Name: Shayan Umar

Designation: AI intern

Department: Artificial Intelligence

Date: 1-8-2025

Table of Contents

1. Theoretical Understanding: 3

1.1 Production ML Systems: 3

1.2 Automated Machine Learning (AutoML): 14

1.3 Fairness: 18

2. Unsupervised Machine Learning: 28

3. Practical Learning: 40

4. References: 44

1. Theoretical Understanding:

1.1 Production ML Systems:

Static vs Dynamic Training in Machine Learning

1. Static Training (Offline)

- **Definition**: Train the model once and use it repeatedly without updates.
- **Advantage**: Simple and cost-effective; easier to test and deploy.
- **Disadvantage**: Can become outdated if data patterns change over time.

Example: A flower-purchase prediction model trained only on July–August data performs poorly on Valentine's Day due to changed behavior.

---

2. Dynamic Training (Online)

- **Definition**: Continuously or frequently retrain the model with new data.
- **Advantage**: More adaptable to evolving data and user behavior.
- **Disadvantage**: Requires ongoing work—frequent updates, testing, and deployment.

Key Takeaway

- If data remains stable, **static training** is cheaper and sufficient.
- If data changes over time (which it often does), **dynamic training** helps maintain performance.
- Even with static training, you must **monitor input data** regularly for shifts.

Table 1. Primary advantages and disadvantages.

	Static training	Dynamic training
Advantages	**Simpler.** You only need to develop and test the model once.	**More adaptable.** Your model will keep up with any changes to the relationship between features and labels.
Disadvantages	**Sometimes staler.** If the relationship between features and labels changes over time, your model's predictions will degrade.	**More work.** You must build, test, and release a new product all the time.

Static vs Dynamic Inference in Machine Learning

Inference is the process of making predictions by applying a trained model to **unlabeled examples**. Broadly speaking, a model can infer predictions in one of two ways:

- **Static inference** (also called **offline inference** or **batch inference**) means the model makes predictions on a bunch of common **unlabeled examples** and then caches those predictions somewhere.
- **Dynamic inference** (also called **online inference** or real-time inference) means that the model only makes predictions on demand, for example, when a client requests a prediction.

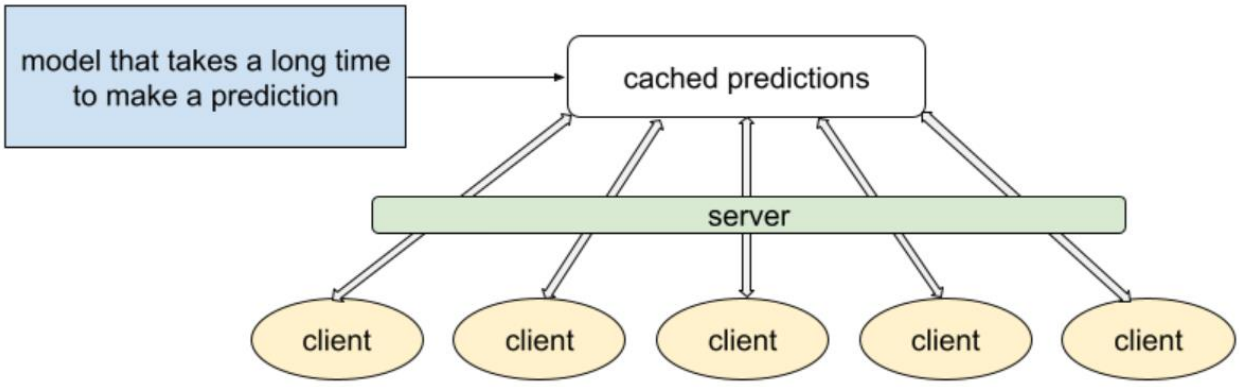


Figure 4. *In static inference, a model generates predictions, which are then cached on a server.*

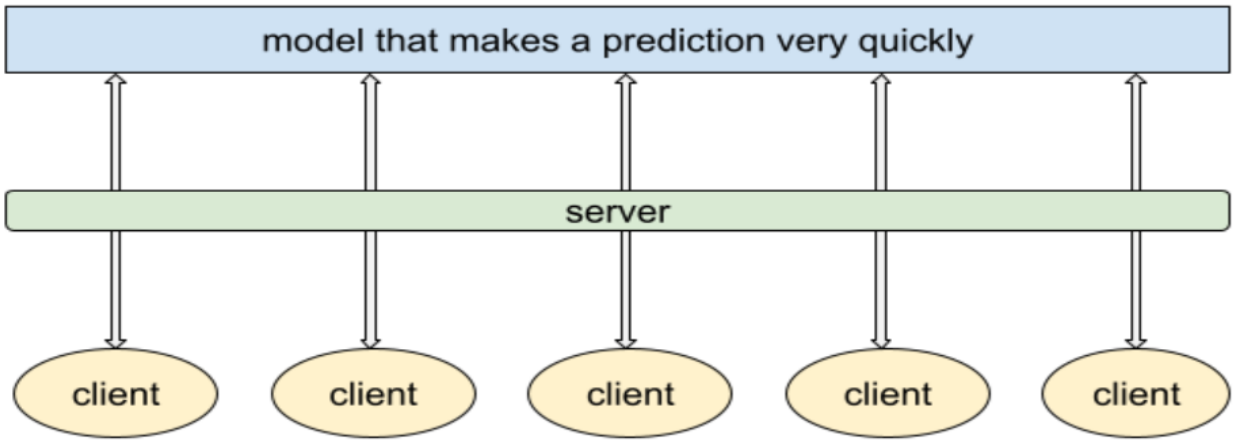


Figure 5. *In dynamic inference, a model infers predictions on demand.*

Static inference

Static inference offers certain advantages and disadvantages.

Advantages

- Don't need to worry much about cost of inference.
- Can do post-verification of predictions before pushing.

Disadvantages

- Can only serve cached predictions, so the system might not be able to serve predictions for uncommon input examples.
- Update latency is likely measured in hours or days.

Dynamic inference

Dynamic inference offers certain advantages and disadvantages.

Advantages

- Can infer a prediction on *any* new item as it comes in, which is great for long tail (less common) predictions.

Disadvantages

- Compute intensive and latency sensitive. This combination may limit model complexity; that is, you might have to build a simpler model that can infer predictions more quickly than a complex model could.
- Monitoring needs are more intensive.

When to Transform Data

Raw data must be feature engineered (transformed). When should you transform data? Broadly speaking, you can perform feature engineering during either of the following two periods:

- *Before* training the model.
- *While* training the model.

Transforming data before training

In this approach, you follow two steps:

1. Write code or use specialized tools to transform the raw data.
2. Store the transformed data somewhere that the model can ingest, such as on disk.

Advantages

- The system transforms raw data only once.
- The system can analyze the entire dataset to determine the best transformation strategy.

Disadvantages

- You must recreate the transformations at prediction time. Beware of **training-serving skew**!

Training-serving skew is more dangerous when your system performs dynamic (online) inference. On a system that uses dynamic inference, the software that transforms the raw dataset usually differs from the software that serves predictions, which can cause training-serving skew. In contrast, systems that use static (offline) inference can sometimes use the same software.

Transforming data while training

In this approach, the transformation is part of the model code. The model ingests raw data and transforms it.

Advantages

- You can still use the same raw data files if you change the transformations.
- You're ensured the same transformations at training and prediction time.

Disadvantages

- Complicated transforms can increase model latency.

- Transformations occur for each and every batch.

Transforming the data per batch can be tricky. For example, suppose you want to use **Z-score normalization** to transform raw numerical data. Z-score normalization requires the mean and standard deviation of the feature. However, transformations per batch mean you'll only have access to *one batch of data*, not the full dataset. So, if the batches are highly variant, a Z-score of, say, -2.5 in one batch won't have the same meaning as -2.5 in another batch. As a workaround, your system can precompute the mean and standard deviation across the entire dataset and then use them as constants in the model.

Deploying the Unicorn Model

Objective

Deploy and maintain a machine learning model that predicts unicorn appearances using a robust and tested pipeline.

---

1. Core Areas to Validate Before Deployment

- **Input Data**: Ensure the input format, range, and values are consistent and clean.
- **Feature Engineering**: Check that feature transformations perform as expected.
- **Model Quality**: Evaluate new versions for:

  Before pushing a new model version to production, test for the following two types of quality degradations:

- **Sudden degradation.** A bug in the new version could cause significantly lower quality. Validate new versions by checking their quality against the previous version.
- **Slow degradation.** Your test for sudden degradation might not detect a slow degradation in model quality over multiple versions. Instead, ensure your model's predictions on a validation dataset meet a fixed threshold. If your validation dataset deviates from live data, then update your validation dataset and ensure your model still meets the same quality threshold.
- **Serving Infrastructure**: Verify the model runs correctly on the production environment.
- **Component Integration**: Ensure that updates to one pipeline component don't break others.

---

2. Challenges with Testing in ML

Unlike traditional software:

- ML models require discovering performance metrics (like loss) before setting meaningful test thresholds.
- Reproducibility is hard due to randomness and nondeterminism in training.

---

3. Tips for Reproducible Training

- **Fix random seeds** to reduce variability.
- **Consistent model initialization** (handled by most libraries).
- **Run and average multiple training runs**.
- **Version control all code and parameters**, even during experimentation.

---

4. Testing Strategies

Unit Tests

- Example: Generate random data and run one step of gradient descent to check API stability.

Integration Tests

- Run the full pipeline (on reduced data or simpler models) to verify that all parts work together.
- Run with every model/software update to catch breakages early.

---

5. Model Quality Checks Before Deployment

- Compare new model to previous version to catch **sudden quality drops**.
- Set **minimum quality thresholds** on a validation set to catch **slow degradation**.
- Regularly update the validation dataset to reflect changes in live data.

---

6. Infrastructure Compatibility

- Ensure the model and server environment are aligned (dependencies, frameworks, etc.).

- Test in a **sandboxed version** of the production server before deploying.

Monitoring Pipelines

1. Define a Data Schema to Validate Raw Input Data

A **data schema** is a set of rules that your raw input data must follow. These rules help catch errors early in your pipeline.

Steps to Define a Schema

- **Understand distributions** of each feature (e.g., ranges, frequency, categories).
- **Write validation rules**, such as:
    - rating ∈ [1, 5]
    - Categorical feature color ∈ {"red", "blue", "green"}
    - Word frequency checks (e.g., "the" should be most common in English text)

Detectable Issues

- Anomalies or outliers
- Unknown categorical values
- Schema drift or distribution shift

---

2. Write Unit Tests to Validate Feature Engineering

Raw data isn't directly used for training; models use **engineered features**. You must test these separately.

Examples of Feature Tests

- Normalized features are between 0 and 1
- Z-score normalization yields a mean near 0
- One-hot vectors contain exactly one 1
- Outliers are clipped or handled
- No NaNs, Infs, or invalid values

---

3. Slice-Based Performance Monitoring

Global metrics can hide poor performance in specific contexts. Monitor **data slices** (e.g., region, demographic) to ensure fairness and robustness.

Example

Your unicorn model performs well globally, but fails in the Sahara region.

What to Do

- Define slices of interest (e.g., region, device type, user segment)
- Compare slice metrics (accuracy, AUC) to overall metrics
- Investigate and correct bias or underperformance

---

4. Real-World Metrics vs. Model Metrics

Don’t rely solely on traditional model metrics like AUC or accuracy.

Real-World Examples

- Track user engagement or satisfaction
- Survey feedback ("Did the unicorn actually appear?")
- Business metrics (e.g., conversions, revenue)

---

5. Detect Training-Serving Skew

Types of Skew

Type	Cause	Solution
Schema Skew	Training and serving data formats or stats differ	Use a shared schema validator across training and serving
Feature Skew	Feature engineering logic differs across environments	Share transformation logic or apply statistical validation consistently

---

6. Check for Label Leakage

Label leakage occurs when the ground truth leaks into features, causing inflated performance during evaluation but poor real-world results.

Example

Model uses hospital_name which correlates with cancer diagnosis due to regional specialization.

Prevention

- Use only features available at prediction time
- Carefully audit high-performance features for proxy labels

---

7. Monitor Model Age and Freshness

Even good models degrade over time if data changes.

Track:

- Time since last retraining
- Data drift metrics (input and prediction distributions)
- Stalls in pipeline (e.g., no updates in N days)

---

8. Validate Numerical Stability

Tests to Include

- No NaN or Inf in weights or outputs
- 50% of output values should be non-zero (catching dead layers)

---

9. Monitor Model and API Performance

System-Level Monitoring

- Training steps per second (track regressions)
- Memory usage over time (catch memory leaks)
- API latency (e.g., 95th percentile response times)
- Throughput (queries/second)

---

10. Test Live Model Quality on Served Data

Real-world data often lacks labels. You still need to test:

Options

- Use human raters to validate a subset
- Monitor bias in predictions
- Compare to downstream events (e.g., clicks, complaints)
- Shadow test new versions on a small % of live traffic

---

11. Ensure Reproducibility with Controlled Randomization

Key Techniques

- **Seed your random number generators**
- **Use deterministic hash keys** (e.g., hash(user_id + date) instead of hash(random))

This ensures consistent data splits, better experiments, and reproducible results.

Questions To Ask in the End

1. Is Each Feature Helpful?

- **Why it's important:** Features that don't improve model performance introduce noise and potential risks.
- **Key takeaway:**
    - Continuously **monitor feature importance**.
    - Remove features that add little value, especially if their **input data can drift or fluctuate**.
- **Example:** A weather-based feature might improve accuracy slightly, but if weather APIs become unreliable, it could hurt performance more than help.

---

2. Does the Usefulness Justify the Cost?

- **Why it's important:** More features = higher **maintenance**, **data gathering**, and **engineering costs**.
- **Key takeaway:**
    - Even if a feature boosts performance slightly, assess whether the **gain is worth the complexity**.
    - Simpler models are **easier to debug**, **monitor**, and **scale**.

---

3. Is Your Data Source Reliable?

- **Why it's important:** Your model is only as good as the **consistency and reliability** of its input data.
- **Key considerations:**
    - Is the signal **consistently available**?
    - Can upstream data systems **change** without warning?
    - Implement **version control** and **staging mechanisms** for upstream data.
- **Best practice:** Make a **controlled copy** of upstream data and upgrade **only when tested**.

---

4. Is Your Model Part of a Feedback Loop?

- **Why it's important:** Models can **influence** the very data they learn from, often **indirectly**.
- **Types of feedback loops:**
    - **Self-reinforcing**: Model predictions become features in future predictions.
    - **Cross-model contamination**: Output from one model affects another (as in the stock example).
- **Danger:** Feedback loops can cause **data drift**, **overconfidence**, or even **systemic failure**.

1.2 Automated Machine Learning (AutoML):

Machine learning (ML) development often involves repetitive and complex tasks such as experimenting with different algorithms and tuning hyperparameters. While this manual approach might work for small or exploratory projects, it becomes time-consuming and challenging in larger projects, especially for teams without specialized ML skills like data scientists.

To address these challenges, **Automated Machine Learning (AutoML)** has emerged as a solution. AutoML refers to a set of tools and technologies designed to automate key parts of the ML workflow. This makes model building faster and more accessible, even for non-experts.

AutoML typically focuses on automating repetitive and technical aspects of the **model development cycle**, including:

- **Data engineering**
- **Feature engineering and selection**
- **Algorithm selection**
- **Hyperparameter tuning**
- **Model evaluation** using test and validation data

By automating these steps, AutoML allows users to concentrate more on understanding their data and solving real-world problems, rather than spending time on low-level implementation and experimentation.

Benefits of AutoML

- **Saves time:** Reduces the need for manual tuning and experimentation.
- **Improves model quality:** Can search extensively across algorithms and hyperparameters to find strong models.
- **Requires less specialized knowledge:** Opens up ML capabilities to non-experts.
- **Good for quick dataset evaluation:** Helps determine whether a dataset has enough signal to build a meaningful model.
- **Feature evaluation:** Offers insights into which features contribute to model performance.
- **Enforces ML best practices:** Ensures standard techniques like cross-validation, feature scaling, and regularization are properly applied.

Limitations of AutoML

- **Model quality may fall short:** A skilled practitioner can often build better models manually with enough time.
- **Lack of transparency:** The internal decision-making of AutoML tools can be difficult to understand or reproduce.
- **Inconsistent results:** Different AutoML runs can yield significantly different models due to varying search paths.
- **Limited customization:** It's hard or impossible to intervene or tweak the training process.
- **Still needs data:** While AutoML simplifies model building, high-quality data remains critical.
- **Not ideal for highly specialized tasks:** For projects requiring unique model architectures or logic, manual training is better.

Data Requirements

Like manual ML, AutoML still needs substantial data. However, some modern AutoML systems use **transfer learning** to reduce the amount of labeled data needed, especially for tasks like image classification.

When to Use AutoML

AutoML is ideal when:

- You want to quickly prototype or baseline a model.
- You lack deep ML expertise.
- You're working on common problems like classification or regression with structured data.
- Customization isn't required.

Manual model training is better when:

- Model performance is critical.
- You need full control over the architecture or training logic.
- Interpretability and reproducibility are important.

AutoML Tools and Workflow

AutoML tools simplify the machine learning (ML) workflow by automating complex and repetitive tasks. These tools fall into two categories:

Types of AutoML Tools

1. **No-Code Tools**
    - User-friendly web interfaces
    - Require no programming
    - Ideal for beginners and fast prototyping
2. **API/CLI Tools**
    - Require coding and more ML expertise
    - Offer greater control and customization
    - Suitable for advanced users or complex use cases

This module focuses on no-code tools, which are simpler to use but may be limited in flexibility compared to API/CLI-based solutions.

AutoML Workflow

1. Problem Definition

- Clearly define your ML objective (e.g., classification, regression).
- Ensure the AutoML tool supports your goal and data type.

2. Data Gathering

- Collect data in a format and location supported by your AutoML tool.
- Validate the compatibility in terms of size and data types.

3. Data Preparation

While AutoML helps, manual preparation is still necessary:

- **Label your data**: Every example must have a target label.
- **Clean and format data**: Handle missing values, outliers, etc.

- **Feature transformations**: Some tools offer automatic handling; others may need manual transformation.

Example: Converting text length into a numeric feature using LENGTH(description).

4. Model Development (No-Code AutoML)

Before training, configure the experiment:

- **Import data** and assign semantic data types to features (e.g., changing postal codes from numeric to categorical).
- **Analyze and refine data** using built-in tools to validate and clean your dataset.
- **Configure parameters**:
    - Problem type (classification, regression)
    - Target column
    - Feature columns
    - Algorithms to consider
    - Evaluation metric (e.g., accuracy, RMSE)

Training can take hours depending on data size and algorithms.

---

5. Evaluate Model

- Examine:
    - Feature importance
    - Model architecture and hyperparameters
    - Evaluation metrics and performance plots

---

6. Productionization (Deployment)

- Some AutoML tools assist in deploying the trained model into production systems, though this topic is outside the current scope.

---

7. Model Retraining

- Periodically retrain with new or refined data to improve model accuracy.

- Use insights from previous runs to enhance your dataset and reconfigure future AutoML experiments.

Conclusion:
AutoML tools—especially no-code ones—streamline the ML process, making it easier and faster to develop models without requiring deep technical expertise. However, successful use still depends on good data preparation and an understanding of the problem you're solving. For teams or individuals seeking productivity, accessibility, and simplicity in model development, AutoML can be a powerful ally.

1.3 Fairness:

Machine Learning and Bias

Machine learning (ML) models are not inherently objective. ML practitioners train models using datasets of training examples, and human involvement in providing and curating this data can introduce bias into a model's predictions.

When building models, it's important to recognize common human biases that may appear in your data so you can take proactive steps to mitigate their effects.

1. Reporting Bias

Definition:

Occurs when the frequency of events, properties, and outcomes in a dataset does not match real-world frequencies. This happens because people tend to report unusual or memorable events, assuming the ordinary does not need recording.

Example:

A sentiment-analysis model is trained on book reviews from a popular website. Most reviews in the dataset express extreme opinions (either very positive or very negative), because people are less likely to review books that didn't strongly impact them. As a result, the model struggles to interpret neutral or subtly worded reviews.

2. Historical Bias

Definition:

Occurs when historical data reflects inequities that existed at the time it was collected.

Example:

A housing dataset from the 1960s contains home-price data influenced by discriminatory lending practices active during that time.

3. Automation Bias

Definition:

The tendency to favor results generated by automated systems over those generated by humans, regardless of actual accuracy.

Example:

ML practitioners at a manufacturing company were ready to deploy a model to detect defects in sprockets. However, the factory supervisor pointed out that the model's precision and recall were both 15% lower than those of human inspectors.

4. Selection Bias

Occurs when data is selected in a way that does not reflect the true population or real-world distribution. It can take multiple forms:

4.1 Coverage Bias

Definition:

Occurs when data selection is not representative of the full population.

Example:

A model is trained to predict sales based on surveys of people who bought the product. Those who chose a competitor's product were not surveyed, leaving them unrepresented in the data.

4.2 Non-Response Bias

Definition:

Also known as participation bias. Happens when the dataset becomes unrepresentative due to uneven participation.

Example:

A model is trained using survey responses from both buyers of a product and a competitor's product. However, people who bought the competitor’s product were 80% more likely to refuse the survey, leading to underrepresentation of their data.

4.3 Sampling Bias

Definition:

Occurs when proper randomization is not used during data collection.

Example:

A model is trained using responses from the first 200 consumers who replied to an email survey. These early respondents might have stronger-than-average enthusiasm, skewing the data.

5. Group Attribution Bias

A tendency to generalize traits of individuals to their entire group. It includes two common forms:

5.1 In-Group Bias

Definition:

Preference for individuals from the same group as yourself or who share characteristics with you.

Example:

Two ML engineers building a résumé-screening model believe that applicants from the same computer science academy they attended are more qualified.

5.2 Out-Group Homogeneity Bias

Definition:

Assumes members of a different group are more alike than they actually are.

Example:

Those same engineers believe that all applicants not from their academy lack adequate skills, ignoring individual differences.

6. Implicit Bias

Definition:

Occurs when assumptions are made based on personal experiences or thinking patterns that are not universally applicable.

Example:

An ML practitioner builds a gesture-recognition model using head shaking to represent "no." However, in some cultures, a head shake actually means "yes."

7. Confirmation Bias

Definition:

Happens when model builders interpret data in a way that confirms their preexisting beliefs or hypotheses.

Example:

An ML engineer builds a model to predict dog aggressiveness. Due to a childhood encounter with a toy poodle, they associate the breed with aggression and unconsciously discard data that shows poodles as gentle.

8. Experimenter's Bias

Definition:

Occurs when a model builder keeps tweaking a model until it gives results that match their original hypothesis.

Fairness: Identifying Bias in ML Models

When preparing data and evaluating models, it's crucial to detect and address **bias** to ensure **fairness** in predictions. Key areas to watch:

1. **Missing Feature Values**:
    - Missing data can signal under-representation.
    - Especially check if missingness is linked to specific subgroups (e.g., all city dogs missing temperament).
2. **Unexpected Feature Values**:
    - Outliers or incorrect values (e.g., a 35-year-old dog) might signal data issues and potential bias.
3. **Data Skew**:
    - Over- or under-representation of certain groups (e.g., more big dogs than small dogs) can lead to biased outcomes.
    - Always evaluate model performance **by subgroup**, not just overall.

The goal is to ensure the model performs **equally well across different groups** (e.g., breed, age, size) and not just on average.

Bias Mitigation Techniques

Once bias is identified in your training data, you can reduce its impact using:

---

1. Data Augmentation

- **What**: Add more representative data (especially from underrepresented groups).
- **When**: Ideal if data is missing, incorrect, or skewed.
- **Limitations**: Might be costly, time-consuming, or restricted (privacy/legal issues).

---

2. Adjusting the Loss Function

- **What**: Modify the loss function to penalize unfair outcomes.

- **Why**: Standard losses (e.g., log loss) don't consider subgroup fairness.

Common Techniques:

- **MinDiff**: Penalizes the model if prediction distributions differ across groups.
- **CLP (Counterfactual Logit Pairing)**: Penalizes if changing a sensitive attribute (e.g., gender) changes the prediction, despite all other features being the same.

These strategies are chosen based on the model's use case and fairness goals.

Evaluating For Bias:

When evaluating **fairness** in machine learning models, especially in sensitive domains like admissions, hiring, or lending, aggregate performance metrics (like overall accuracy or precision) can be misleading. These metrics might show high performance overall but **hide poor results** for specific **subgroups**, often leading to **biased decisions** against minority or underrepresented groups.

To detect and mitigate such bias, we use **fairness metrics** that evaluate how the model performs across **different demographic groups**. The three key fairness metrics commonly used are:

- Demographic Parity
- Equality of opportunity.
- Counterfactual Fairness.

What is Demographic Parity?

Demographic Parity (also called **Statistical Parity**) is a **fairness criterion** used in machine learning to evaluate whether a model's decisions are **independent of a sensitive attribute**, such as gender, race, or age.

Definition:

A model satisfies **demographic parity** if the **outcome (positive prediction rate)** is the **same across all demographic groups**.

Example:

In a **loan approval** system:

- If **60%** of male applicants are approved
- And **60%** of female applicants are approved
- → The model satisfies **demographic parity** for gender.

But if:

- Males = 70% approval
- Females = 50% approval
  → **Demographic parity is violated**, indicating **potential bias**.

Limitations

- **Ignores qualification**: A model could achieve parity by approving the same percentage of each group, even if some groups are **more or less qualified**.
- Can result in **reverse discrimination** if not used carefully.

When to Use

Use demographic parity when:

- Your goal is **equal treatment**, not necessarily equal outcome accuracy.
- You want to ensure no group is systematically favored in **positive decisions**.

Related Metrics

Metric	Focus	Notes
Demographic Parity	Equal positive outcomes	Doesn't consider qualification
Equal Opportunity	Equal true positive rate	More fairness-aware

Metric	Focus	Notes
Predictive Parity	Equal precision across groups	Focuses on correctness of positive predictions

---

Summary

Demographic Parity ensures that a model gives **positive outcomes at equal rates** across groups, regardless of whether those groups differ in actual qualification.

Equality of Opportunity

Equality of Opportunity is a **fairness metric** used to evaluate machine learning models, especially in classification tasks. It requires that the **true positive rate (TPR)** be equal across different demographic groups.

Definition:

A model satisfies **Equality of Opportunity** if the probability of correctly predicting a positive outcome (true positive) is the same for all groups **given that the true label is positive**.

In simpler terms:

If someone **deserves** a positive prediction (e.g., admission, loan approval, hiring), the model should be **equally likely** to give it to them **regardless of their group membership** (e.g., gender, race).

Example:

Suppose your model predicts university admissions, and the true label indicates whether a student is actually qualified. If:

- 80% of **qualified male students** are admitted (true positive rate),
- then **equality of opportunity** means that **80% of qualified female students** should also be admitted.

Why it matters:

This fairness metric is especially relevant in **high-stakes decisions** where denying a deserved opportunity (false negative) is harmful — like in education, hiring, or healthcare.

Counterfactual Fairness

Counterfactual Fairness means that a model's prediction **would not change** if a person's sensitive attribute (like race, gender, or age) were changed — while keeping all else about the person the same.

Definition:

A prediction is **counterfactually fair** if **for any individual**, the model's prediction **would be the same in a counterfactual world** where only the sensitive attribute was different.

Example:

Let's say your model predicts whether a person should be approved for a loan.

- For a female applicant who gets **denied**, you generate a **counterfactual** version of her where everything stays the same — income, job, credit history — **except gender is now male**.
- If the model would approve the **male version**, the model is **not counterfactually fair**.

What it ensures:

Counterfactual fairness ensures that decisions **aren't based on attributes people can't control**, like race or gender — even **implicitly** through correlated features.

Challenge:

It requires building **causal models** to simulate counterfactual versions of individuals, which is complex and not always feasible in practice.

Comparison:

Demographic parity, equality of opportunity, and counterfactual fairness are three key metrics used to evaluate fairness in machine learning models. **Demographic parity** requires that the selection rate (e.g., being admitted to a university) is equal across different demographic groups, regardless of whether individuals are actually qualified. While this ensures equal representation, it may result in favoring less qualified individuals from one group over more qualified ones from another. **Equality of opportunity**, on the other hand, ensures that among those who are truly qualified, individuals from all groups have an equal chance of being selected. This focuses on fairness for those who deserve the positive outcome but does not address false positives or treatment of unqualified individuals. **Counterfactual fairness** takes a more individual-level approach, ensuring that a model's prediction would remain the same even if a person's sensitive attribute (like race or gender) were changed. It relies on causal reasoning to isolate the effect of the protected attribute, making it more precise but also more difficult to implement in practice. Together, these metrics offer different lenses through which model fairness can be understood and evaluated.

2. Unsupervised Machine Learning:

Unsupervised learning is a branch of **machine learning** that deals with unlabeled data. Unlike supervised learning, where the data is labeled with a specific category or outcome, unsupervised learning algorithms **are tasked with finding patterns and relationships within the data without any prior knowledge of the data's meaning**. Unsupervised machine learning algorithms **find hidden patterns and data without any human intervention, i.e., we don't give output to our model. The training model has only input parameter values and discovers the groups or patterns on its own.**

The input to the unsupervised learning models is as follows:

- **Unstructured data**: May contain noisy(meaningless) data, missing values, or unknown data
- **Unlabeled data**: Data only contains a value for input parameters, there is no targeted value(output). It is easy to collect as compared to the labeled one in the Supervised approach.

Unsupervised Learning Algorithms

There are mainly 3 types of Algorithms which are used for Unsupervised dataset.

- **Clustering**
- **Association Rule Learning**
- **Dimensionality Reduction**

1. Clustering Algorithms

Clustering in unsupervised machine learning is the process of grouping unlabeled data into clusters based on their similarities. The goal of clustering is to identify patterns and relationships in the data without any prior knowledge of the data's meaning.

Broadly this technique is applied to group data based on different patterns, such as similarities or differences, our machine model finds. These algorithms are used to process raw, unclassified data objects into groups. For example, in the above figure, we have not given output parameter values, so this technique will be used to group clients based on the input parameters provided by our data.

Some common clustering algorithms:

- ***K-means Clustering:*** *Groups data into K clusters based on how close the points are to each other.*
- ***Hierarchical Clustering:*** *Creates clusters by building a tree step-by-step, either merging or splitting groups.*
- ***Density-Based Clustering (DBSCAN):*** *Finds clusters in dense areas and treats scattered points as noise.*
- ***Mean-Shift Clustering:*** *Discovers clusters by moving points toward the most crowded areas.*
- ***Spectral Clustering:*** *Groups data by analyzing connections between points using graphs.*

2. Association Rule Learning

Association rule learning is also known as association rule mining is a common technique used to discover associations in unsupervised machine learning. This technique is a rule-based ML technique that finds out some very useful relations between parameters of a large data set. This technique is basically used for market basket analysis that helps to better understand the relationship between different products.

For e.g. shopping stores use algorithms based on this technique to find out the relationship between the sale of one product w.r.t to another's sales based on customer behavior. **Like if a customer buys milk, then he may also buy bread, eggs, or butter**. Once trained well, such models can be used to increase their sales by planning different offers.

Some common Association Rule Learning algorithms:

- ***Apriori Algorithm:*** *Finds patterns by exploring frequent item combinations step-by-step.*
- ***FP-Growth Algorithm:*** *An Efficient Alternative to Apriori. It quickly identifies frequent patterns without generating candidate sets.*
- ***Eclat Algorithm:*** *Uses intersections of itemsets to efficiently find frequent patterns.*
- ***Efficient Tree-based Algorithms:*** *Scales to handle large datasets by organizing data in tree structures.*

3. Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of features in a dataset while preserving as much information as possible. This technique is useful for improving the performance of machine learning algorithms and for data visualization.

Imagine a dataset of 100 features about students (height, weight, grades, etc.). To focus on key traits, you reduce it to just 2 features: height and grades, making it easier to visualize or analyze the data.

Here are some popular **Dimensionality Reduction algorithms**:

- **Principal Component Analysis (PCA):** Reduces dimensions by transforming data into uncorrelated principal components.
- **Linear Discriminant Analysis (LDA):** Reduces dimensions while maximizing class separability for classification tasks.
- **Non-negative Matrix Factorization (NMF):** Breaks data into non-negative parts to simplify representation.
- **Locally Linear Embedding (LLE):** Reduces dimensions while preserving the relationships between nearby points.
- **Isomap:** Captures global data structure by preserving distances along a manifold.

Challenges of Unsupervised Learning

Here are the key challenges of unsupervised learning:

- **Noisy Data**: Outliers and noise can distort patterns and reduce the effectiveness of algorithms.
- **Assumption Dependence**: Algorithms often rely on assumptions (e.g., cluster shapes), which may not match the actual data structure.
- **Overfitting Risk**: Overfitting can occur when models capture noise instead of meaningful patterns in the data.
- **Limited Guidance**: The absence of labels restricts the ability to guide the algorithm toward specific outcomes.
- **Cluster Interpretability**: Results, such as clusters, may lack clear meaning or alignment with real-world categories.
- **Sensitivity to Parameters**: Many algorithms require careful tuning of hyperparameters, such as the number of clusters in k-means.
- **Lack of Ground Truth**: Unsupervised learning lacks labeled data, making it difficult to evaluate the accuracy of results.

Applications of Unsupervised learning

Unsupervised learning has diverse applications across industries and domains. Key applications include:

- **Customer Segmentation:** Algorithms cluster customers based on purchasing behavior or demographics, enabling targeted marketing strategies.
- **Anomaly Detection:** Identifies unusual patterns in data, aiding fraud detection, cybersecurity, and equipment failure prevention.
- **Recommendation Systems**: Suggests products, movies, or music by analyzing user behavior and preferences.
- **Image and Text Clustering**: Groups similar images or documents for tasks like organization, classification, or content recommendation.
- **Social Network Analysis**: Detects communities or trends in user interactions on social media platforms.
- **Astronomy and Climate Science:** Classifies galaxies or groups weather patterns to support scientific research

K means Clustering

K-Means Clustering is an Unsupervised Machine Learning algorithm which groups unlabeled dataset into different clusters. It is used to organize data into **groups based on their similarity**.

For example online store uses K-Means to group customers based on purchase frequency and spending creating segments like Budget Shoppers, Frequent Buyers and Big Spenders for personalised marketing.

The algorithm works by first randomly picking some central points called **centroids** and each data point is then assigned to the closest centroid forming a cluster. After all the points are assigned to a cluster the centroids are updated by finding the average position of the points in each cluster. This process repeats until the centroids stop changing forming clusters. The goal of clustering is to divide the data points into clusters so that similar data points belong to same group.

How k-means clustering works?

We are given a data set of items with certain features and values for these features like a vector. The task is to categorize those items into groups. To achieve this we will use the K-means algorithm. 'K' in the name of the algorithm represents the number of groups/clusters we want to classify our items into.

The algorithm will categorize the items into k groups or clusters of similarity. To calculate that similarity we will use the Euclidean distance as a measurement. The algorithm works as follows:

1. First we randomly initialize k points called means or cluster centroids.
2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that cluster so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

Selecting the right number of clusters is important for meaningful segmentation to do this we use Elbow Method for optimal value of k in KMeans which is a graphical tool used to determine the optimal number of clusters (k) in K-means.

Final Analogy Summary:

Think of the entire process like:

1. **Scattering dots** on a board.
2. Dropping **magnets** randomly.
3. Each dot goes to its nearest magnet.
4. Each magnet moves to the **average location** of its attracted dots.
5. Repeat until magnets **settle down** (not shown here, but needed for full K-means).
6. Plot the final "teams" of dots and their leaders.

Hierarchical Clustering

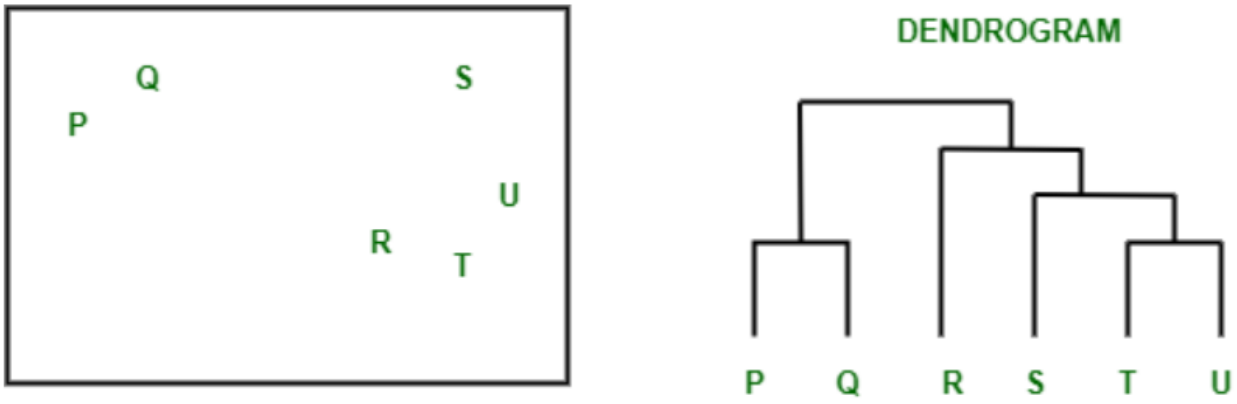
Hierarchical clustering is used to group similar data points together based on their similarity creating a **hierarchy or tree-like structure**. The key idea is to begin with each data point as its own separate cluster and then progressively merge or split them based on their similarity.

*Imagine you have four fruits with different weights: an **apple (100g)**, **a banana (120g)**, **a cherry (50g) and a grape (30g)**. Hierarchical clustering starts by treating each **fruit as its own group**.*

- It then merges the closest groups based on their weights.
- First the cherry and grape are grouped together because they are the lightest.
- Next the apple and banana are grouped together.

Dendogram

A **dendrogram** is like a family tree for clusters. It shows how individual data points or groups of data merge together. The bottom shows each data point as its own group, and as you move up, similar groups are combined. The lower the merge point, the more similar the groups are. It helps you see how things are grouped step by step. The working of the dendrogram can be explained using the below diagram:



In the above image on the left side there are five points labeled P, Q, R, S and T. These represent individual data points that are being clustered. On the right side there's a **dendrogram** which show how these points are grouped together step by step.

- At the bottom of the dendrogram the points P, Q, R, S and T are all separate.
- As you move up, the closest points are merged into a single group.
- The lines connecting the points show how they are progressively merged based on similarity.
- The height at which they are connected shows how similar the points are to each other; the shorter the line the more similar they are

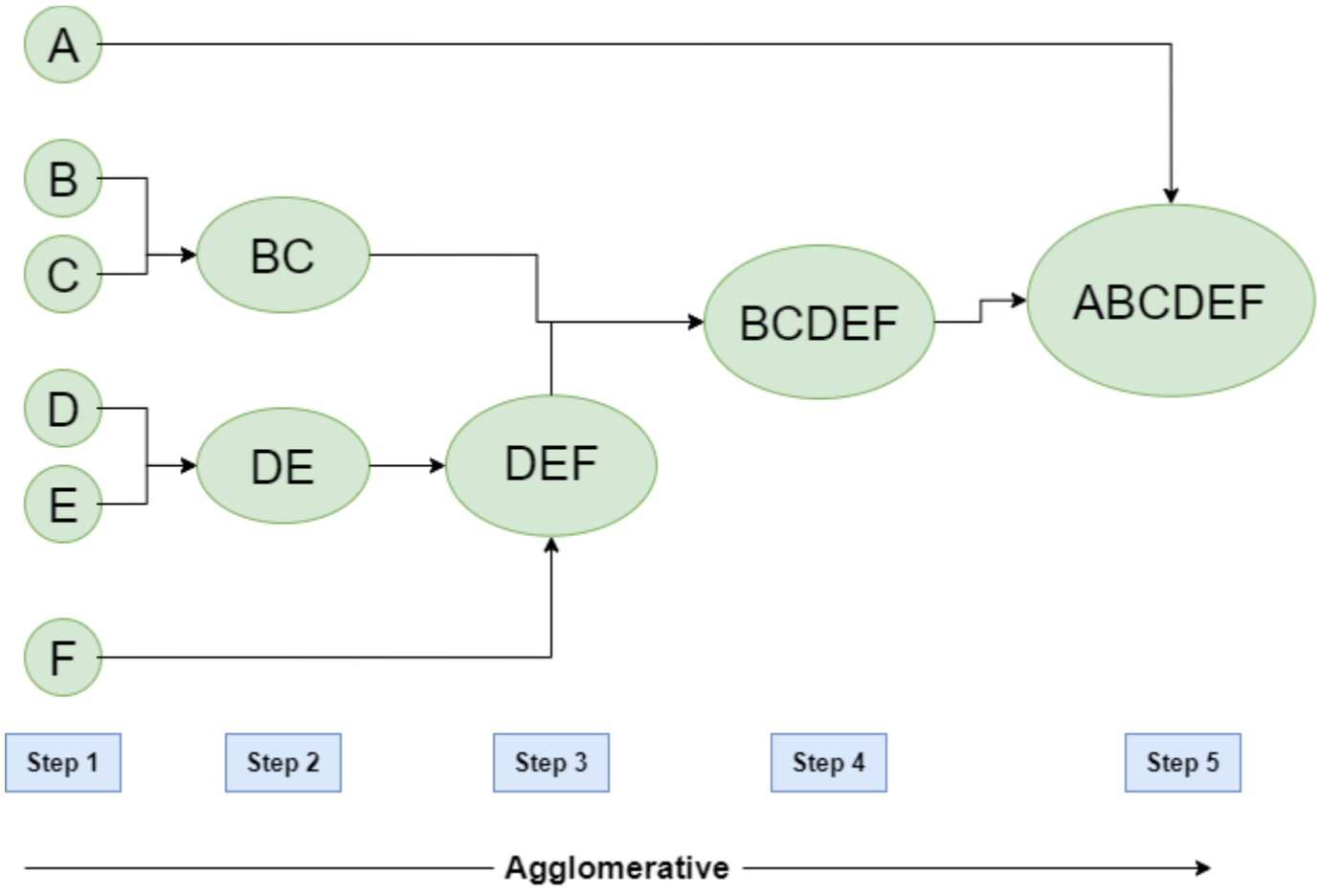
Types of Hierarchical Clustering

Now we understand the basics of hierarchical clustering. There are two main types of hierarchical clustering.

1. Agglomerative Clustering
2. Divisive clustering

Hierarchical Agglomerative Clustering

It is also known as the **bottom-up approach** or **hierarchical agglomerative clustering (HAC)**. Unlike flat clustering hierarchical clustering provides a structured way to group data. This clustering algorithm does not require us to prespecify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerate pairs of clusters until all clusters have been merged into a single cluster that contains all data.



Hierarchical Agglomerative Clustering

Workflow for Hierarchical Agglomerative clustering

1. **Start with individual points**: Each data point is its own cluster. For example if you have 5 data points you start with 5 clusters each containing just one data point.
2. **Calculate distances between clusters**: Calculate the distance between every pair of clusters. Initially since each cluster has one point this is the distance between the two data points.
3. **Merge the closest clusters**: Identify the two clusters with the smallest distance and merge them into a single cluster.
4. **Update distance matrix**: After merging you now have one less cluster. Recalculate the distances between the new cluster and the remaining clusters.
5. **Repeat steps 3 and 4**: Keep merging the closest clusters and updating the distance matrix until you have only one cluster left.
6. **Create a dendrogram**: As the process continues you can visualize the merging of clusters using a tree-like diagram called a **dendrogram**. It shows the hierarchy of how clusters are merged.

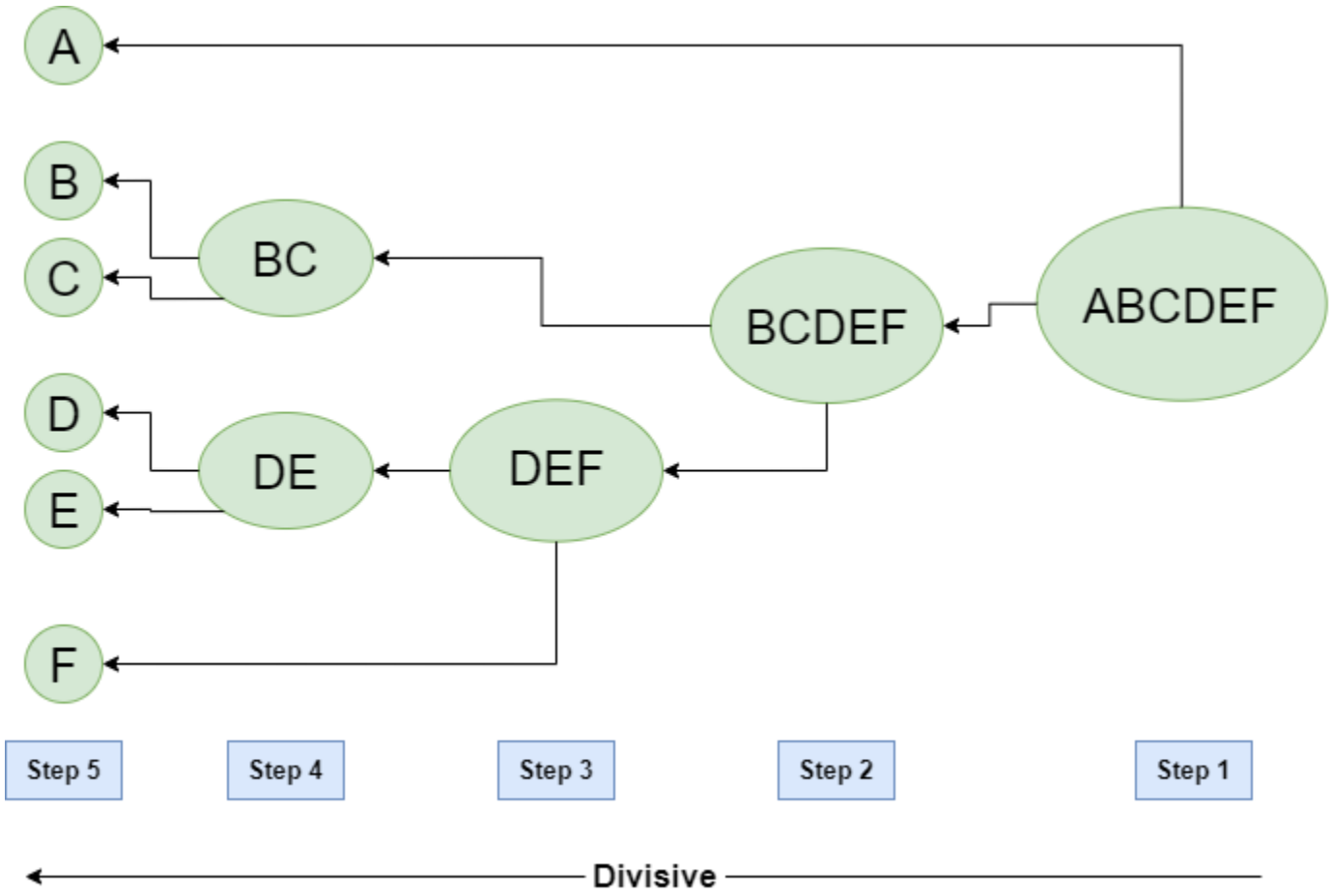
Hierarchical Divisive clustering

It is also known as a **top-down approach**. This algorithm also does not require to prespecify the number of clusters. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been split into singleton clusters.

Workflow for Hierarchical Divisive clustering :

1. **Start with all data points in one cluster**: Treat the entire dataset as a single large cluster.
2. **Split the cluster**: Divide the cluster into two smaller clusters. The division is typically done by finding the two most dissimilar points in the cluster and using them to separate the data into two parts.
3. **Repeat the process**: For each of the new clusters, repeat the splitting process:
    1. Choose the cluster with the most dissimilar points.
    2. Split it again into two smaller clusters.

4. **Stop when each data point is in its own cluster**: Continue this process until every data point is its own cluster, or the stopping condition (such as a predefined number of clusters) is met.



Hierarchical Divisive clustering

Computing Distance Matrix

While merging two clusters we **check the distance between two every pair of clusters and merge the pair with the least distance/most similarity**. But the question is how is that distance determined. There are different ways of defining Inter Cluster distance/similarity. Some of them are:

1. **Min Distance:** Find the minimum distance between any two points of the cluster.
2. **Max Distance:** Find the maximum distance between any two points of the cluster.
3. **Group Average:** Find the average distance between every two points of the clusters.
4. **Ward's Method:** The similarity of two clusters is based on the increase in squared error when two clusters are merged.

DBSCAN Clustering in ML - Density based clustering

DBSCAN is a density-based clustering algorithm that groups data points that are closely packed together and marks outliers as noise based on their density in the feature space. It identifies clusters as dense regions in the data space separated by areas of lower density. Unlike K-Means or hierarchical clustering which assumes clusters are compact and spherical, DBSCAN perform well in handling real-world data irregularities such as:

- **Arbitrary-Shaped Clusters**: Clusters can take any shape not just circular or convex.
- **Noise and Outliers**: It effectively identifies and handles noise points without assigning them to any cluster.

Key Parameters in DBSCAN

1. eps: This defines the radius of the neighborhood around a data point. If the distance between two points is less than or equal to eps they are considered neighbors. A common method to determine eps is by analyzing the k-distance graph. Choosing the right eps is important:

- If eps is too small most points will be classified as noise.
- If eps is too large clusters may merge and the algorithm may fail to distinguish between them.

2. MinPts: This is the minimum number of points required within the **eps** radius to form a dense region. A general rule of thumb is to set MinPts >= D+1 where **D** is the number of dimensions in the dataset.

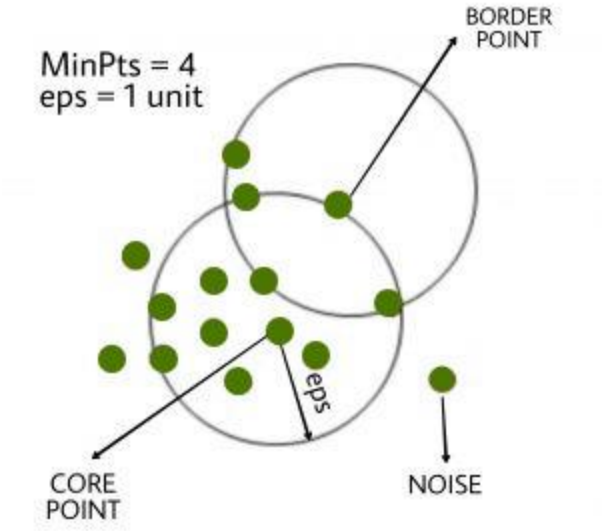
*For most cases a minimum value of **MinPts = 3** is recommended.*

How Does DBSCAN Work?

DBSCAN works by categorizing data points into three types:

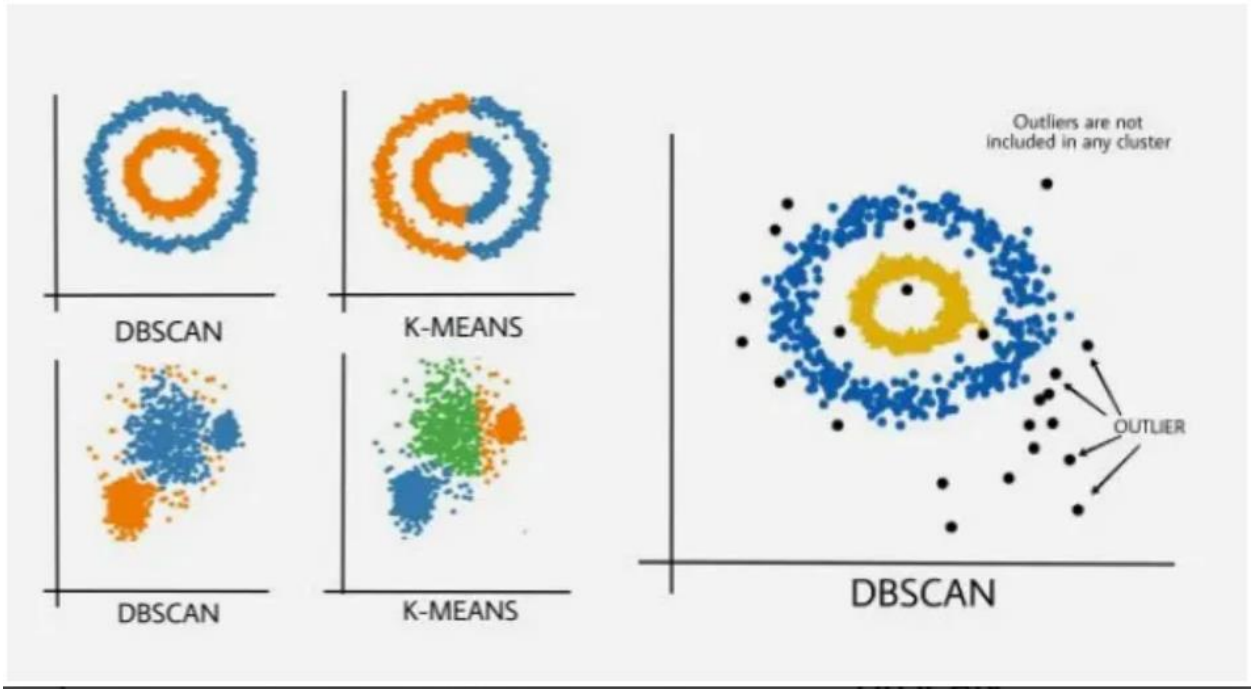
1. Core points which have a sufficient number of neighbors within a specified radius (eplison)
2. Border points which are near core points but lack enough neighbors to be core points themselves
3. Noise points which do not belong to any cluster.

By iteratively expanding clusters from core points and connecting density-reachable points, DBSCAN forms clusters without relying on rigid assumptions about their shape or size.



Steps in the DBSCAN Algorithm

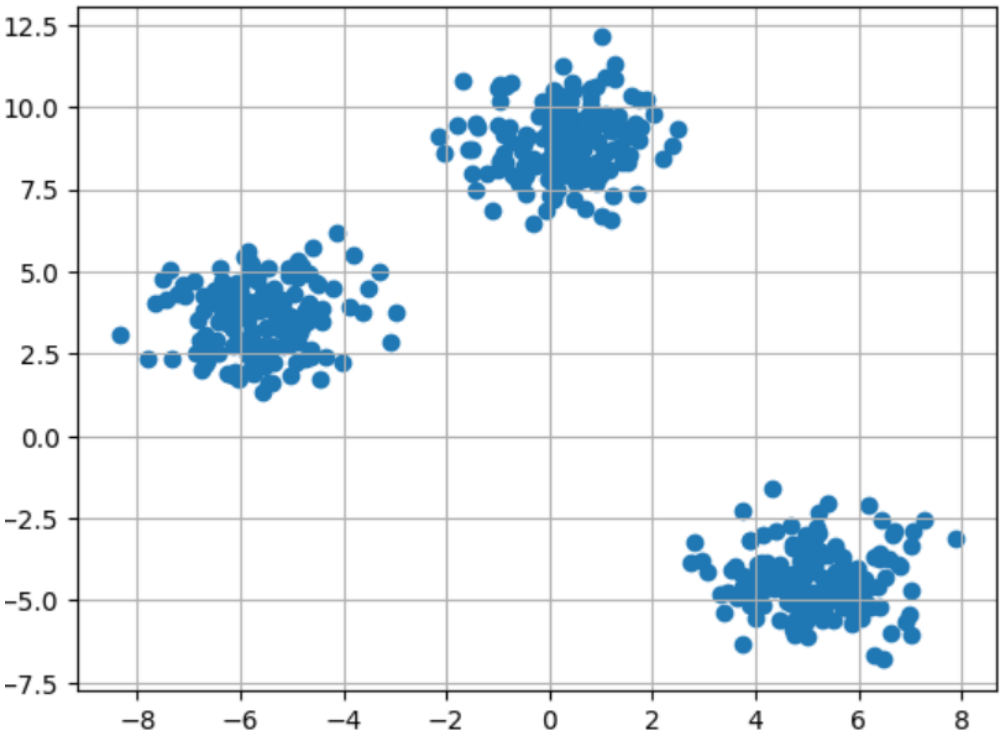
1. **Identify Core Points**: For each point in the dataset count the number of points within its eps neighborhood. If the count meets or exceeds MinPts mark the point as a core point.
2. **Form Clusters**: For each core point that is not already assigned to a cluster create a new cluster. Recursively find all density-connected points i.e points within the eps radius of the core point and add them to the cluster.
3. **Density Connectivity**: Two points a and b are density-connected if there exists a chain of points where each point is within the eps radius of the next and at least one point in the chain is a core point. This chaining process ensures that all points in a cluster are connected through a series of dense regions.
4. **Label Noise Points**: After processing all points any point that does not belong to a cluster is labeled as noise.


DBSCAN
K-MEANS
DBSCAN
K-MEANS
Outliers are not included in any cluster
OUTLIER
DBSCAN

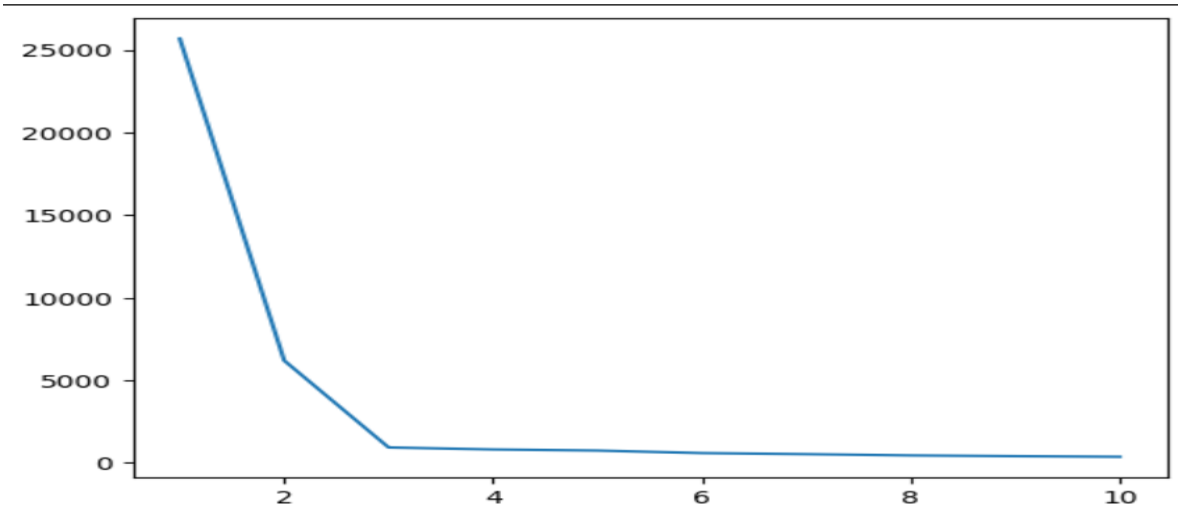
3. Practical Learning:

KMeansClustering:

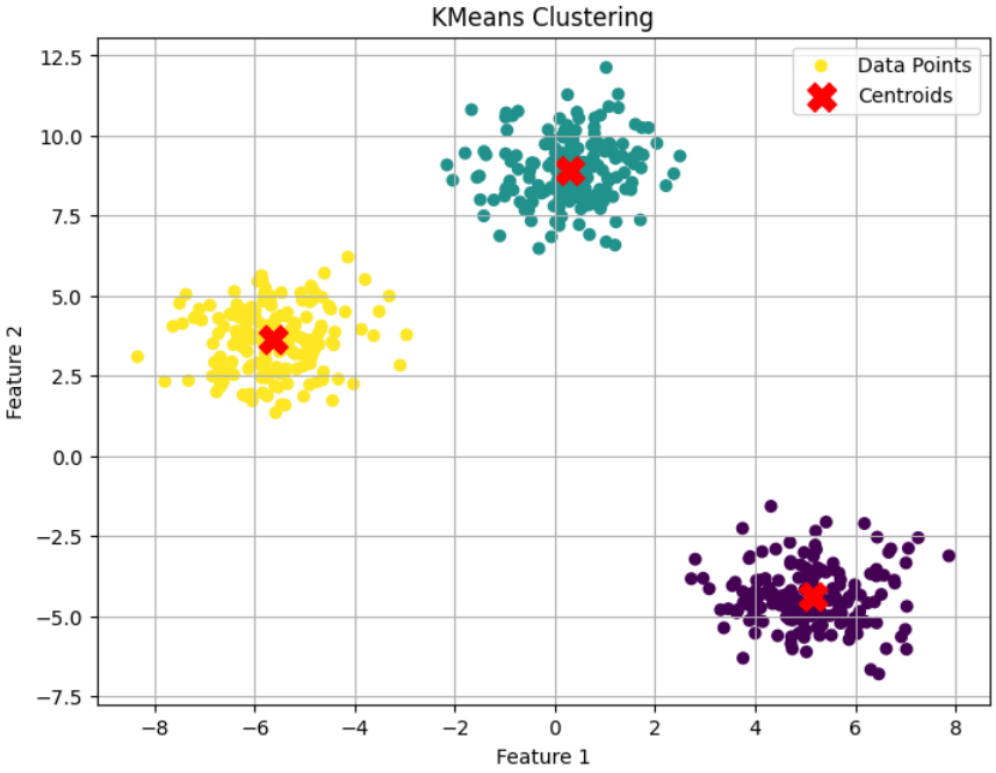
Below Images clearly demonstrate how this algorithm makes clusters in a given dataset.



Finding Number of clusters using elbow method:

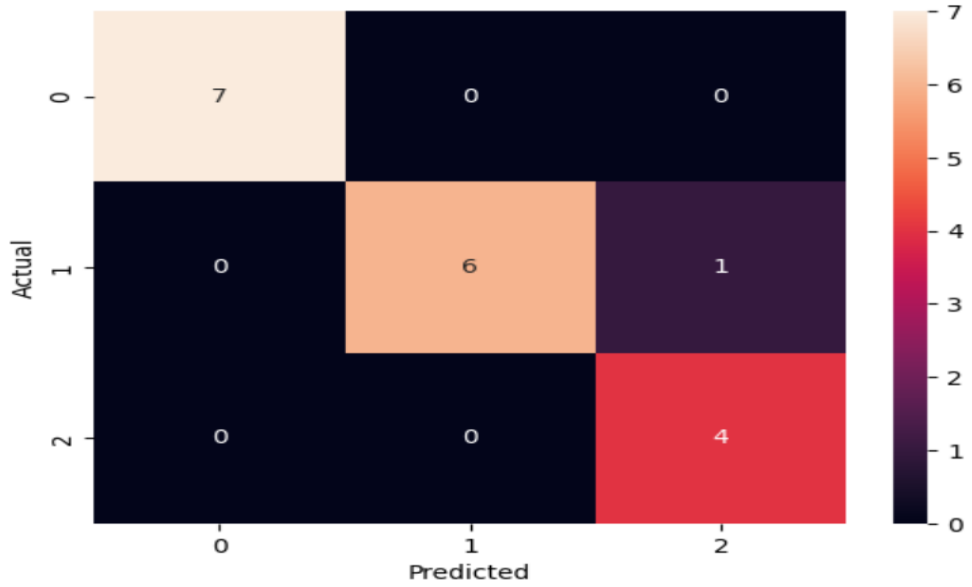


After applying KMeans:



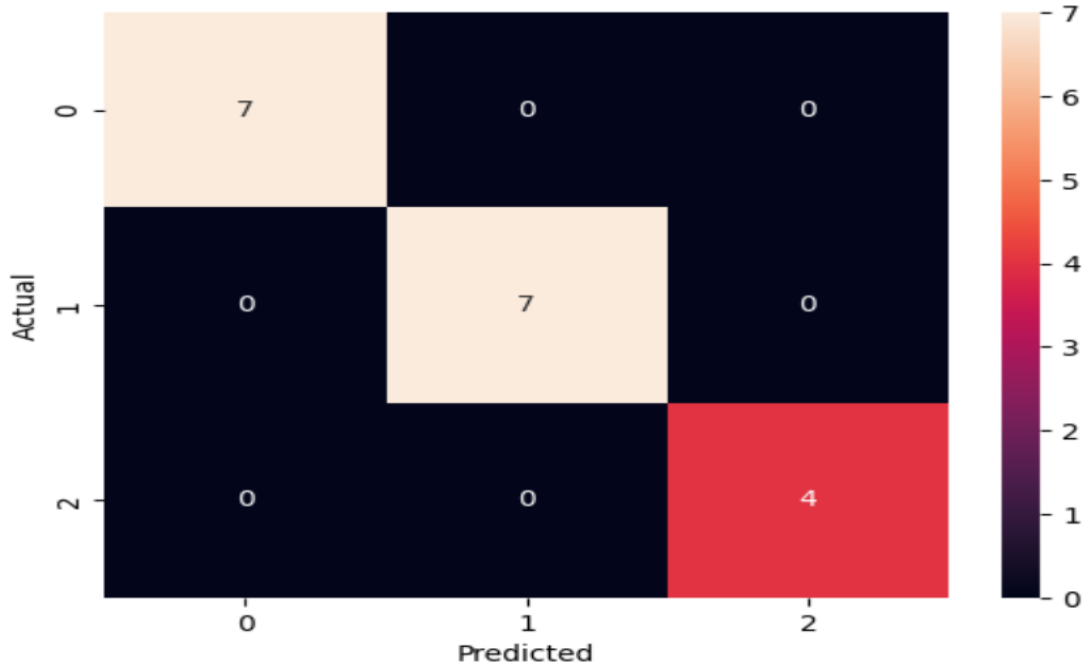
Principal Component Analysis (PCA):

Without PCA and using all the given 13 features of the Wine dataset:



	precision	recall	f1-score	support
0	1.00	1.00	1.00	7
1	1.00	0.86	0.92	7
2	0.80	1.00	0.89	4
accuracy			0.94	18
macro avg	0.93	0.95	0.94	18
weighted avg	0.96	0.94	0.95	18

With PCA and using only 2 features instead of 13:



	precision	recall	f1-score	support
0	1.00	1.00	1.00	7
1	1.00	0.86	0.92	7
2	0.80	1.00	0.89	4
accuracy			0.94	18
macro avg	0.93	0.95	0.94	18
weighted avg	0.96	0.94	0.95	18

Accuracy with number of components 1: 0.72

Accuracy with number of components 2: 1.0

Accuracy with number of components 3: 1.0

Accuracy with number of components 4: 0.94

Accuracy with number of components 5: 0.94

Accuracy with number of components 6: 0.94

Accuracy with number of components 7: 0.94

Accuracy with number of components 8: 0.94

Accuracy with number of components 9: 0.94

Accuracy with number of components 10: 0.94

Accuracy with number of components 11: 0.94

Accuracy with number of components 12: 0.94

4. References:

- https://scikit-learn.org/stable/
- https://developers.google.com/machine-learning/crash-course/
- https://www.youtube.com/
- https://www.youtube.com/@campusx-official