loss pill, lose 20 pounds in a week",
    # Non-spam emails
    "Meeting scheduled for tomorrow at 10am",
    "Please review the attached document",
    "Your monthly report is ready",
    "Thank you for your recent purchase"
]

# Labels: 1 for spam, 0 for non-spam
labels = np.array([1, 1, 1, 1, 0, 0, 0, 0])

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    emails, labels, test_size=0.25, random_state=42
)
```

```python
# Convert text to numerical features (bag of words)
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)

# Train the Naive Bayes classifier
clf = MultinomialNB()
clf.fit(X_train_counts, y_train)

# Make predictions
y_pred = clf.predict(X_test_counts)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Test with new emails
new_emails = [
    "Congratulations! You won a lottery ticket",
    "Can we reschedule our meeting to Friday?"
]
new_counts = vectorizer.transform(new_emails)
predictions = clf.predict(new_counts)

print("\nPredictions for new emails:")
for email, pred in zip(new_emails, predictions):
    print(f"Email: '{email}'")
    print(f"Prediction: {'Spam' if pred == 1 else 'Not Spam'}")
```

No Copy-Paste, Only Typing

AMEYA

4.6 Advantages and Limitations of Naive Bayes

Advantages:

- Simple and fast to train
- Works well with high-dimensional data (like text)
- Performs surprisingly well even when the independence assumption is violated
- Requires less training data than many other algorithms
- Handles missing values well

Limitations:

- The "naive" independence assumption is rarely true in real life
- Doesn't capture relationships between features
- Can be outperformed by more sophisticated models
- May give probability estimates that are not very accurate

💡 **Interactive Question #6**: In which of these scenarios would Naive Bayes likely perform BEST?

A) Predicting house prices based on size, location, and age

B) Classifying emails as spam or not spam

C) Identifying objects in images

D) Forecasting stock market prices

5. Evaluating Our Results

How do we know if our clustering or classification models are any good? Let's look at some simple ways to evaluate them.

5.1 Evaluating Clustering Results

Evaluating clustering can be tricky because we often don't have "correct" labels. Some simple approaches include:

- **Visual inspection**: Sometimes just looking at a plot of the clusters can tell you if they make sense
- **Silhouette score**: Measures how similar points are to their own cluster compared to other clusters (higher is better)

AMEYA

No Copy-Paste, Only Typing

```
from sklearn.metrics import silhouette_score

# Calculate silhouette score for our K-means clustering
score = silhouette_score(X, clusters)
print(f"Silhouette Score: {score:.2f}")  # Closer to 1 is better
```

No Copy-Paste, Only Typing

5.2 Evaluating Classification Results

For classification like our Naive Bayes model, we have several useful metrics:

- **Accuracy**: The **percentage** of **predictions** that were **correct**
  - Example: If our email filter correctly identifies 90 out of 100 emails, that's 90% accuracy
- **Confusion Matrix**: A table showing **correct and wrong** predictions
  - Shows true positives, false positives, true negatives, and false negatives

No Copy-Paste, Only Typing

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Create a simple confusion matrix
y_true = np.array([0, 0, 0, 0, 1, 1, 1, 1])  # Actual labels (0=not spam, 1=spam)
y_pred = np.array([0, 0, 1, 0, 0, 1, 1, 1])  # Predicted labels

# Calculate and display confusion matrix
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Spam', 'Spam'],
            yticklabels=['Not Spam', 'Spam'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Calculate accuracy
accuracy = (cm[0,0] + cm[1,1]) / cm.sum()
print(f"Accuracy: {accuracy:.2f}")
```

No Copy-Paste, Only Typing

- **Precision**: When the model predicts something is positive, **how often is it correct**?
    - Example: When our filter says an email is spam, how often is it really spam?
    - Formula: True Positives / (True Positives + False Positives)
- **Recall**: What **percentage of actual positives** does the model catch?
    - Example: Out of all actual spam emails, how many did our filter catch?
    - Formula: True Positives / (True Positives + False Negatives)

In a spam filter:

- High precision means emails marked as spam are almost certainly spam (few legitimate emails in spam folder)
- High recall means most spam is caught (few spam emails in inbox)

💡 **Interactive Question #7**: What does high precision but low recall indicate about a classification model?

A) The model correctly identifies most positive cases but has many false positives

B) The model correctly identifies most positive cases and has few false positives

C) The model misses many positive cases but when it predicts positive, it's usually correct

D) The model misses many positive cases and has many false positives

AMEYA

Glossary

- **Bayes' Theorem**: A mathematical formula that shows how to update the probability of a hypothesis when given evidence
- **Centroid**: The center point of a cluster in K-means
- **Classification**: Predicting which category something belongs to
- **Clustering**: Grouping similar data points together
- **DBSCAN**: Density-Based Spatial Clustering of Applications with Noise - a clustering algorithm that finds clusters of any shape
- **Dendrogram**: A tree diagram showing the hierarchical relationship between clusters
- **Elbow Method**: A technique to determine the optimal number of clusters
- **Hierarchical Clustering**: A clustering method that creates a tree-like structure of clusters
- **Inertia**: The sum of squared distances of samples to their closest cluster center
- **K-means**: A clustering algorithm that partitions data into K clusters
- **Naive Bayes**: A family of probabilistic classifiers based on Bayes' theorem
- **Outlier**: A data point that differs significantly from other observations
- **Precision**: The proportion of positive identifications that were actually correct
- **Recall**: The proportion of actual positives that were identified correctly
- **Unsupervised Learning**: Learning patterns from data without predefined labels

Multiple Choice Answers

1. C) Clustering
2. B) The number of clusters you want to create
3. C) DBSCAN
4. C) 3
5. B) It assumes all features are independent of each other
6. B) Classifying emails as spam or not spam
7. C) The model misses many positive cases but when it predicts positive, it's usually correct

AMEYA