A Short Note on

Machine Learning

September 2024

The Machine Learning Landscape

What Is Machine Learning?

- Machine Learning is the science (and art) of programming computers so they can learn from data.
- Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed. **—Arthur Samuel, 1959**
- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. **— Tom Mitchell, 1997**

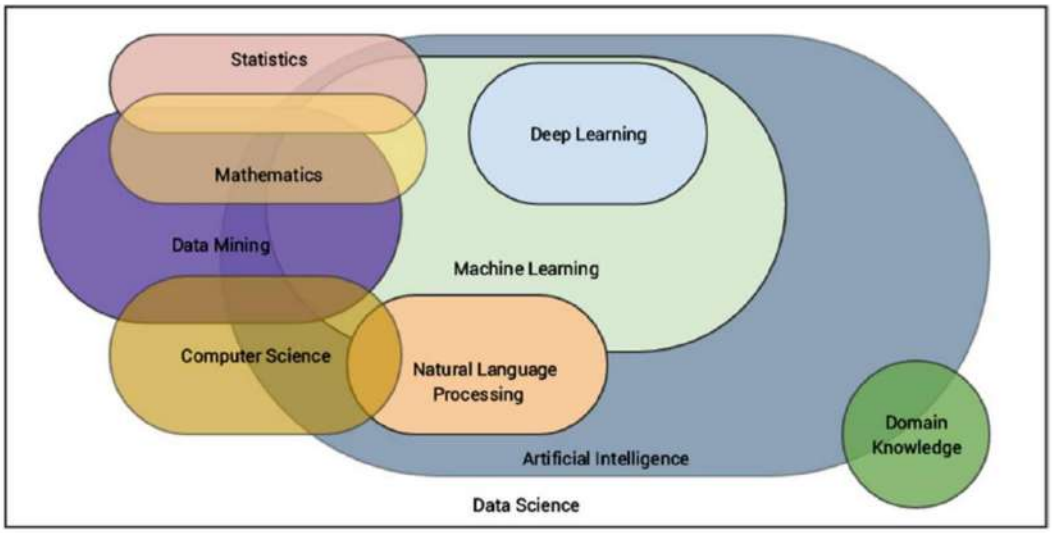


Figure 1-4. *Machine Learning: a true multi-disciplinary field*

- Example: Email SPAM Filtering
  - The task T is to flag spam for new emails
  - the experience E is the training data, and
  - the performance measure P needs to be defined:
    - the ratio of correctly classified emails (Accuracy)

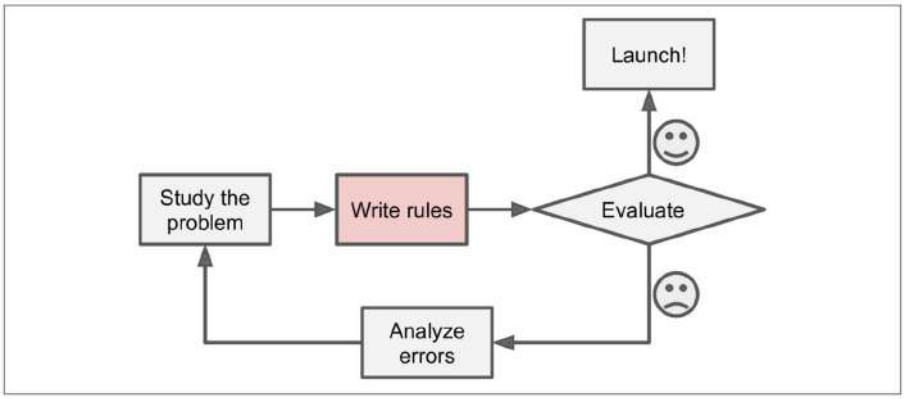


Figure 1-1. The traditional approach

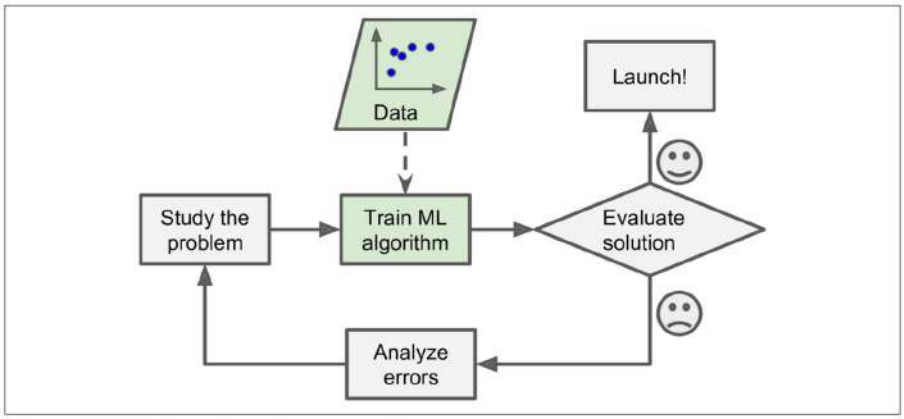


Figure 1-2. Machine Learning approach

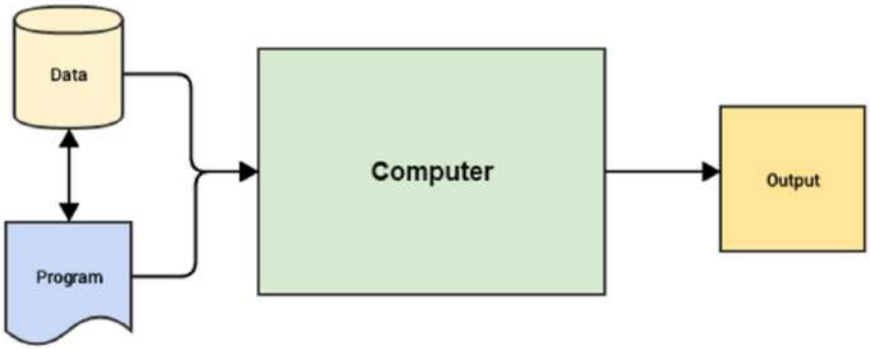


Figure 1-1. *Traditional programming paradigm*

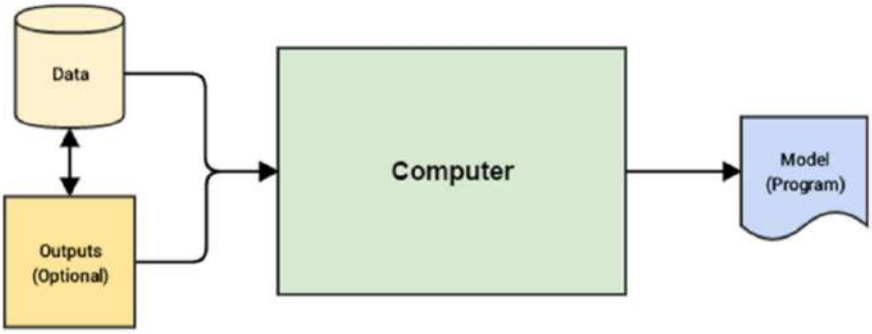
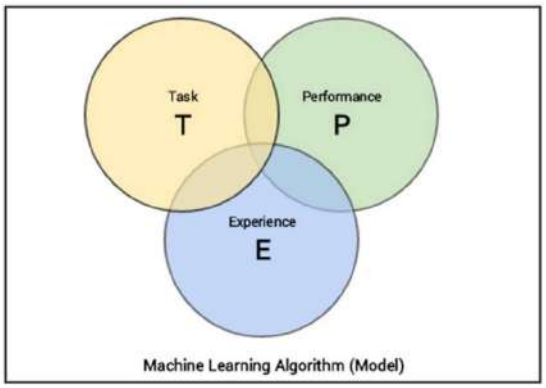


Figure 1-2. *Machine Learning paradigm*



We can simplify the definition as follows.
Machine Learning is a field that consists of learning algorithms that:

- Improve their performance P
- At executing some task T
- Over time with experience E

Machine Learning is great for:

- Problems for which existing solutions require a lot of hand-tuning or long lists of rules: one Machine Learning algorithm can often simplify code and perform better.
- Complex problems for which there is no good solution at all using a traditional approach: the best Machine Learning techniques can find a solution.
- Fluctuating environments: a Machine Learning system can adapt to new data.
- Getting insights about complex problems and large amounts of data.

Types of features in Machine Learning

In machine learning, features (or attributes/variables) are the inputs used to train a model. They can be of various types, each playing a different role in the learning process. Here are the main types of features.

Feature Type	Description	Examples
Numerical Features	Continuous or discrete numbers that can be measured on a scale.	Age, height, weight, income, temperature
Categorical Features	Discrete values that can be grouped into categories.	Gender, color, zip code, brand
Binary Features	A special case of categorical features where the data takes on one of two possible values.	Yes/No, True/False, 0/1
Text Features	Features derived from textual data, often requiring preprocessing like tokenization or embedding.	Reviews, comments, articles
Time-based Features	Features that represent temporal data, often used in time series analysis.	Timestamps, dates, time intervals
Interaction Features	Features created by combining two or more features to capture interactions between them.	Age * Income, Price * Quantity

Derived Features	Features created from existing features through transformations or aggregations.	Log-transformed values, moving averages, polynomial features
Image Features	Features extracted from images, often using techniques like convolutional neural networks (CNNs).	Pixel values, edges, textures
Audio Features	Features extracted from audio signals, often using techniques like Fourier transforms.	Frequency, amplitude, pitch, MFCC (Mel-frequency cepstral coefficients)
Geospatial Features	Features that represent spatial data, often used in geographic information systems (GIS).	Latitude, longitude, altitude, distance

1. Numerical Features

- Continuous: These are numerical values that can take any value within a range. Examples include age, temperature, and salary.
- Discrete: These are numerical values that take distinct, separate values. Examples include the number of children in a family or the number of cars in a garage.

2. Categorical Features

- Nominal: These are categories with no intrinsic ordering. Examples include gender, marital status, or colors (red, blue, green).
- Ordinal: These are categories with a meaningful order but without consistent differences between categories. Examples include education level (high school, bachelor's, master's) or rating scales (poor, fair, good, excellent).

3. Binary Features

These are a special case of categorical features with only two possible values, often represented as 0 and 1. Examples include true/false, yes/no, or male/female.

4. Text Features

These are features based on textual data. Text features often require preprocessing and transformation, such as tokenization, stemming, and converting to numerical vectors using techniques like TF-IDF or word embeddings.

5. Date and Time Features

These features involve time-related data, such as timestamps, dates, and durations. They often need to be converted into meaningful components like day, month, year, hour, minute, or calculated intervals.

6. Derived or Engineered Features

These are new features created from the existing ones to provide more information to the model. Examples include ratios, interactions between features, or polynomial transformations.

7. Spatial Features

These features involve geographic or spatial data. Examples include latitude and longitude, distance between locations, or spatial patterns.

8. Sequence and Time Series Features

These features involve sequential data points collected over time. Examples include stock prices over time, sensor readings, or historical weather data.

9. Image Features

These features are extracted from image data. Techniques such as convolutional neural networks (CNNs) are used to automatically detect and extract relevant features from images.

10. Audio Features

These features are derived from audio signals. Examples include frequency components, pitch, and spectrograms. Techniques like Mel-frequency cepstral coefficients (MFCCs) are commonly used.

11. Interaction Features

These features are created by combining multiple features to capture interactions between them. For example, creating a feature that is the product of two numerical features.

12. Aggregated Features

These features are aggregated statistics from groups of data points. Examples include average purchase amount per customer, total sales per region, or count of transactions per day.

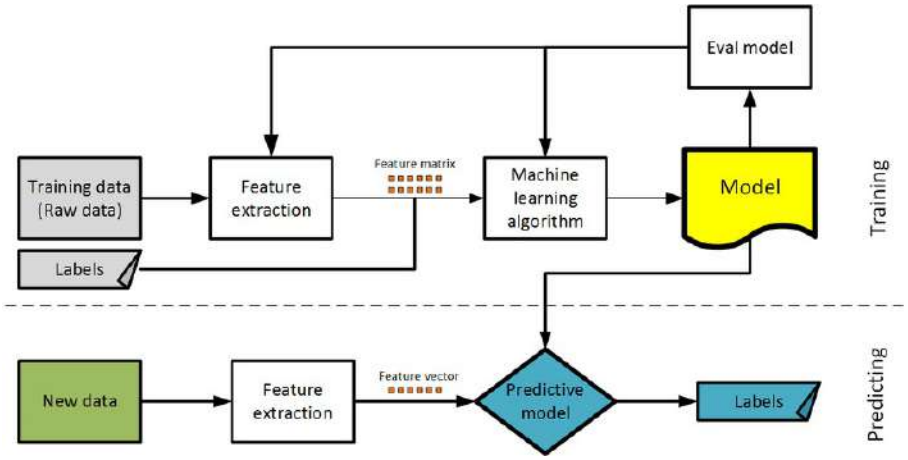
Types of Machine Learning Systems

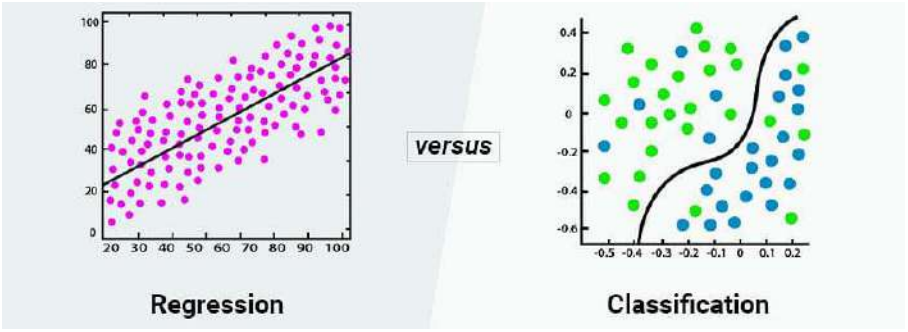
Category	Type	Description	Examples
By Learning Approach	Supervised Learning	Trained with labelled data to map inputs to outputs.	Linear Regression, Decision Trees, SVM, Neural Networks
	Unsupervised Learning	Given input data without labelled responses; finds patterns or groupings.	K-means Clustering, PCA, Association Rule Learning
	Semi-Supervised Learning	Combines labelled and unlabelled data for training.	Semi-Supervised SVM
	Reinforcement Learning	Learns through trial and error by receiving feedback in the form of rewards/penalties.	Q-Learning, Deep Q-Networks (DQN), AlphaGo
By Type of Output	Regression	Predicts continuous values.	Predicting house prices
	Classification	Predicts discrete categories or classes.	Spam Detection
	Clustering	Identifies groups or clusters within data.	Customer Segmentation
	Anomaly Detection	Identifies unusual data points that don't fit the general pattern.	Fraud Detection
	Recommendation Systems	Suggests products or content based on user behaviour.	Netflix or Amazon Recommendations
By Training Method	Batch Learning	Trained on the entire dataset at once; doesn't learn further unless retrained.	Static Data Models
	Online Learning	Trained incrementally with data instances sequentially; adapts to changing data.	Dynamic, Real-time Systems
By Similarity to Human Brain	Artificial Neural Networks (ANN)	Layers of interconnected nodes inspired by the human brain.	Feedforward Neural Networks, CNN, RNN
	Deep Learning	Deep neural networks with multiple layers to extract high-level features.	Image Recognition with CNNs, NLP with Transformers
By Application	Predictive Analytics	Uses historical data to predict future outcomes.	Time Series Forecasting, Predictive Maintenance
	Computer Vision	Enables machines to interpret visual information.	Image Classification, Object Detection, Facial Recognition
	Natural Language Processing (NLP)	Enables machines to understand and generate human language.	Chatbots, Sentiment Analysis, Machine Translation
Other Models	**Ensemble Learning Models**	Combine multiple models to improve overall performance.	Bagging, Boosting, Stacking
	Transfer Learning Models	Models leverage knowledge gained from one task to improve learning in a different but related task.	Fine-Tuning Pre-Trained Models like ResNet, VGG, BERT.

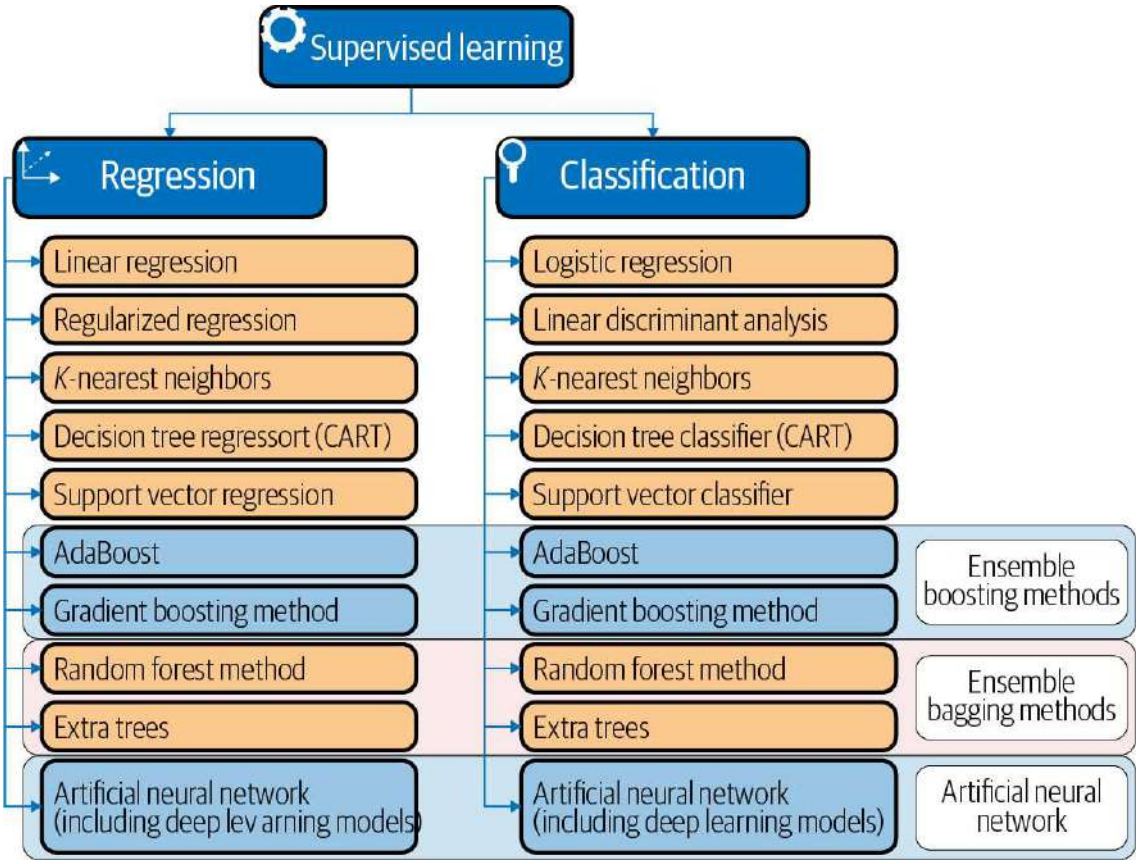
There are so many different types of Machine Learning systems that it is useful to classify them in broad categories based on:

Whether or not they are trained with human supervision

- **Supervised Learning**
  - In supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels.
  - Two types of supervised learning
    - Classification or categorization
      - This typically encompasses the list of problems or tasks where the machine has to take in data points or samples and assign a specific class or category to each sample.
    - Regression
      - These types of tasks usually involve performing a prediction such that a real numerical value is the output instead of a class or category for an input data point.



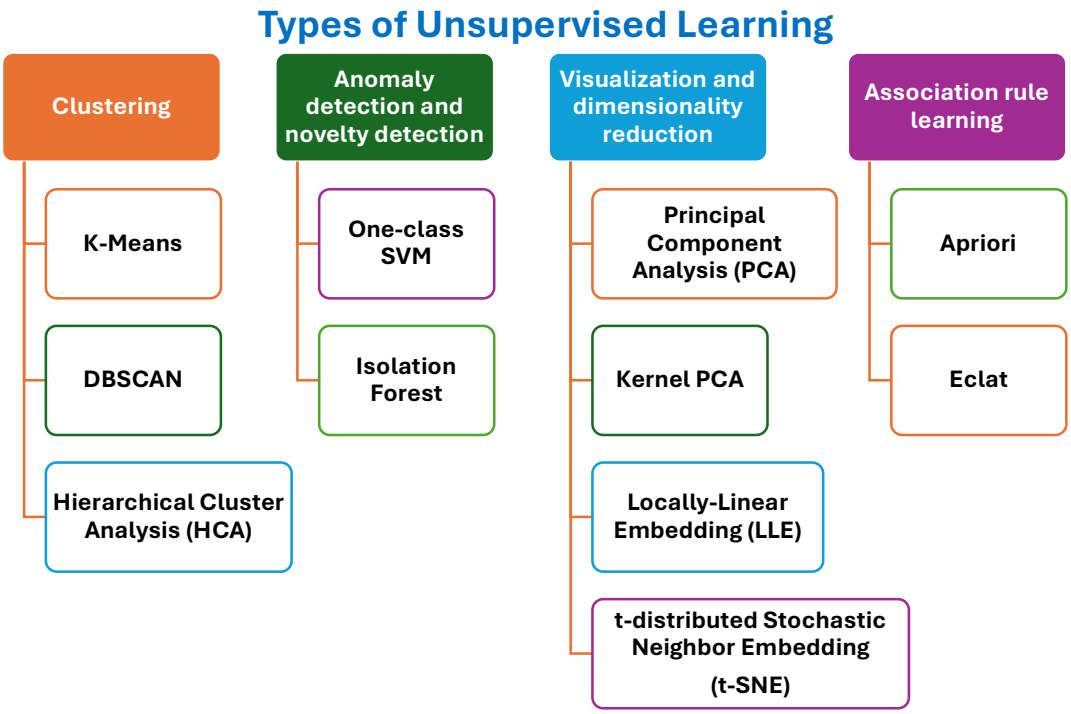




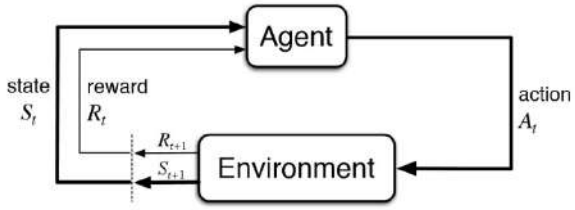
Key Concepts in Supervised Learning

Key Concepts in Supervised Learning
•**Features** • input variables used to make predictions. •**Labels** • output variables or target values •**Training Set** • portion of the data used to train the model •**Test Set** • portion of the data used to evaluate the model •**Model** • mathematical representation that maps inputs to outputs •**Prediction** • mathematical representation that maps inputs to outputs

- **Unsupervised Learning**
  - In unsupervised learning, as you might guess, the training data is unlabelled. The system tries to learn without a teacher.



- **Semi-supervised Learning**
  - Some algorithms can deal with partially labelled training data, usually a lot of unlabelled data and a little bit of labelled data. This is called semi-supervised learning.
- **Reinforcement Learning**
  - The learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards)
  - It must then learn by itself what is the best strategy, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.



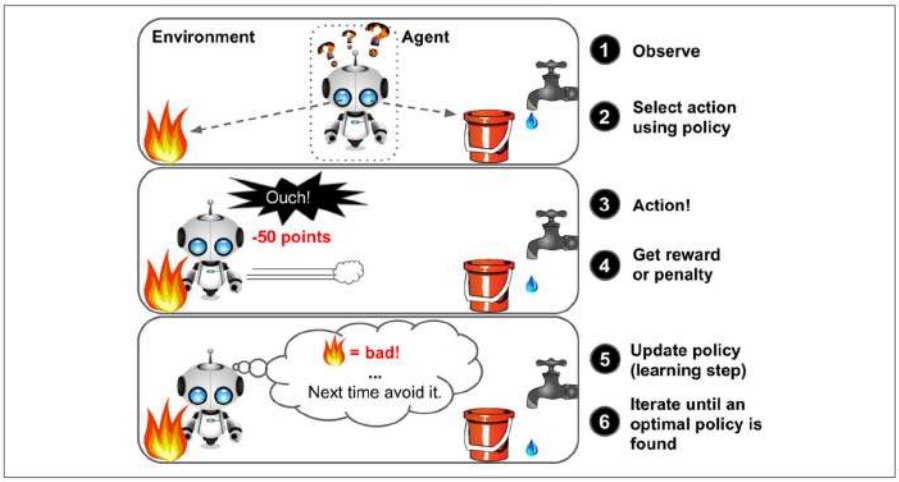


Figure 1-12. Reinforcement Learning

Whether or not they can learn incrementally on the fly

- **Online learning**
  - In online learning, you train the system incrementally by feeding it data instances sequentially, either individually or by small groups called mini batches.
  - Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives.
  - Online learning is great for systems that receive data as a continuous flow (e.g., stock prices) and need to adapt to change rapidly or autonomously.
  - It is also a good option if you have limited computing resources: once an online learning system has learned about new data instances, it does not need them anymore, so you can discard them.
- **Batch learning**
  - In batch learning, the system is incapable of learning incrementally: it must be trained using all the available data. This will generally take a lot of time and computing resources, so it is typically done offline. This is called offline learning.

Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model, much like scientists do

- **Instance-based learning**
  - The system learns the examples by heart, then generalizes to new cases by comparing them to the learned examples (or a subset of them), using a similarity measure like distance or correlation.
  - Advantages of instance-based learning:
    - No explicit model training is required, so it's easy to adapt to new data.
    - Can handle complex relationships and adapt well to varying data distributions.
  - Disadvantages of instance-based learning:
    - Can be computationally expensive, especially with large datasets.
    - Sensitive to irrelevant or noisy features in the data.
    - Lack of a learned model makes it harder to interpret or explain predictions.

- **Model-based learning**
    - Another way to generalize from a set of examples is to build a model of these examples, then use that model to make predictions. This is called model-based learning.
    - Model-based learning involves creating a model that generalizes from the training data to make predictions on new, unseen data. This model can capture patterns, relationships, and features within the data to provide a way to understand and predict outcomes.

Prerequisites and Tools for Machine Learning

To begin with machine learning, it's essential to have a foundational understanding and access to key tools:

1. Mathematics:

- **Linear Algebra**: Understanding vectors and matrices is crucial for many ML algorithms.
- **Probability and Statistics**: Helps in interpreting data and building probabilistic models.
- **Calculus**: Useful for optimization techniques, particularly in deep learning.

2. Programming Skills:

- **Python**: The most popular language for ML due to its simplicity and extensive libraries like **Scikit-learn**, **TensorFlow**, and **PyTorch**.
- **R**: Often used for statistical analysis and data visualization.
- Familiarity with **SQL** for database management and retrieving data.

3. Data Handling Tools:

- **Pandas**: For data manipulation and analysis.
- **NumPy**: For handling large datasets and performing mathematical operations efficiently.

4. Development Environments:

- **Jupyter Notebook**: A widely-used tool for coding and visualizing results interactively.
- **Google Colab**: Provides a cloud-based environment with free GPU access for running ML models.

5. Machine Learning Libraries and Frameworks:

- **Scikit-learn**: Ideal for beginners to practice standard algorithms like regression and clustering.
- **TensorFlow and PyTorch**: Popular frameworks for building deep learning models.

6. Version Control:

- **Git**: Essential for managing code, tracking changes, and collaborating with others.

7. Cloud Platforms:

- **AWS, Azure, Kaggle and Google Cloud**: Offer scalable resources and tools like AutoML for deploying and managing ML models efficiently.

Category	Details
Mathematics	• **Linear Algebra**: Understanding vectors and matrices is crucial for many ML algorithms. • **Probability and Statistics**: Helps in interpreting data and building probabilistic models. • **Calculus**: Useful for optimization techniques, particularly in deep learning.
Programming Skills	• **Python**: Popular for ML with libraries like Scikit-learn, TensorFlow, and PyTorch. • **R**: Used for statistical analysis and data visualization. • **SQL**: Familiarity with SQL for database management and data retrieval.

Data Handling Tools	• **Pandas**: For data manipulation and analysis. • **NumPy**: For handling large datasets and performing mathematical operations efficiently.
Development Environments	• **Jupyter Notebook**: Tool for coding and visualizing results interactively. • **Kaggle, Google Colab**: Cloud-based environment with free GPU access for running ML models.
ML Libraries and Frameworks	• **Scikit-learn**: Ideal for beginners to practice standard algorithms like regression and clustering. • **TensorFlow and PyTorch**: Frameworks for building deep learning models.
Version Control	• **Git**: For managing code, tracking changes, and collaborating.
Cloud Platforms	• **AWS, Azure, Google Cloud**: Scalable resources and tools like AutoML for ML model deployment.

Main Challenges of Machine Learning

Main Challenges of Machine Learning	Inadequate Training Data
	Poor Quality of Data
	Non-Representative Training Data
	Overfitting and Underfitting
	Monitoring and Maintenance
	Data Bias
	Lack of Explainability
	Lack of Skilled Resources
	Process Complexity of Machine Learning
	Slow Implementations and Results
	Irrelevant Features
	Getting Bad Recommendations

1. Inadequate Training Data

One of the primary challenges in machine learning is the availability of **adequate training data**. Machine learning models require large amounts of high-quality data to learn effectively. However, in many domains, obtaining such data is difficult due to factors like **privacy concerns**, **costs of data collection**, and **data sparsity**.

When the training dataset is too small, models can struggle to capture meaningful patterns, resulting in poor performance on unseen data. This problem becomes particularly pronounced in fields like healthcare, where collecting large, diverse datasets is challenging.

Solutions:

- **Data Augmentation**: Techniques such as data augmentation, which artificially increases the size of the dataset by modifying existing data, can help mitigate the problem of limited data.
- **Synthetic Data Generation**: Tools like **GANs (Generative Adversarial Networks)** can generate synthetic data to expand training datasets.
- **Transfer Learning**: Transfer learning allows models to leverage knowledge from other related tasks, reducing the need for large amounts of data.

Addressing the challenge of inadequate training data is essential for building robust and accurate machine learning models.

2. Poor Quality of Data

The quality of data directly impacts the performance of machine learning models. **Poor-quality data**, which may be incomplete, noisy, or inconsistent, can lead to inaccurate predictions and flawed outcomes. **Data preprocessing** is a crucial step to ensure that data is clean and ready for analysis.

Common Issues in Data Quality:

- **Missing Values**: Gaps in data can cause models to make incorrect predictions.
- **Outliers**: Extreme values can skew the model's understanding of normal behavior.
- **Noisy Data**: Unreliable or incorrect data points can reduce the accuracy of the model.

Best Practices for Data Quality:

- **Data Cleaning**: Techniques like **imputation** (filling missing values) and **outlier detection** are essential for improving data quality.
- **Normalization and Scaling**: Ensuring that data is on a consistent scale can improve the model's ability to learn patterns.
- **Feature Engineering**: Creating new features from existing data can provide the model with more meaningful information.

Ensuring high-quality data through proper preprocessing steps is key to improving model performance.

3. Non-Representative Training Data

Non-representative training data occurs when the training dataset does not accurately reflect the **real-world distribution** of data. This can result in models that perform well on the training data but fail to generalize to new, unseen data.

Consequences:

- **Poor Generalization**: Models trained on biased or unrepresentative data may perform well in controlled environments but poorly in real-world applications.
- **Bias in Predictions**: If the training data is not representative, the model's predictions will be biased toward certain outcomes, potentially leading to unfair or inaccurate results.

Solutions:

- **Data Sampling**: Use **stratified sampling** techniques to ensure the training dataset accurately reflects the distribution of the target population.
- **Cross-Validation**: Employ cross-validation methods to test the model's generalization capabilities across different subsets of the data.

Addressing non-representative data is essential for ensuring that models can make accurate predictions in real-world scenarios.

4. Overfitting and Underfitting

Overfitting occurs when a machine learning model becomes too complex and fits the noise in the training data rather than the underlying patterns. This results in poor generalization to new data. **Underfitting**, on the other hand, occurs when a model is too simple to capture the underlying patterns in the data.

Causes:

- **Overfitting**: Caused by models with too many parameters or when there is insufficient regularization.
- **Underfitting**: Occurs when the model is too simple or lacks the capacity to capture complex patterns.

Strategies to Address Overfitting and Underfitting:

- **Cross-Validation**: Regularly test models on unseen data during training to prevent overfitting.
- **Regularization Techniques**: Methods like **L1** and **L2 regularization** can prevent the model from becoming too complex.
- **Early Stopping**: Stop the training process when the model's performance on a validation set starts to degrade, preventing overfitting.

Balancing model complexity is essential to avoid both overfitting and underfitting, ensuring optimal model performance.

5. Monitoring and Maintenance

Once a machine learning model is deployed, **continuous monitoring** is essential to ensure that it remains accurate and relevant. As the data landscape changes, models may begin to drift from their original performance levels.

Challenges:

- **Model Drift**: Over time, changes in the data distribution can lead to model performance degradation, a phenomenon known as model drift.

- **Retraining Needs**: Models require periodic updates and retraining to ensure they continue to deliver accurate predictions as new data becomes available.

Solutions:

- **Automated Monitoring**: Implement monitoring systems to detect when a model's performance starts to decline.
- **Scheduled Retraining**: Regularly retrain models using new data to keep them up to date.

Effective monitoring and maintenance strategies are critical for ensuring that machine learning models remain accurate over time.

6. Data Bias

Data bias occurs when the training data used to build a model is not representative of the broader population, leading to biased predictions. This can result in models that **discriminate against certain groups** or fail to generalize to all users.

Examples:

- **Gender Bias in Hiring Models**: Algorithms trained on biased hiring data may favour one gender over another, perpetuating inequalities.
- **Facial Recognition**: Systems trained predominantly on lighter-skinned individuals often fail to accurately identify people with darker skin tones.

Detecting and Reducing Bias:

- **Bias Detection Tools**: Tools like **IBM AI Fairness 360** can help identify and reduce bias in machine learning models.
- **Diverse Training Data**: Ensuring that the training dataset includes diverse examples can help mitigate bias.

Addressing data bias is critical for building **fair and equitable machine learning models**, especially in industries like healthcare, finance, and criminal justice.

7. Lack of Explainability

Many machine learning models, especially **deep learning** models, are often described as "**black boxes**" due to the difficulty in understanding how they make decisions. This **lack of explainability** presents challenges in industries where transparency is crucial, such as **healthcare** and **finance**.

Consequences:

- **Regulatory Compliance**: In some industries, regulations require that models provide clear explanations for their decisions. Lack of explainability can hinder the adoption of machine learning in these fields.
- **Trust**: Without understanding how a model arrives at a decision, stakeholders may be reluctant to trust its predictions.

Methods to Improve Explainability:

- **LIME (Local Interpretable Model-agnostic Explanations)**: LIME explains individual predictions by approximating the model locally.
- **SHAP (SHapley Additive exPlanations)**: SHAP values provide insights into how each feature contributes to a prediction.

Improving explainability is essential for increasing trust in machine learning models and ensuring compliance with industry regulations.

8. Lack of Skilled Resources

The demand for skilled machine learning professionals far exceeds the available supply, creating a **skills gap** that slows the adoption of machine learning technologies.

Impact:

- **Delayed Adoption**: Organizations may struggle to implement machine learning solutions due to a lack of qualified personnel.

- **Increased Costs**: The scarcity of skilled professionals drives up salaries, making it costly for organizations to hire and retain talent.

Solutions:

- **Education and Training**: Companies can invest in training programs and partnerships with universities to upskill their current workforce.
- **Collaborations**: Partnering with **data science institutes** and offering internships can help build a pipeline of talent.

Closing the skills gap is crucial for accelerating the adoption of machine learning technologies across industries.

9. Process Complexity of Machine Learning

The **development and deployment** of machine learning models can be complex, requiring expertise in **data preprocessing**, **model selection**, and **hyperparameter tuning**. Scaling these processes for larger datasets or diverse use cases adds to the challenge.

Challenges:

- **Data Preparation**: Preprocessing large, complex datasets requires significant time and effort.
- **Model Scaling**: Adapting models to handle larger datasets or real-time applications can be difficult.

Solutions:

- **Automated Machine Learning (AutoML)**: AutoML platforms automate many of the tasks involved in building machine learning models, reducing the complexity of the process.
- **Pipeline Automation**: Automating data pipelines can streamline the process of moving from data collection to model deployment.

Simplifying the machine learning workflow through automation tools can help overcome the complexity of the process.

10. Slow Implementations and Results

Implementing machine learning models and obtaining actionable results can be a slow process, particularly for complex algorithms or large datasets.

Causes:

- **Data Processing Delays**: Preprocessing large datasets can take significant time.
- **Complexity of Algorithms**: Models like **deep learning** often require large amounts of computational resources, leading to delays.

Solutions:

- **Parallel Computing**: Using distributed computing frameworks like **Apache Spark** can speed up data processing and model training.
- **Simplified Models**: In some cases, simpler models can deliver faster results without sacrificing accuracy.

Streamlining the model-building process and optimizing algorithms for efficiency can help reduce the time it takes to implement machine learning solutions.

11. Irrelevant Features

Irrelevant or redundant features in the training data can negatively impact model performance. These features add noise, increase computational costs, and may lead to **overfitting**.

Solutions:

- **Feature Selection**: Techniques like **Principal Component Analysis (PCA)** and **Lasso regression** help reduce the number of features by selecting the most relevant ones.
- **Domain Knowledge**: Leveraging domain expertise can help identify which features are likely to be relevant and which can be discarded.

Reducing irrelevant features improves model accuracy and efficiency, leading to better results and lower computational costs.

12. Getting Bad Recommendations

Recommendation systems are widely used in platforms like **e-commerce** and **streaming services**. However, these systems can provide **bad recommendations** due to **data inaccuracies**, **user behavior changes**, or poorly designed algorithms.

Consequences:

- **User Dissatisfaction**: Poor recommendations can lead to a negative user experience, reducing engagement and customer retention.
- **Loss of Revenue**: Inaccurate recommendations can impact business outcomes by driving users away from the platform.

Solutions:

- **Collaborative Filtering**: Collaborative filtering techniques analyze user behavior to provide more personalized recommendations.
- **Reinforcement Learning**: Reinforcement learning allows recommendation systems to adapt and improve over time by learning from user feedback.

Improving recommendation systems with advanced algorithms can enhance user experience and drive better business outcomes.

Challenge	Description	Solutions
Inadequate Training Data	Difficulty in obtaining large amounts of high-quality data. Models may struggle to capture patterns with small datasets.	• Data Augmentation • Synthetic Data Generation (GANs) • Transfer Learning
Poor Quality of Data	Incomplete, noisy, or inconsistent data impacts model performance.	• Data Cleaning (imputation, outlier detection) • Normalization and Scaling • Feature Engineering
Non-Representative Training Data	Training data not reflecting real-world distribution leads to poor generalization and biased predictions.	• Data Sampling (stratified) • Cross-Validation
Overfitting and Underfitting	Overfitting: Model fits noise, poor generalization. Underfitting: Model too simple, can't capture patterns.	• Cross-Validation • Regularization (L1, L2) • Early Stopping
Monitoring and Maintenance	Models need continuous monitoring and retraining as data changes over time to avoid performance degradation (model drift).	• Automated Monitoring • Scheduled Retraining
Data Bias	Biased training data leads to discriminatory predictions or failure to generalize.	• Bias Detection Tools (e.g., IBM AI Fairness 360) • Diverse Training Data
Lack of Explainability	Difficulty in understanding complex models' decisions, challenging for industries requiring transparency (e.g., healthcare).	• - LIME (Local Interpretable Model-agnostic Explanations)

Challenge	Description	Solutions
		• SHAP (SHapley Additive exPlanations)
Lack of Skilled Resources	High demand for skilled ML professionals creates a skills gap, slowing adoption and increasing costs.	• - Education and Training • Collaborations with academic and research institutions
Process Complexity of ML	Developing and deploying models requires expertise in data preprocessing, model selection, and hyperparameter tuning. Scaling adds complexity.	• Automated Machine Learning (AutoML) • Pipeline Automation
Slow Implementations and Results	Complex algorithms and large datasets can slow down implementation and result generation.	• Parallel Computing (e.g., Apache Spark) • Simplified Models
Irrelevant Features	Redundant or irrelevant features in data add noise, increase costs, and can cause overfitting.	• Feature Selection (PCA, Lasso regression) • Domain Knowledge
Getting Bad Recommendations	Poor recommendations impact user experience and revenue due to data inaccuracies, user behaviour changes, or poorly designed algorithms.	• Collaborative Filtering • Reinforcement Learning

End-to-End Machine Learning Project

The Machine Learning Pipeline

A machine learning pipeline is a set of repeatable, linked, and often automated steps you follow to engineer, train, and deploy ML models to production.

Data collection:

In this initial stage, new data is collected from various data sources, such as databases, APIs or files. This data ingestion often involves raw data which may require preprocessing to be useful. Here are a few places you can look to get data:

- **Popular open data repositories:**
  - UC Irvine Machine Learning Repository
  - Kaggle datasets
  - Amazon's AWS datasets
- **Meta portals (they list open data repositories):**
  - http://dataportals.org/
  - http://opendatamonitor.eu/
  - http://quandl.com/
- **Other pages listing many popular open data repositories:**
  - Wikipedia's list of Machine Learning datasets
  - Quora.com question
  - Datasets subreddit

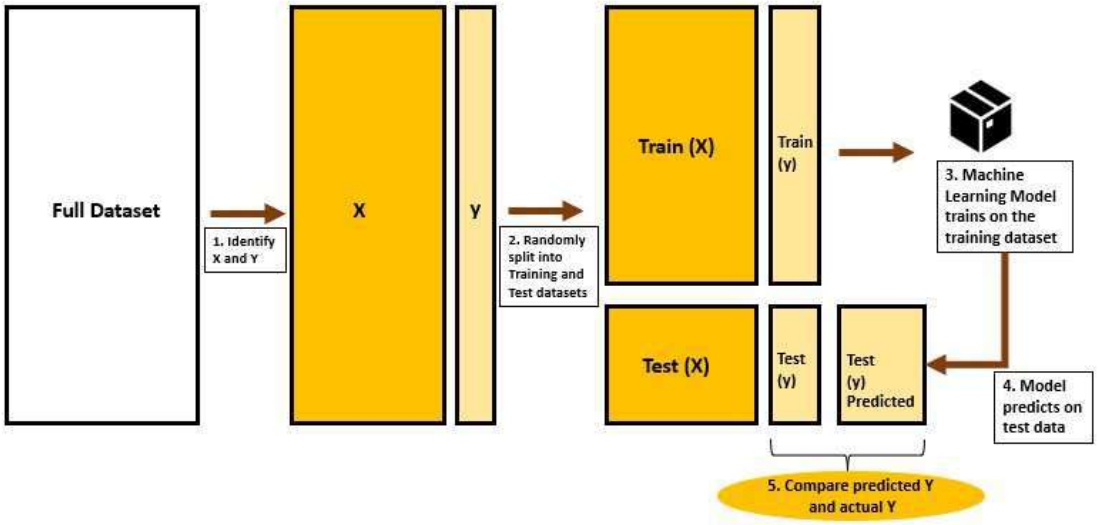
Data preprocessing:

This stage involves cleaning, transforming and preparing input data for modelling. Common preprocessing steps include

- **Exploratory Data Analysis**
  1. **Univariate Analysis**
     - **Summary Statistics**: Mean, median, mode, variance, standard deviation, and percentiles.
     - **Visualizations**: Histograms, bar plots, box plots, and density plots.
  2. **Bivariate Analysis**
     - **Comparative Statistics**: Correlation coefficients, covariance.
     - **Visualizations**: Scatter plots, pair plots, and joint plots.
  3. **Multivariate Analysis**
     - Techniques: Principal Component Analysis (PCA), clustering (e.g., k-means), and factor analysis.
     - Visualizations: Heatmaps, pair plots, and parallel coordinates plots.
  4. **Distribution Analysis**
     - Purpose: Understanding the distribution of data and identifying outliers.
     - Visualizations: Histograms, box plots, and QQ plots.

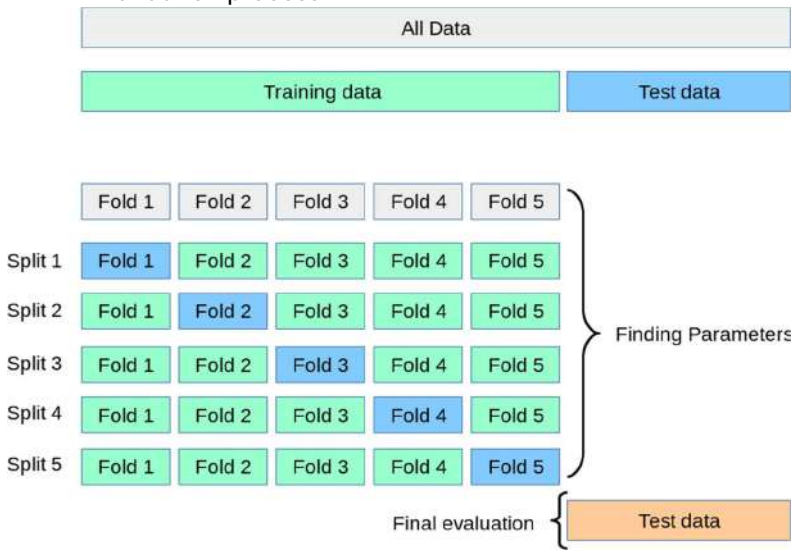
5. **Missing Value Analysis**
    - **Purpose**: Identifying and handling missing data.
    - **Techniques**: Imputation methods, visualization of missing data patterns.
6. **Outlier Detection**
    - **Purpose**: Identifying and analysing outliers.
    - **Techniques**: Z-scores, IQR (Interquartile Range) method, and visualizations like box plots.
7. **Correlation Analysis**
    - **Purpose**: Measuring the relationship between variables.
    - **Techniques**: Pearson, Spearman, and Kendall correlation coefficients.
    - **Visualizations**: Correlation heatmaps, scatter plots.
8. **Trend Analysis**
    - **Purpose**: Identifying patterns or trends over time.
    - **Techniques**: Time series analysis, moving averages.
    - **Visualizations**: Line plots, area plots.
9. **Data Aggregation and Grouping**
    - **Purpose**: Summarizing data based on categories or groups.
    - **Techniques**: Group by operations, pivot tables.
    - **Visualizations**: Bar plots, pie charts, box plots.
10. **Data Visualization**
    - **Purpose**: Creating graphical representations to understand data.
    - **Tools**: Matplotlib, Seaborn, Plotly, and other visualization libraries.
    - **Visualizations**: Various plots (scatter, line, bar, etc.), interactive dashboards.

- **Encoding Categorical Variables**
    - — Encoding categorical variables is a crucial step in preparing data for machine learning models. Since most algorithms require numerical input, we need to convert categorical data into a numerical format. Here are some common techniques:
        - One-Hot Encoding
        - Label Encoding
        - Ordinal Encoding
        - Binary Encoding
        - Target Encoding
        - Frequency Encoding
- **Scaling Numerical Features**
    - — Scaling numerical features is a key step in data preprocessing that can significantly improve the performance of machine learning models. It is scaling the data to be analysed to a specific range such as [0.0, 1.0] to provide better results. Here's a list of the most common techniques for scaling numerical features:
        - Standardization (Z-score Normalization)
        - Min-Max Scaling (Normalization)
        - Robust Scaling
        - Log Transformation
        - Square Root Transformation
        - Exponential Transformation
        - Maximum Absolute Scaling
- **Splitting The Data into Training and Testing Sets.**
    - — Splitting the data into training and testing sets is a fundamental step in building machine learning models. It ensures that the model's performance can be



evaluated on unseen data, providing a measure of its generalizability. Some of the data splitting are given as

- **Simple Split**
  A simple and common approach is to randomly split the dataset into two parts: the training set and the testing set. The training set is used to train the model, while the testing set is used to evaluate its performance.
- **Stratified Split**
  When dealing with imbalanced datasets, it's important to ensure that the split maintains the same proportion of classes in both the training and testing sets. This is known as stratified sampling.
- **Cross-Validation Split**
  Cross-validation is a more robust method that involves splitting the data into multiple folds. The model is trained and evaluated multiple times, each time using a different fold as the testing set and the remaining folds as the training set. This helps in getting a more reliable estimate of the model's performance. The diagram below shows the k-fold cross validation process



Feature engineering:

Feature engineering is the process of creating new features or selecting relevant features from the data that can improve the model's predictive power. This step often requires domain knowledge and creativity.

1. **Creation of New Features**
   - **Combining Features:** Creating new features by combining existing ones. For example, if you have height and weight, you might create a new feature BMI (Body Mass Index).
   - **Decomposition:** Splitting a feature into multiple components. For instance, a date feature can be decomposed into day, month, year, or even day of the week.
2. **Transformation of Features**
   - **Normalization/Standardization:** Scaling features so that they have a mean of 0 and a standard deviation of 1. This is especially useful for algorithms like gradient descent.
   - **Encoding Categorical Features:** Converting categorical variables into numeric values. Common methods include one-hot encoding and label encoding.
4. **Feature Selection**
   - **Filter Methods:** Using statistical tests to select features based on their relationship with the target variable.
   - **Wrapper Methods:** Employing algorithms to search for the best subset of features.
   - **Embedded Methods:** Methods like Lasso regression that perform feature selection during the model training process.
5. **Feature Interaction**
   - **Polynomial Features:** Creating new features by taking the polynomial combination of existing features.
   - **Interaction Terms:** Multiplying or combining features to capture interactions between them.
6. **Dimensionality Reduction**
   - **Principal Component Analysis (PCA):** Reducing the dimensionality of the data while retaining most of the variance.
   - **t-Distributed Stochastic Neighbour Embedding (t-SNE):** A technique for visualizing high-dimensional data by reducing it to two or three dimensions.

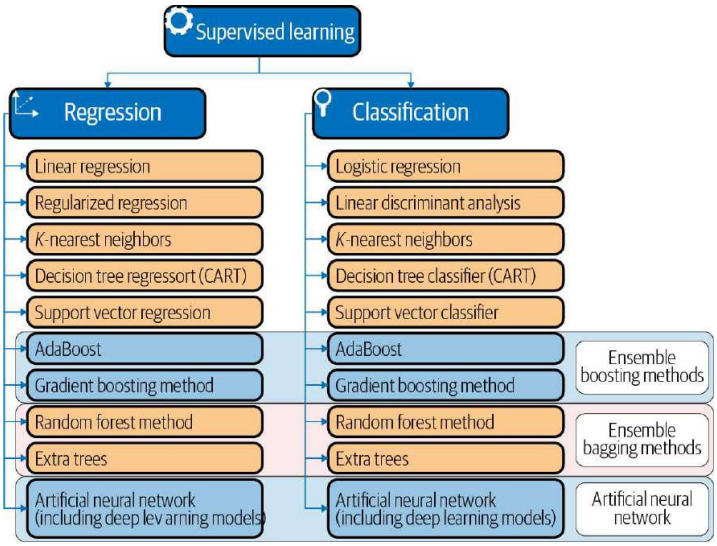
Model selection:

In this stage, you choose the appropriate machine learning algorithm(s) based on the problem type, data characteristics, and performance requirements. You may also consider hyperparameter tuning.

Model training:

The selected model(s) are trained on the training dataset using the chosen algorithm(s). This involves learning the underlying patterns and relationships within the training data.

- **Supervised learning:** When input data along with correct output is supplied to the model, the learning is known as supervised learning.
- **Unsupervised learning:** When unlabelled data is provided and the aim of the algorithm is to find patterns in data and cluster them or to find the association, this sort of learning is known as unsupervised learning.
- **Reinforcement learning:** It's basic aim is to learn to take some suitable action in a particular environment so as to maximise the reward.



Pre-trained models can also be used, rather than training a new model.

Model evaluation:

Validation Process

Before we evaluate our model onto a test dataset, it is a good idea to validate the model on the validation set. This is because when we have trained our model, we can't say for sure that our model works well on unseen dataset i.e. performs with required accuracy.

Thus, the process of validation is to get confidence that our model can give desired results on unseen data or to give us an assurance that the way we have assumed relations between data to produce some outputs are indeed correct.

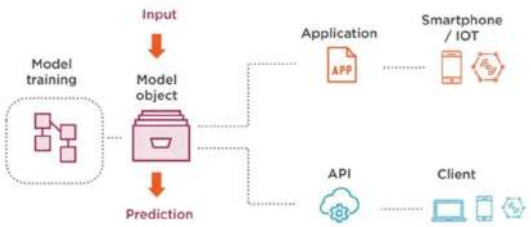
Testing Process

After training, the model's performance is assessed using a separate testing dataset or through cross-validation. The table provides different metrics usually used for regression and classification applications.

Regression Metrics	Classification Metrics
• Mean Absolute Error • Mean Squared Error • Root Mean Square Error • Root Mean Square Logarithmic Error • R^2 – Score	• Classification Accuracy • Logarithmic loss • Area under Curve • F1 score • Precision • Recall • Confusion Matrix

Model deployment:

Once a satisfactory model is developed and evaluated, it can be deployed to a production environment where it can make predictions on new, unseen data. Deployment may involve creating APIs and integrating with other systems.



Model serving platforms are programs or frameworks that make managing, scaling, and deploying machine learning models in real-world settings easier. Some of the popular platforms are listed below.

- Amazon SageMaker
- Google Cloud AI Platform
- Hugging Face
- IBM Watson Machine Learning
- KServe
- Kubeflow
- Microsoft Azure ML
- MLflow
- TensorFlow Serving

Monitoring and maintenance:

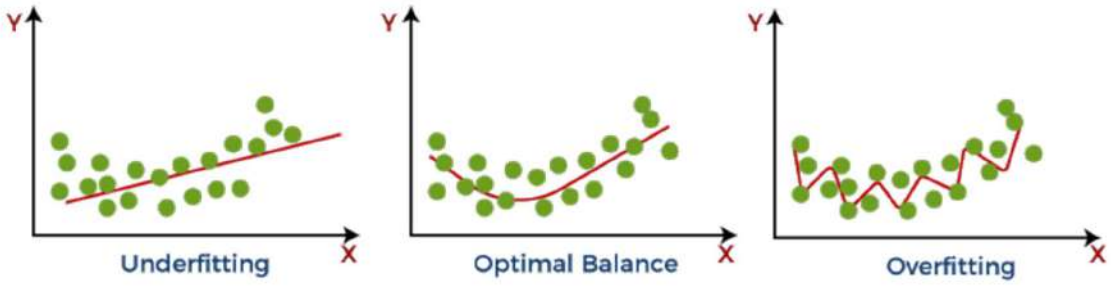
After deployment, it's important to continuously monitor the model's performance and retrain it as needed to adapt to changing data patterns. This step ensures that the model remains accurate and reliable in a real-world setting.

Bias – Variance Trade-off

Bias and variance are two sources of error in predictive models. Getting the right balance between the bias and variance trade-off is fundamental to effective machine learning algorithms. Here is a quick explanation of these concepts:

Bias

- Bias refers to error caused by a model for solving complex problems that is over simplified, makes significant assumptions, and misses important relationships in your data.
- Its also known as underfitting.
- Bias in ML is sometimes called the "too simple" problem. Bias is considered a systematic error that occurs in the machine learning model itself due to incorrect assumptions in the ML process.
- Technically, we can define bias as the error between average model prediction and the ground truth. Moreover, it describes how well the model matches the training data set:
    - **High bias.** A model with a higher bias would not match the data set closely.
    - **Low bias.** A low bias model will closely match the training data set.
- Characteristics of a high bias model include:
    - Failure to capture proper data trends
    - Potential towards underfitting
    - More generalized/overly simplified
    - High error rate



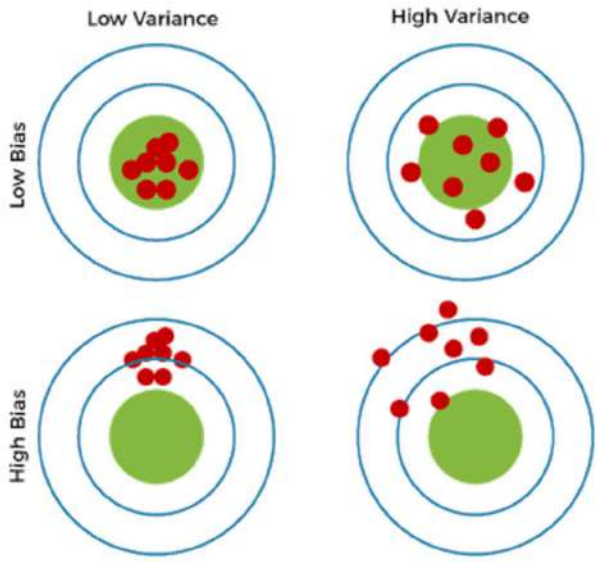
Variance

- Variance is an error caused by an algorithm that is too sensitive to fluctuations in data, creating an overly complex model that sees patterns in data that are actually just randomness.
- Variance in machine learning is sometimes called the "too sensitive" problem. Variance in ML refers to the changes in the model when using different portions of the training data set.
- Simply stated, variance is the variability in the model prediction—how much the ML function can adjust depending on the given data set. Variance comes from highly complex models with a large number of features.
    - **Low variance**. Models with high bias will have low variance.
    - **High variance**. Models with high variance will have a low bias.
- All these contribute to the flexibility of the model. For instance, a model that does not match a data set with a high bias will create an inflexible model with a low variance that results in a suboptimal machine learning model.
- Characteristics of a high variance model include:
    - Noise in the data set

- Potential towards overfitting
- Complex models
- Trying to put all data points as close as possible

Different Combinations of Bias-Variance

There are four possible combinations of bias and variances, which are represented by the below diagram:



Low-Bias, Low-Variance:

The combination of low bias and low variance shows an ideal machine learning model. However, it is not possible practically.

Low-Bias, High-Variance:

With low bias and high variance, model predictions are inconsistent and accurate on average. This case occurs when the model learns with a large number of parameters and hence leads to an **overfitting**

High-Bias, Low-Variance:

With High bias and low variance, predictions are consistent but inaccurate on average. This case occurs when a model does not learn well with the training dataset or uses few numbers of the parameter. It leads to **underfitting** problems in the model.

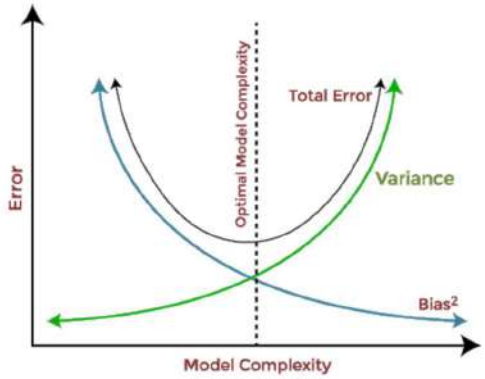
High-Bias, High-Variance:

With high bias and high variance, predictions are inconsistent and also inaccurate on average.

Bias–variance trade-off

- While building the machine learning model, it is really important to take care of bias and variance in order to avoid overfitting and underfitting in the model.

- If the model is very simple with fewer parameters, it may have low variance and high bias. Whereas, if the model has a large number of parameters, it will have high variance and low bias.
- So, it is required to make a balance between bias and variance errors, and this balance between the bias error and variance error is known as the Bias-Variance trade-off.



- For an accurate prediction of the model, algorithms need a low variance and low bias. But this is not possible because bias and variance are related to each other:
    - If we decrease the variance, it will increase the bias.
    - If we decrease the bias, it will increase the variance.
- Bias-Variance trade-off is a central issue in supervised learning.
- Ideally, we need a model that accurately captures the regularities in training data and simultaneously generalizes well with the unseen dataset.
- Unfortunately, doing this is not possible simultaneously. Because
    - a high variance algorithm may perform well with training data, but it may lead to overfitting to noisy data.
    - a high bias algorithm generates a much simple model that may not even capture important regularities in the data.
- So, we need to find a sweet spot between bias and variance to make an optimal model.

Classification

What is classification?

- In machine learning, classification solves the problem of identifying the category to which a new data point belongs. We build the classification model based on the training dataset containing data points and the corresponding labels.
- For example, let's say that we want to check whether the given image contains a person's face or not.
    - We would build a training dataset containing classes corresponding to these two classes: face and no-face.
    - We then train the model based on the training samples we have.
    - This trained model is then used for inference.
- A good classification system makes it easy to find and retrieve data.
- This is used extensively in face recognition, spam identification, recommendation engines, and so on.

Classification Steps

Data

- We will be using the MNIST dataset, which is a set of 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau.
- Each image is labelled with the digit it represents.
- Datasets loaded by Scikit-Learn generally have a similar dictionary structure including:
    - A `DESCR` key describing the dataset
    - A `data` key containing an array with one row per instance and one column per feature
    - A `target` key containing an array with the labels
- There are 70,000 images, and each image has 784 features. This is because each image is 28×28 pixels, and each feature simply represents one pixel's intensity, from 0 (white) to 255 (black).

- To download the MNIST dataset the following code is used.

```
from sklearn.datasets import fetch_openml
mnist=fetch_openml('mnist_784',version=1)
```

Data Visualization

Model Selection

In this project, binary classification was performed to distinguish between zeros and non-zeros. The SGD classifier was used to perform the binary classification task.

Performance Evaluation

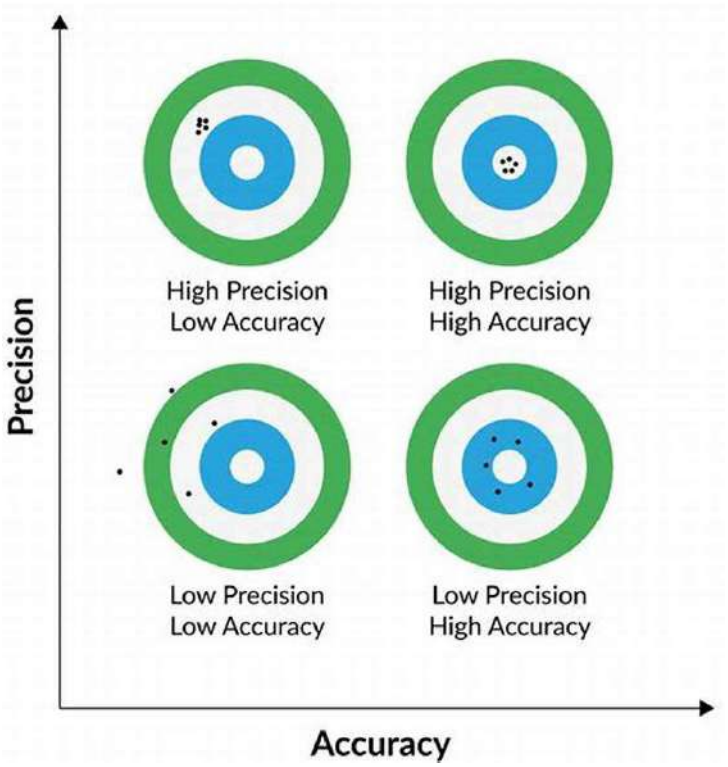
- Confusion Matrix
  - The confusion matrix is another performance measure used in classification. It is a table that allows visualization of the algorithm's performance.
  - Components of a Confusion Matrix
  - For binary classification, the confusion matrix summarizes 4 results in a 2x2 matrix.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive	False Negative
Actual Negative	False Positive	True Negative

- Classification Report
  - It finds the parameters such as Accuracy, Precision, Recall, F1 Score.

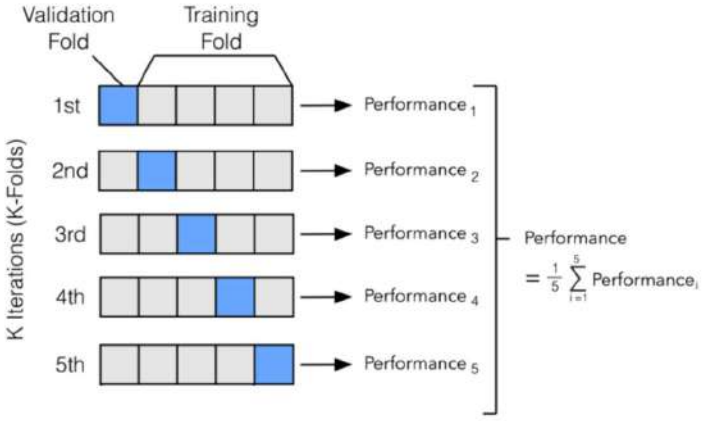
Parameter	Equation	Remark
Accuracy	$\frac{TN+TP}{TN+FP+TP+FN}$	• represents the number of correctly classified data instances over the total number of data instances. • may not be a good measure if the dataset is not balanced
Precision (Specificity)	$\frac{TP}{TP+FP}$	• Precision is a metric that gives you the proportion of true positives to the amount of total positives that the model predicts.
Recall (Sensitivity)	$\frac{TP}{TP+FN}$	• Also known as sensitivity or true positive rate

		• Recall focuses on how good the model is at finding all the positives.
F1-Score	$2 * \frac{Precision * Recall}{Precision + Recall}$	• Tells us how precise (It correctly classifies how many instances) and robust (does not miss any significant number of instances) our classifier is. • Useful in imbalanced datasets



- K-Fold Cross-Validation
  - K-fold cross-validation approach divides the input dataset into K groups of samples of equal sizes. These samples are called folds.
  - For each learning set, the prediction function uses k-1 folds, and the rest of the folds are used for the test set.
  - The brilliance of K-Fold Cross-Validation lies in its ability to mitigate the bias associated with the random shuffling of data into training and test sets, ensuring that every observation from the original dataset has the chance of appearing in the training and test set. This is crucial for models that are sensitive to the data on which they are trained.
  - **Steps to Perform K-Fold Cross-Validation**

- **Split the Dataset**: The dataset is divided into 'K' number of folds. Typically, K is set to 5 or 10, but the choice depends on the dataset size and the computational cost you're willing to incur.
- **Iterate Through Folds**: For each iteration, select one fold as the test set and the remaining K-1 folds as the training set.
- **Train and Evaluate**: Train the model on the training set and evaluate it on the test set. Record the performance score determined by your evaluation metric.
- **Repeat**: Repeat this process K times, with each of the folds serving as the test set exactly once.
- **Aggregate Results**: Calculate the average of the performance scores. This average is your model's performance metric.



- Precision-Recall Curve
  - Most imbalanced classification problems involve two classes:
    - a negative case with the majority of examples
    - a positive case with a minority of examples.
  - Two diagnostic tools that help in the interpretation of binary (two-class) classification predictive models are ROC Curves and Precision-Recall curves.
  - Plots from the curves can be created and used to understand the trade-off in performance for different threshold values when interpreting probabilistic predictions. Each plot can also be summarized with an area under the curve score that can be used to directly compare classification models.

Regression Example:
California Housing Price Prediction

Description:

The US Census Bureau has published California Census Data which has 10 types of metrics such as the population, median income, median housing price, and so on for each block group in California. The dataset also serves as an input for project scoping and tries to specify the functional and nonfunctional requirements for it.

Background of the Problem Statement:

The project aims at building a model of housing prices to predict median house values in California using the provided dataset. This model should learn from the data and be able to predict the median housing price in any district, given all the other metrics. Districts or block groups are the smallest geographical units for which the US Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people). There are 20,640 districts in the project dataset.

Dataset Description:

Variable	Type	Description
longitude	signed numeric (float)	Longitude value for the block in California, USA
latitude	numeric (float)	Latitude value for the block in California, USA
housing_median_age	numeric (int)	Median age of the house in the block
total_rooms	numeric (int)	Count of the total number of rooms (excluding bedrooms) in all houses in the block
total_bedrooms	numeric (float)	Count of the total number of bedrooms in all houses in the block
population	numeric (int)	Count of the total number of population in the block
households	numeric (int)	Count of the total number of households in the block
median_income	numeric (float)	Median of the total household income of all the houses in the block
ocean_proximity	numeric (categorical)	Type of the landscape of the block ['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND']
median_house_value	numeric (int)	Median of the household prices of all the houses in the block

- **Dataset Size**: 20640 rows x 10 columns

Steps:

1. Load the data:

- Read the “housing.csv” file from the folder into the program.
- Print first few rows of this data.

2. Perform Exploratory Data Analysis

- Perform different EDA techniques such as
    - Statistical analysis
    - Histograms
    - Joint Plot
    - Pair Plot
    - Correlation Plot
    - Regression Plot
- Extract input (X) and output (y) data from the dataset.

3. Encode categorical data:

- Convert categorical column in the dataset to numerical data.

4. Handle missing values:

- Fill the missing values with the mean of the respective column.

5. Split the dataset:

- Split the data into 80% training dataset and 20% test dataset.

6. Standardize data:

- Standardize training and test datasets.

7. Perform Linear Regression:

- Perform Linear Regression on training data.
- Predict output for test dataset using the fitted model.
- Print root mean squared error (RMSE) from Linear Regression.

8. Perform other regression methods and compare the results

- Predict using Decision Tree
- Predict using Random Forest

Types of Regression Metrics

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R Squared (R2)

Mean Absolute Error (MAE)

MAE is a very simple metric which calculates the absolute difference between actual and predicted values.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Advantages of MAE

- The MAE you get is in the same unit as the output variable.
- It is most Robust to outliers.

Disadvantages of MAE

- The graph of MAE is not differentiable so we have to apply various optimizers like Gradient descent which can be differentiable.

Python Code

```
from sklearn.metrics import mean_absolute_error
print("MAE",mean_absolute_error(y_test,y_pred))
```

Mean Squared Error (MSE)

MSE is a most used and very simple metric with a little bit of change in mean absolute error. Mean squared error states that finding the squared difference between actual and predicted value.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Advantages of MSE

- The graph of MSE is differentiable, so you can easily use it as a loss function.

Disadvantages of MSE

- The value you get after calculating MSE is a squared unit of output. for example, the output variable is in meter(m) then after calculating MSE the output we get is in meter squared.
- If you have outliers in the dataset then it penalizes the outliers most and the calculated MSE is bigger. So, in short, It is not robust to outliers which were an advantage in MAE.

Python Code

```
from sklearn.metrics import mean_squared_error
print("MSE",mean_squared_error(y_test,y_pred))
```

Root Mean Squared Error (RMSE)

As RMSE is clear by the name itself, that it is a simple square root of mean squared error.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

Advantages of RMSE

- The output value you get is in the same unit as the required output variable which makes interpretation of loss easy.

Disadvantages of RMSE

- It is not that robust to outliers as compared to MAE.

Python Code

```
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))Copy Code
```

R Squared Score (R2)

- R2 score is a metric that tells the performance of your model, not the loss in an absolute sense that how many wells did your model perform.

$$R2 = 1 - \frac{SST}{SSR}$$

Where:

  - $R2$ is the R-Squared.
  - SSR represents the sum of squared residuals between the predicted values and actual values.
  - SST represents the total sum of squares, which measures the total variance in the dependent variable.

- In contrast, MAE and MSE depend on the context as we have seen whereas the R2 score is independent of context.
- So, with help of R squared we have a baseline model to compare a model which none of the other metrics provides.
- So basically, R2 squared calculates how must regression line is better than a mean line.
- Hence, R2 squared is also known as Coefficient of Determination or sometimes also known as Goodness of fit.
- If the R2 score is zero then the above regression line by mean line is equal means 1 so 1-1 is zero. So, in this case, both lines are overlapping means model performance is worst, It is not capable to take advantage of the output column.
- If R2 score is 1, it means when the division term is zero and it will happen when the regression line does not make any mistake, it is perfect. In the real world, it is not possible.
- The regression line moves towards perfection, R2 score move towards one. And the model performance improves.

Python Code

```
from sklearn.metrics import r2_score
r2 = r2_score(y_test,y_pred)
print(r2)
```

Binary Classification Example: PIMA Indian Diabetes Dataset Classification

Description:

The Pima Indian Diabetes Dataset, originally from the National Institute of Diabetes and Digestive and Kidney Diseases, contains information of 768 women from a population near Phoenix, Arizona, USA.

The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Background of the Problem Statement:

The outcome tested was Diabetes, 268 tested positive and 500 tested negative. Therefore, there is one target (dependent) variable and the 8 attributes: pregnancies, OGTT (Oral Glucose Tolerance Test), blood pressure, skin thickness, insulin, BMI (Body Mass Index), age, pedigree diabetes function. The Pima population has been under study by the National Institute of Diabetes and Digestive and Kidney Diseases at intervals of 2 years since 1965. As epidemiological evidence indicates that T2DM results from interaction of genetic and environmental factors, the Pima Indians Diabetes Dataset includes information about attributes that could and should be related to the onset of diabetes and its future complications.

Dataset Description:

Features

Table 1. The attributes of PIMA dataset.

Attribute	Description	Type	Average/Mean
Preg	Number of times pregnant.	Numeric	3.85
Glucose	Plasma glucose concentration 2 h in an oral glucose tolerance test.	Numeric	120.89
BP	Diastolic blood pressure (mm Hg).	Numeric	69.11
SkinThickness	Triceps skinfold thickness (mm).	Numeric	20.54
Insulin	2-hour serum insulin (μlU/mL).	Numeric	79.80
BMI	Body mass index (kg/m^2).	Numeric	32
DPF	Diabetes pedigree function.	Numeric	0.47
Age	Age (years).	Numeric	33
Outcome	Diabetes diagnose results (tested_positive: 1, tested_negative: 0)	Nominal	–

Dataset Size: 768 rows x 9 columns

Steps:

1. Load the data:

- Read the “diabetes.csv” file from the folder into the program.
- Print first few rows of this data.

2. Perform Exploratory Data Analysis

- Perform different EDA techniques such as
    - Statistical analysis
    - Histograms
    - Joint Plot
    - Pair Plot
    - Correlation Plot
    - Regression Plot
- Extract input (X) and output (y) data from the dataset.

3. Encode categorical data:

- Convert categorical column in the dataset to numerical data.

4. Handle missing values:

- Fill the missing values with the mean of the respective column.

5. Split the dataset:

- Split the data into 80% training dataset and 20% test dataset.

6. Standardize data:

- Standardize training and test datasets.

7. Perform Classification using Logistic Regression:

- Perform Logistic Regression on training data.
- Predict output for test dataset using the fitted model.
- Find different performance parameters for classification.
- Generate the ROC-AUC graph
- Perform k-fold cross validation

8. Test with other classification methods and compare the results

- Classify using Random Forest

Performance Measures

Confusion Matrix

- The confusion matrix is another performance measure used in classification. It is a table that allows visualization of the algorithm’s performance.
- For binary classification, the confusion matrix summarizes 4 results in a 2x2 matrix.

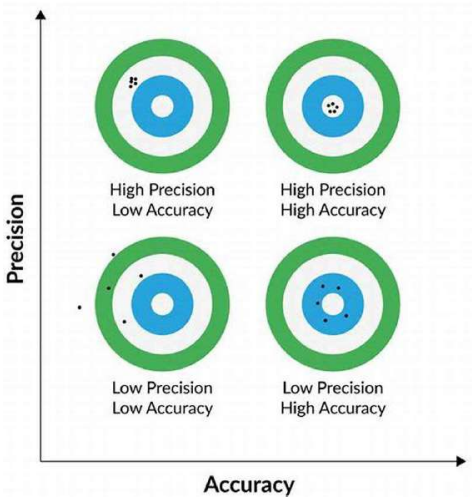
	Actual Positive	Actual Negative

Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

Classification Report

- It finds the parameters such as Accuracy, Precision, Recall, F1 Score.

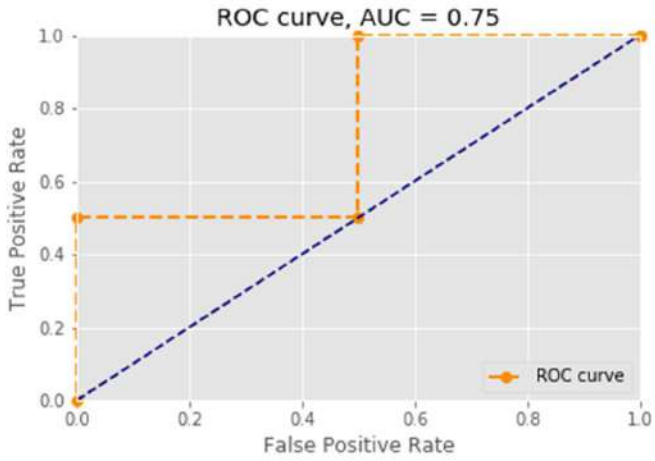
Parameter	Equation	Remark
Accuracy	$\frac{TN + TP}{TN + FP + TP + FN}$	• represents the number of correctly classified data instances over the total number of data instances. • may not be a good measure if the dataset is not balanced
Precision (Specificity)	$\frac{TP}{TP + FP}$	• Precision is a metric that gives you the proportion of true positives to the amount of total positives that the model predicts.
Recall (Sensitivity)	$\frac{TP}{TP + FN}$	• Also known as sensitivity or true positive rate • Recall focuses on how good the model is at finding all the positives.
F1-Score	$2 * \frac{Precision * Recall}{Precision + Recall}$	• Tells us how precise (correctly classifies how many instances) and robust (does not miss any significant number of instances) our classifier is. • Useful in imbalanced datasets
False Positive Rate	$\frac{FP}{FP + TN}$	• Corresponds to the proportion of negative data points that are mistakenly considered as positive, wrt all negatives • The higher FPR, the more negative data points will be misclassified.



ROC - AUC Curve

- Most imbalanced classification problems involve two classes:
  - a negative case with the majority of examples
  - a positive case with a minority of examples.
- Two diagnostic tools that help in the interpretation of binary (two-class) classification predictive models are ROC Curves and Precision-Recall curves.
- Plots from the curves can be created and used to understand the trade-off in performance for different threshold values when interpreting probabilistic predictions.
- Each plot can also be summarized with an area under the curve score that can be used to directly compare classification models.
- The **ROC AUC score** is a crucial metric in machine learning, particularly for evaluating the performance of binary classification models. It stands for **Receiver Operating Characteristic - Area Under the Curve** and provides a graphical representation of a model's ability to distinguish between positive and negative classes.
- **ROC Curve**
  - The **ROC curve** is a plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the **True Positive Rate (TPR)** against the **False Positive Rate (FPR)** at various threshold settings.

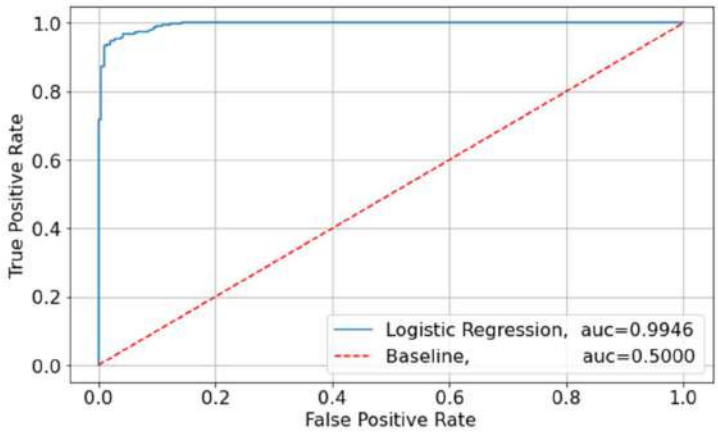
		Threshold					
Actual Class (y)	**Predicted Probabilities ($\hat{y}$)**	($\hat{y}_0$)	($\hat{y}_{0.2}$)	($\hat{y}_{0.4}$)	($\hat{y}_{0.6}$)	($\hat{y}_{0.8}$)	($\hat{y}_1$)
1	*0.8*	1	1	1	1	1	0
0	*0.6*	1	1	1	1	0	0
1	*0.4*	1	1	1	0	0	0
0	*0.2*	1	1	0	0	0	0
TPR		1	1	1	0.5	0.5	0
FPR		1	1	0.5	0.5	0	0



- **AUC (Area Under the Curve)**
  - The **AUC** represents the area under the ROC curve and provides a single scalar value to summarize the performance of the classifier. An AUC of 1.0 indicates a perfect model, while an AUC of 0.5 suggests a model with no discriminative power, equivalent to random guessing.
  - For example, let's have a binary classification problem with 4 observations. We know true class and predicted probabilities obtained by the algorithm. All we need to do, based on different threshold values, is to compute True Positive Rate (TPR) and False Positive Rate (FPR) values for each of the thresholds and then plot TPR against FPR.

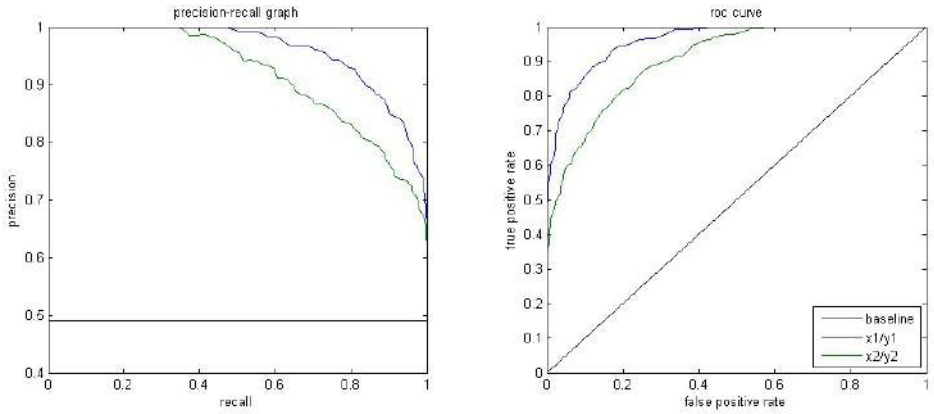
  - **ROC-AUC for Binary Classification**

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# Load data
X, y = load_breast_cancer(return_X_y=True)

# Train model
clf = LogisticRegression(solver="liblinear", random_state=0).fit(X, y)

# Compute ROC AUC score
roc_auc = roc_auc_score(y, clf.predict_proba(X)[:, 1])

y_pred_proba = clf.predict_proba(X)[::,1]
fpr, tpr, _ =roc_curve(y,  y_pred_proba)
auc =roc_auc_score(y, y_pred_proba).round(4)
plt.plot(fpr,tpr,label="Logistic Regression,  auc="+str(auc))
plt.plot([0, 1], [0, 1], 'r--', label='Baseline,
auc=0.5000')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend(loc=4)
plt.show()
```



Precision Recall Curve

- **Precision** is the proportion of *correct* positive classifications (true positive) divided by the total number of *predicted* positive classifications that were made (true positive + false positive).
- **Recall** is the proportion of *correct* positive classifications (true positive) divided by the total number of the *truly* positive classifications (true positive + false negative).
- It is important to note that Precision is also called the Positive Predictive Value (PPV).
- The recall is also called Sensitivity, Hit Rate, or True Positive Rate (TPR).
- A PR curve is simply a graph with Precision values on the y-axis and Recall values on the x-axis.



- **Interpreting PR Curve**
  - It is desired that the algorithm should have both high precision and high recall.
  - However, most machine learning algorithms often involve a trade-off between the two. A good PR curve has greater AUC (area under the curve).
  - In the figure above, the classifier corresponding to the blue line has better performance than the classifier corresponding to the green line.
  - It is important to note that the classifier that has a higher AUC on the ROC curve will always have a higher AUC on the PR curve as well.
  - Precision-Recall curves are preferable when dealing with imbalanced datasets, focusing on positive class prediction performance.
  - Precision-Recall provides insights into the model's ability to correctly classify positive instances.

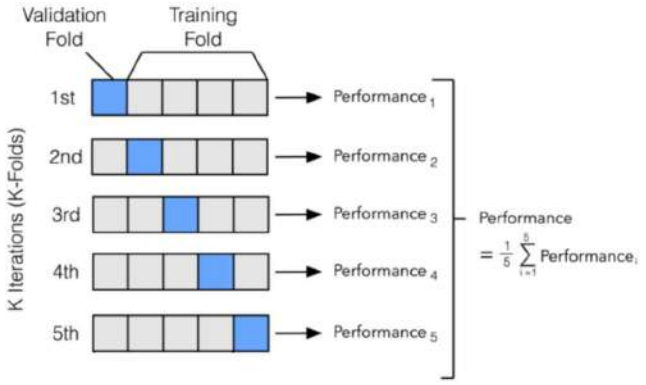
```
from sklearn.datasets import load_breast_cancer
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_curve, auc

# Load data
X, y = load_breast_cancer(return_X_y=True)

# Train – Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Train a logistic regression model (you can replace this with your own
classifier)
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict probabilities for positive class
y_scores = model.predict_proba(X_test)[:, 1]

# Calculate precision and recall
precision, recall, thresholds = precision_recall_curve(y_test, y_scores)

# Calculate Area Under the Curve (AUC) for precision-recall curve
auc_score = auc(recall, precision)
no_skill = len(y[y==1]) / len(y)

# Plot precision-recall curve
plt.figure(figsize=(10, 6))
plt.plot(recall, precision, label=f'Precision-Recall Curve (AUC =
{auc_score:.4f})')
plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No
Skill(%ge of positive samples)')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid()
plt.show()
```

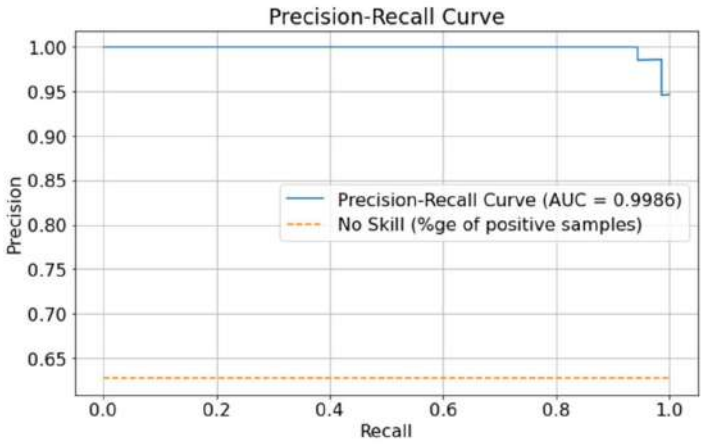
K-Fold Cross-Validation

- K-fold cross-validation approach divides the input dataset into K groups of samples of equal sizes. These samples are called folds.
- For each learning set, the prediction function uses k-1 folds, and the rest of the folds are used for the test set.
- The brilliance of K-Fold Cross-Validation lies in its ability to mitigate the bias associated with the random shuffling of data into training and test sets, ensuring that every observation from the original dataset has the chance of appearing in the training and test set. This is crucial for models that are sensitive to the data on which they are trained.
- **Steps to Perform K-Fold Cross-Validation**

i. **Split the Dataset**: The dataset is divided into 'K' number of folds. Typically, K is set to 5 or 10, but the choice depends on the dataset size and the computational cost you're willing to incur.
ii. **Iterate Through Folds**: For each iteration, select one fold as the test set and the remaining K-1 folds as the training set.
iii. **Train and Evaluate**: Train the model on the training set and evaluate it on the test set. Record the performance score determined by your evaluation metric.
iv. **Repeat**: Repeat this process K times, with each of the folds serving as the test set exactly once.
v. **Aggregate Results**: Calculate the average of the performance scores. This average is your model's performance metric.

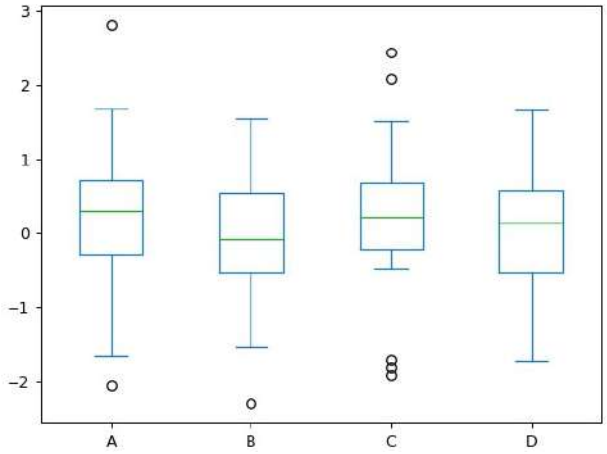



```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LogisticRegression
import numpy as np

# Load data
X, y = load_breast_cancer(return_X_y=True)

# Train model
model = LogisticRegression(solver="liblinear").fit(X, y)

# Define cross-validation method to use
cv = KFold(n_splits=5, random_state=1, shuffle=True)

# Use k-fold CV to evaluate model
scores = cross_val_score(model, X, y, cv=cv, n_jobs=-1)

# View mean performance
mean_score = np.mean(np.absolute(scores))

print('Foldwise Score:',scores.round(4))
print('Average  Score: ',mean_score.round(4))
```



Outlier handling

- A data point that varies greatly from other results is referred to as an outlier.
- An outlier may also be described as an observation in our data that is incorrect or abnormal as compared to other observations.
- **Causes and Consequences**
  - **Measurement errors:** Errors in data collection or measurement processes can lead to outliers.
  - **Sampling errors:** In some cases, outliers can arise due to issues with the sampling process.
  - **Natural variability:** Inherent variability in certain phenomena can also lead to outliers. Some systems may exhibit extreme values due to the nature of the process being studied.
  - **Data entry errors:** Human errors during data entry can introduce outliers.
  - **Experimental errors:** In experimental settings, anomalies may occur due to uncontrolled factors, equipment malfunctions, or unexpected events.
  - **Sampling from multiple populations:** Data is inadvertently combined from multiple populations with different characteristics.
  - **Intentional outliers:** Outliers are introduced intentionally to test the robustness of statistical methods.
- To find outliers, we can simply plot the box plot. Outliers are points that are outside of the minimum and maximum values, as seen in the image below.



Visualizing and Removing Outliers Using Box Plot

- Boxplot summarizes sample data using 25th, 50th, and 75th percentiles.
- One can just get insights (quartiles, median, and outliers) into the dataset by just looking at its boxplot.

```
# Importing
import sklearn
from sklearn.datasets import load_diabetes
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


# Load the dataset
diabetics = load_diabetes()

# Create the dataframe
column_name = diabetics.feature_names
df = pd.DataFrame(diabetics.data)
df.columns = column_name
print(df.shape)
df.head()

# Create the Box Plot
sns.boxplot(df_diabetics['bmi'])

# Choose a threshold from Boxplot and remove the outliers
threshold = 0.12
df_no_outliers1 = df[df['bmi'] <= threshold]

df_no_outliers1.shape
```

Removal of Outliers with Z-Score

- Z- Score is also called a standard score.

$$Zscore = \frac{(data_{point} - mean)}{std.deviation}$$

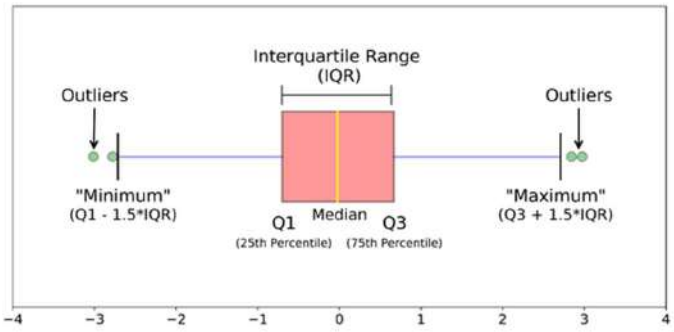
- This value/score helps to understand that how far is the data point from the mean. And after setting up a threshold value one can utilize z score values of data points to define the outliers.

```
from scipy import stats
import numpy as np

z = np.abs(stats.zscore(df['bmi']))
sns.boxplot(z)
print(z)

threshold_z = 2.3

outlier_indices = np.where(z > threshold_z)[0]
df_no_outliers2 = df.drop(outlier_indices)
print("Original DataFrame Shape:", df.shape)
print("DataFrame Shape after Removing Outliers:", df_no_outliers2.shape)
```

Handling Outliers using IQR (Inter Quartile Range)

- IQR (Inter Quartile Range) Inter Quartile Range approach to finding the outliers is the most commonly used and most trusted approach used in the research field.

$$IQR = Quartile_3 - Quartile_1$$




```
# IQR
Q1 = df['bmi'].quantile(0.25)
Q3 = df['bmi'].quantile(0.75)
IQR = Q3 - Q1
print('IQR = ',IQR.round(4))

upper = Q3+1.5*IQR
lower = Q1-1.5*IQR

# Create arrays of Boolean values indicating the outlier rows
upper_index = np.where(df['bmi'] >= upper)[0]
lower_index = np.where(df['bmi'] <= lower)[0]

# Removing the outliers
df.drop(index=upper_index, inplace=True)
df.drop(index=lower_index, inplace=True)

# Print the new shape of the DataFrame
print("New Shape: ", df.shape)
```

Linear Regression

Introduction

Linear regression is a statistical method for modelling the relationship between a dependent variable and one or more independent variables. It's one of the most straightforward and widely used techniques in predictive analytics.

Types of Linear Regression

1. **Simple Linear Regression:** Models the relationship between two variables by fitting a linear equation to the observed data.
2. **Multiple Linear Regression:** Models the relationship between one dependent variable and multiple independent variables.

Mathematical Formulation

Simple Linear Regression

The equation for simple linear regression is:

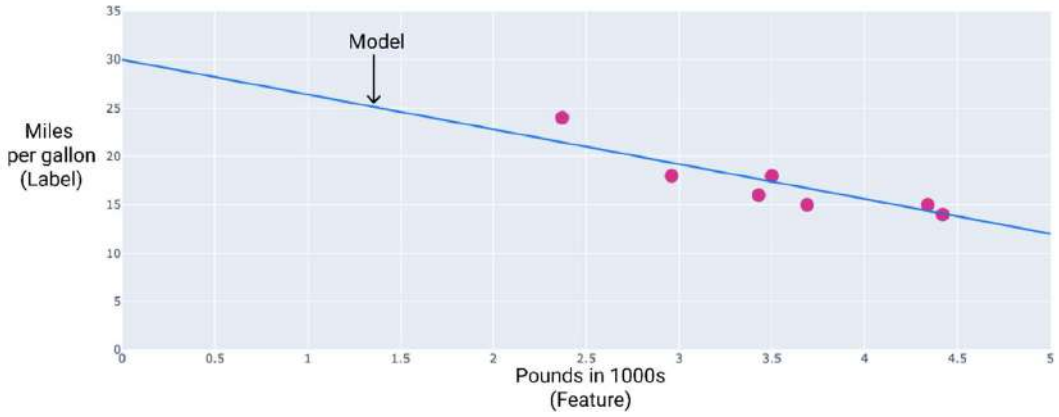
$$y = \theta_0 + \theta_1 x + \epsilon$$

- y is the dependent variable.
- θ_0 is the intercept or **bias**.
- θ_1 is the slope of the line or the **weight** of the feature.
- x is the independent variable.
- ϵ is the error term.

For example, suppose we want to predict a car's fuel efficiency in miles per gallon based on how heavy the car is, and we have the following dataset:

Pounds in 1000s (feature)	Miles per gallon (label)
3.50	18
3.69	15
3.44	18
3.43	16
4.34	15
4.42	14
2.37	24

A model based on the above data can be created as follows. The line here is the best fit line for the dataset.



Multiple Linear Regression

Although the example in this section uses only one feature—the heaviness of the car—a more sophisticated model might rely on multiple features, each having a separate weight (θ_1, θ_2, etc.).

The equation for multiple linear regression is:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_N x_N + \epsilon$$

- y is the dependent variable.
- θ_0 is the intercept or the bias.
- $\theta_1, \theta_2, \cdots, \theta_N$ are the coefficients or weights.
- $x_1, x_2, \cdots, x_N$ are the independent variables.
- ϵ is the error term.

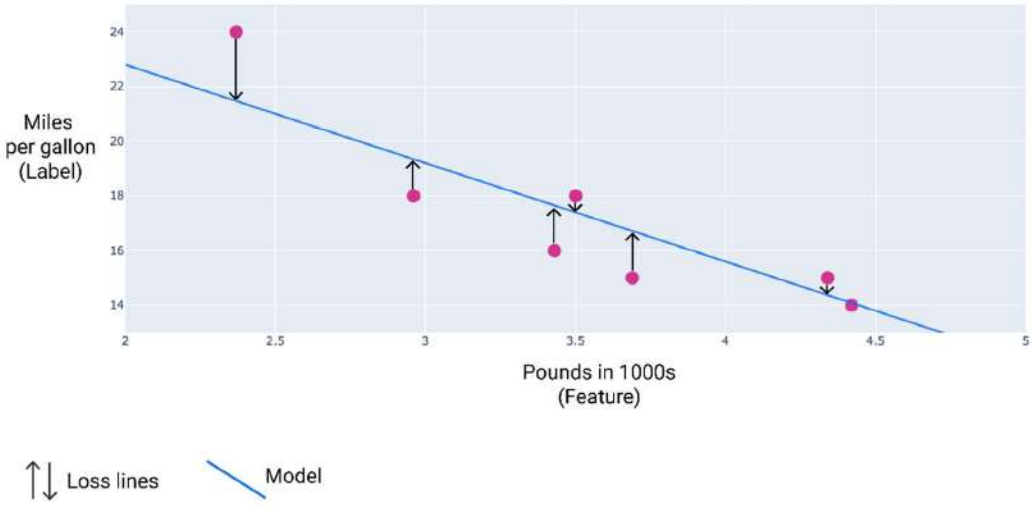
For example, a model that predicts gas mileage could additionally use features such as the following:

- Engine displacement
- Acceleration
- Number of cylinders
- Horsepower

Linear regression: Loss

Loss is a numerical metric that describes how wrong a model's **predictions** are. Loss measures the distance between the model's predictions and the actual labels. The goal of training a model is to minimize the loss, reducing it to its lowest possible value.

In the following image, you can visualize loss as arrows drawn from the data points to the model. The arrows show how far the model's predictions are from the actual values.



Loss is measured from the actual value to the predicted value.

Distance of loss

In statistics and machine learning, loss measures the difference between the predicted and actual values. Loss focuses on the *distance* between the values, not the direction. For example, if a model predicts 2, but the actual value is 5, we don't care that the loss is negative −3 (2−5=−3).

Instead, we care that the *distance* between the values is 3. Thus, all methods for calculating loss remove the sign.

The two most common methods to remove the sign are the following:

- Take the absolute value of the difference between the actual value and the prediction.
- Square the difference between the actual value and the prediction.

Types of loss

In linear regression, there are four main types of loss, which are outlined in the following table.

Loss type	Definition	Equation
L1 loss	The sum of the absolute values of the difference between the predicted values and the actual values.	$\sum\|actual\ value - predicted\ value\|$
Mean absolute error (MAE)	The average of L_1 losses across a set of examples.	$\frac{1}{N}\sum\|actual\ value - predicted\ value\|$
L2 loss	The sum of the squared difference between the predicted values and the actual values.	$\sum(actual\ value - predicted\ value)^2$
Mean squared error (MSE)	The average of L_2 losses across a set of examples.	$\frac{1}{N}\sum(actual\ value - predicted\ value)^2$

- The functional difference between L_1 loss and L_2 loss (or between MAE and MSE) is squaring. When the difference between the prediction and label is large, squaring makes the loss even larger. When the difference is small (less than 1), squaring makes the loss even smaller.
- When processing multiple examples at once, we recommend averaging the losses across all the examples, whether using MAE or MSE.
- Deciding whether to use MAE or MSE can depend on the dataset and the way you want to handle certain predictions. Most feature values in a dataset typically fall within a distinct range. For example,
    - cars are normally between 2000 and 5000 pounds and get between 8 to 50 miles per gallon.
    - An 8,000-pound car, or a car that gets 100 miles per gallon, is outside the typical range and would be considered an **outlier**.
- An outlier can also refer to how far off a model's predictions are from the real values. For instance,
    - a 3,000-pound car or a car that gets 40 miles per gallon are within the typical ranges.
    - However, a 3,000-pound car that gets 40 miles per gallon would be an outlier in terms of the model's prediction because the model would predict that a 3,000-pound car would get between 18 and 20 miles per gallon.
- When choosing the best loss function, consider how you want the model to treat outliers. For instance, MSE moves the model more toward the outliers, while MAE doesn't. L_2 loss incurs a much higher penalty for an outlier than L_1 loss. For example, the following images show a

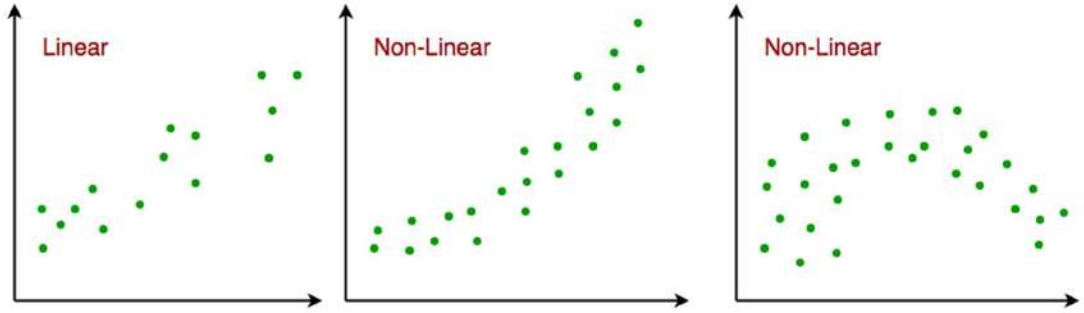
model trained using MAE and a model trained using MSE. The red line represents a fully trained model that will be used to make predictions. The outliers are closer to the model trained with MSE than to the model trained with MAE.

Interactive Exercise

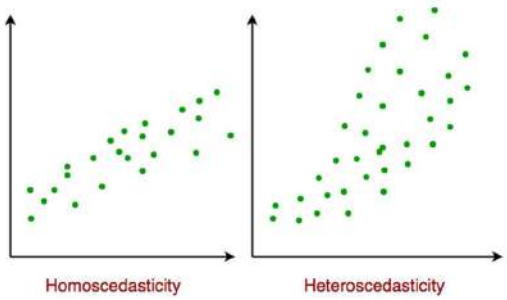
https://developers.google.com/machine-learning/crash-course/linear-regression/parameters-exercise

Assumptions of Linear Regression

1. **Linearity:** The relationship between the dependent and independent variables is linear. This implies that changes in the dependent variable follow those in the independent variable(s) in a linear fashion. This means that there should be a straight line that can be drawn through the data points. If the relationship is not linear, then linear regression will not be an accurate model.



2. **Independence:** The residuals (errors) are independent. This means that the value of the dependent variable for one observation does not depend on the value of the dependent variable for another observation.
3. **Homoscedasticity:** Across all levels of the independent variable(s), the variance of the errors is constant. This indicates that the amount of the independent variable(s) has no impact on the variance of the errors.



4. **Normality:** The residuals of the model are normally distributed. This means that the residuals should follow a bell-shaped curve.
5. **No Multicollinearity:** In multiple regression, the independent variables should not be highly correlated with each other. This indicates that there is little or no correlation between the independent variables. Multicollinearity occurs when two or more independent variables are highly correlated with each other, which can make it difficult to determine the individual effect of each variable on the dependent variable.

Cost Function

It is a function that measures the performance of a model for any given data. Cost Function quantifies the error between predicted values and expected values and presents it in the form of a single real number.

After making a hypothesis with initial parameters, we calculate the Cost function. And with a goal to reduce the cost function, we modify the parameters by using the Gradient descent algorithm over the given data. Here's the mathematical representation for it:

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

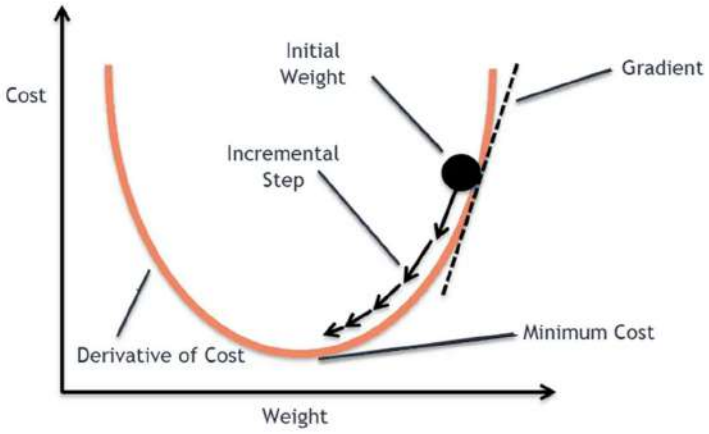
Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Gradient Descent Algorithm

- **Gradient descent** is a mathematical technique that iteratively finds the weights and bias that produce the model with the lowest loss.
- Gradient descent finds the best weight and bias by repeating the following process for a number of user-defined iterations.
- The model begins training with randomized weights and biases near zero, and then repeats the following steps:
  1. Calculate the loss with the current weight and bias.
  2. Determine the direction to move the weights and bias that reduce loss.
  3. Move the weight and bias values a small amount in the direction that reduces loss.
  4. Return to step one and repeat the process until the model can't reduce the loss any further.



$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$
$$(\text{for } j = 1 \text{ and } j = 0)$$
$$\}$$

α is called **Learning rate** – a tuning parameter in the optimization process. It decides the length of the steps.

Types of Gradient Descent Algorithm

The choice of gradient descent algorithm depends on the problem at hand and the size of the dataset.

Batch Gradient Descent

- Batch gradient descent updates the model's parameters using the gradient of the entire training set.
- It calculates the average gradient of the cost function for all the training examples and updates the parameters in the opposite direction.
- Batch gradient descent guarantees convergence to the global minimum but can be computationally expensive and slow for large datasets.

Stochastic Gradient Descent

- Stochastic gradient descent updates the model's parameters using the gradient of one training example at a time.
- It randomly selects a training dataset example, computes the gradient of the cost function for that example, and updates the parameters in the opposite direction.
- Stochastic gradient descent is computationally efficient and can converge faster than batch gradient descent.
- However, it can be noisy and may not converge to the global minimum.

Mini-Batch Gradient Descent

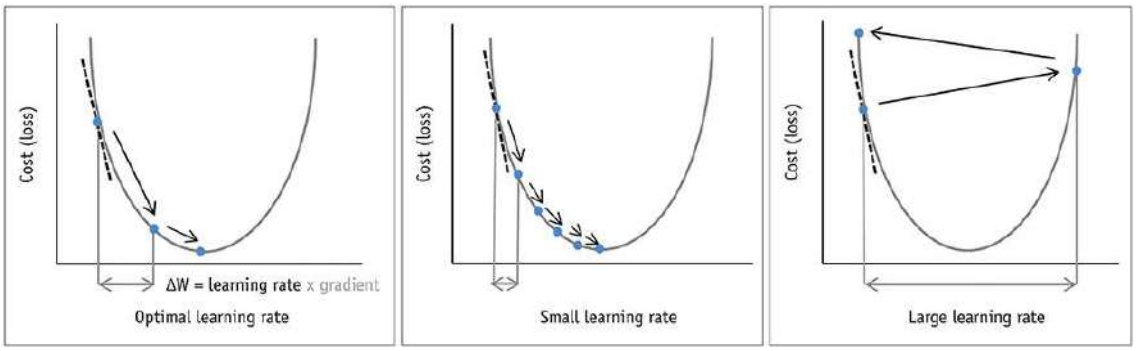
- Mini-batch gradient descent updates the model's parameters using the gradient of a small batch size of the training dataset, known as a mini-batch.
- It calculates the average gradient of the cost function for the mini-batch and updates the parameters in the opposite direction.
- The mini-batch gradient descent algorithm combines the advantages of batch and stochastic gradient descent hence is the most commonly used method in practice.
- It is computationally efficient and less noisy than stochastic gradient descent while still being able to converge to a good solution.

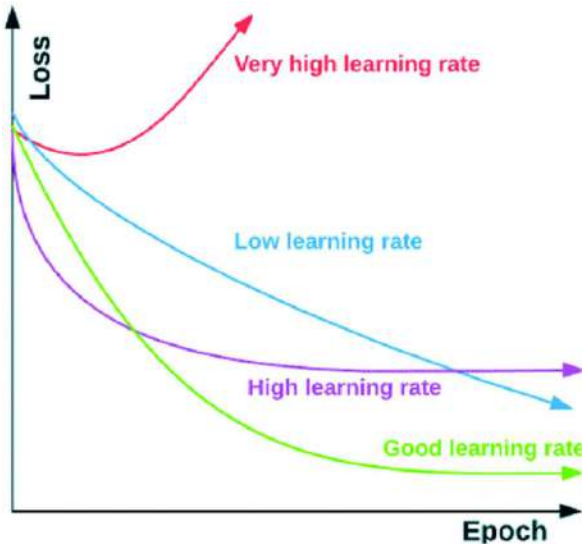
Alpha – The Learning Rate

We have the direction we want to move in. Now, we must decide the size of the step we must take.
It must be chosen carefully to end up with local minima.

- If the learning rate is too high, we might **OVERSHOOT** the minima and keep bouncing without reaching the minima
- If the learning rate is too small, the training might turn out to be too long
- The learning rate is optimal, and the model converges to the minimum.
- The learning rate is too small. It takes more time but converges to the minimum.
- The learning rate is higher than the optimal value. It overshoots but converges.

- The learning rate is very large. It overshoots and diverges, moves away from the minima, and performance decreases in learning.





Model Evaluation Metrics

1. **R-squared (**R2**):** Represents the proportion of variance for the dependent variable that's explained by the independent variables.
2. **Adjusted R-squared**: Adjusted for the number of predictors in the model.
3. **Mean Absolute Error (MAE)**: The average absolute difference between predicted and actual values.
4. **Mean Squared Error (MSE)**: The average squared difference between predicted and actual values.
5. **Root Mean Squared Error (RMSE)**: The square root of MSE, representing the standard deviation of the residuals.

Steps to Perform Linear Regression

1. Data Preparation

- Load and explore the dataset.
- Handle missing values.
- Encode categorical variables.
- Split the data into training and testing sets.

2. Model Training

- Fit the linear regression model to the training data.
- Calculate the coefficients ($\theta_0, \theta_1, \theta_2 \ldots \theta_N$).

3. Model Prediction

- Use the model to make predictions on the test data.

4. Model Evaluation

- Evaluate the model's performance using the metrics discussed earlier.

Example Code in Python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

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

# Explore dataset
print(df.head())

# Handle missing values if any
df = df.dropna()

# Split the data into training and testing sets
X = df[['independent_variable1', 'independent_variable2']]
y = df['dependent_variable']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f'MAE: {mae}')
print(f'MSE: {mse}')
print(f'RMSE: {rmse}')
print(f'R-squared: {r2}')
```

```
# Visualize the results (example for simple linear regression)
plt.scatter(X_test['independent_variable1'], y_test, color='blue',
label='Actual')
plt.scatter(X_test['independent_variable1'], y_pred, color='red',
label='Predicted')
plt.xlabel('Independent Variable 1')
plt.ylabel('Dependent Variable')
plt.legend()
plt.title('Actual vs Predicted')
plt.show()
```

Regularization

Regularization is a vital technique in machine learning, particularly in linear regression, aimed at improving model generalization. It addresses the common issue of overfitting, where a model performs well on training data but poorly on unseen data. By adding a penalty term to the loss function, regularization discourages overly complex models and encourages simpler, more robust solutions.

Key Concepts

1. **Overfitting**: Overfitting occurs when a model learns not only the underlying patterns in the training data but also the noise, leading to poor performance on new data. This is often characterized by large weights in the model.
2. **Bias-Variance Trade-off**: Regularization helps balance bias and variance. While it introduces some bias by constraining model complexity, it significantly reduces variance, leading to better generalization.
3. **Objective Function with Regularization:**
   Regularization modifies the loss function used in linear regression. The general form becomes:
   $$J(w) = Loss + \lambda \cdot Penalty$$
   where:
   $Loss$ is typically the Mean Squared Error (MSE).
   λ is the regularization parameter (controls the strength of the penalty),

   Penalty depends on the type of regularization.

4. **Choosing Regularization Parameters**
   - **λ:** Controls the trade-off between the loss function and the penalty.
     - A larger λ increases regularization, leading to smaller coefficients (possibly underfitting).
     - A smaller λ reduces regularization, increasing the risk of overfitting.
   - **Cross-validation** is often used to determine the optimal value of λ.

Types of Regularization Techniques

There are three primary techniques for regularization in linear regression:

1. **Ridge Regression (L2 Regularization)**:
   - Adds a penalty equal to the square of the magnitude of coefficients.
   - The loss function is modified as follows:

$$J(\theta) = \sum_{i=1}^{N} (y_i - \theta^T x_i)^2 + \lambda \sum_{j=1}^{M} \theta_j^2$$

- Here, λ controls the strength of the penalty; larger values lead to simpler models by shrinking coefficients towards zero without forcing them to be exactly zero.

2. **Lasso Regression (L1 Regularization):**
   - Known as Least Absolute Shrinkage and Selection Operator (LASSO).
   - Introduces a penalty equal to the absolute value of the magnitude of coefficients.
   - The modified loss function is:

$$J(\theta) = \sum_{i=1}^{N} (y_i - \theta^T x_i)^2 + \lambda \sum_{j=1}^{M} |\theta_j|$$

   - Lasso can lead to sparse solutions, effectively performing feature selection by driving some coefficients exactly to zero.

3. **Elastic Net**:
   - Combines both L1 and L2 penalties.
   - Useful when there are multiple features correlated with each other.
   - The loss function becomes:

$$Loss = \sum_{i=1}^{N} (y_i - w^T x_i)^2 + \lambda_1 \sum_{j=1}^{M} |w_j| + \lambda_2 \sum_{j=1}^{M} w_j^2$$

   - This allows for flexibility in regularization by adjusting λ_1 and λ_2

Impact of Regularization

- **Model Complexity**: Regularization reduces model complexity by penalizing large weights, which helps prevent overfitting and enhances generalizability to new data.
- **Training vs. Test Performance**: While regularized models may show slightly worse performance on training data due to increased bias, they typically perform better on test data due to reduced variance.
- **Selection of Regularization Parameter (*λ*)**: The choice of *λ* is crucial. A high *λ* value can lead to underfitting, while a low value may not sufficiently reduce overfitting. Cross-validation is often used to determine an optimal *λ* value.

Program for Regularization

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso,
ElasticNet
from sklearn.metrics import mean_squared_error

# Generate synthetic data
X, y, coef = make_regression(n_samples=1000, n_features=20, noise=0.1,
coef=True, random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize models with regularization parameters
lr_model = LinearRegression()    # Simple Linear Regression
ridge_model = Ridge(alpha=0.8)   # L2 regularization
lasso_model = Lasso(alpha=0.1)   # L1 regularization
elastic_net_model = ElasticNet(alpha=0.05, l1_ratio=0.99)
                                 # Combination of L1 and L2

# Fit the models
lr_model.fit(X_train, y_train)
ridge_model.fit(X_train, y_train)
lasso_model.fit(X_train, y_train)
elastic_net_model.fit(X_train, y_train)

# Make predictions
lr_predictions = lr_model.predict(X_test)
ridge_predictions = ridge_model.predict(X_test)
lasso_predictions = lasso_model.predict(X_test)
elastic_net_predictions = elastic_net_model.predict(X_test)

# Calculate Mean Squared Error for each model
lr_mse = mean_squared_error(y_test, lr_predictions)
ridge_mse = mean_squared_error(y_test, ridge_predictions)
lasso_mse = mean_squared_error(y_test, lasso_predictions)
elastic_net_mse = mean_squared_error(y_test, elastic_net_predictions)

# Print the results
print("    Linear Regression MSE:", lr_mse)
print("     Ridge Regression MSE:", ridge_mse)
print("     Lasso Regression MSE:", lasso_mse)
print("ElasticNet Regression MSE:", elastic_net_mse)
```

```
    Linear Regression MSE: 0.28569378348262137
     Ridge Regression MSE: 0.3218003800476734
     Lasso Regression MSE: 0.384244937541749
ElasticNet Regression MSE: 0.3413492449938259
```

LOGISTIC REGRESSION

* Linear regression ⇒

$\hat{y} = h_\theta(x_i)$

domain ⇒ $[-\infty, +\infty]$

Range ⇒ $[-\infty, +\infty]$

* For classification problems, y takes categorical values.

(i) Binary classification

$y = 0$ or 1.

(ii) Multiclass / Multinomial Classi.

y ⇒ multiple values.

(a) ECG classification

5 Classes
- Normal (N)
- Supraventricular ectopic (S)
- Ventricular ectopic (V)
- Fusion (F)
- Unknown (Q).

(b) Alzheimers using MRI

4 Classes
- Non demented (ND)
- Moderate " (MD)
- Very Mild " (VMD).
- Severe " (SD)

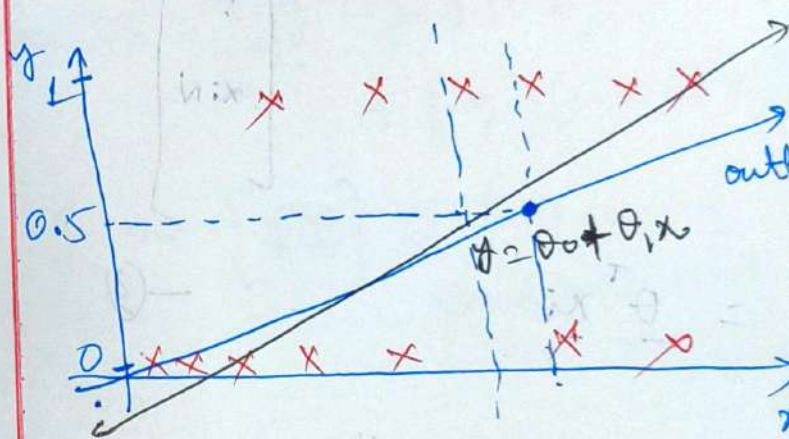
* Objective of a classifier ⇒ predict the probability of an observation belonging to a class.

* Let p ⇒ probability of $y=1$ when $X = x$

* If we use linear regression then

$p = h_\theta(y=1 | X=x; \theta) = \theta_0 + \theta_1 x$

* Since p lies in $[0,1]$, the hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$ will not satisfy as linear functions are unbounded.



* Odds: The odds of an event occuring is the ratio of the expected no. of times occuring and not occuring

$$O = \frac{m}{n-m} = \frac{m/n}{1-m/n}$$

$$\Rightarrow \boxed{O = \frac{p}{1-p}}$$

Range $[0, \infty]$

BUILDING THE LOGISTIC REGRESSION MODEL (LOGIT FUNCTION)

* Logistic Model equation

$$\log\left[\frac{p_i}{1-p_i}\right] = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} - \ldots + \theta_N x_{iN} \quad \text{—①}$$

$\log\left[\frac{p_i}{1-p_i}\right]$ = logit function.

$$\theta_0 + \theta_1 x_{i1} + \cdots + \theta_N x_{iN} \quad \text{—②}$$

$$= [\theta_0, \theta_1, \theta_2 \ldots \theta_N] \begin{bmatrix} 1 \\ x_{i1} \\ x_{i2} \\ \vdots \\ x_{iN} \end{bmatrix}$$

$$= \underline{\theta}^T \underline{x_i} \quad \text{—③}$$

① can be re-written as

$$\log\left[\frac{p_i}{1-p_i}\right] = \underline{\theta}^T \underline{x_i}$$

$$\Rightarrow \frac{p_i}{1-p_i} = e^{\theta^T x_i}$$

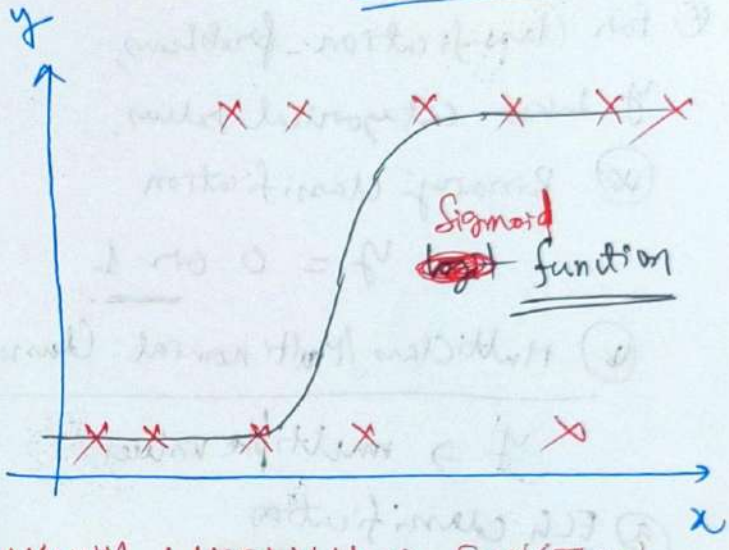
$$\Rightarrow \frac{1-p_i}{p_i} = \frac{1}{e^{\theta^T x_i}}$$

$$\Rightarrow \frac{1}{p_i} = 1 + \frac{1}{e^{\theta^T x_i}}$$

$$\Rightarrow \boxed{p_i = \frac{e^{\theta^T x_i}}{1 + e^{\theta^T x_i}}}$$

$$\Rightarrow \boxed{p_i = \frac{1}{1 + e^{-\theta^T x_i}}}$$

$p_i \Rightarrow$ Range $[0, 1]$



MAXIMUM LIKELYHOOD FUNCTION

* In linear regression we have used least squares method which finds $h_\theta(x)$ for lowest sum of residuals.

* In logistic regression we will use maximum likelyhood estimation.

* Likelyhood function: observed values of dependant variables may be predicted from observing the independant variable.

$$h_\theta(x_i) = \sigma(\theta^T x_i) = \frac{1}{1 + e^{-\theta^T x_i}}$$

$(y_i = \text{category} \mid x_i; \theta) = \underline{\theta}^T \underline{x}$

$\theta^T = [\theta_0, \theta_1, \theta_2 \ldots \theta_N]$

$x = [1, x_1, x_2 \ldots x_N]^T$ (Column Vector)

$\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \ldots \theta_N x_N$

$h_\theta(x)$ = hypothesis

$$h_\theta(x) = \sigma(\theta^T \underline{x}) = \frac{1}{1 + e^{-\theta^T \underline{x}}}$$

$$P_i(y_i = 1 \mid x_i; \theta) = \frac{1}{1 + e^{-\theta^T x_i}} = h_\theta(x_i)$$

$$P_i(y_i = 0 \mid x_i; \theta) = 1 - h_\theta(x_i)$$

↳ complimentary

Combining the above into one equation

$$P(y_i \mid x_i; \theta) = [h_\theta(x_i)]^{(y_i)} [1 - h_\theta(x_i)]^{(1-y_i)}$$

↓

Bernoulli distribution

⊛ gives probability of output for 1 observation.

⊛ Since the dataset has N such observations

$$P(\underline{y} \mid X; \theta) = \prod_{i=1}^{N} h_\theta(x_i)^{y_i} (1 - h_\theta(x_i))^{1-y_i}$$

⊛ we have assumed that the outputs are independent of each other.

$[P(A \cap B) = P(A) \times P(B)]$

↳ A, B independent events

⊛ $L(\theta) = \prod_{i=1}^{N} h_\theta(x_i)^{y_i} (1 - h_\theta(x_i))^{1-y_i}$

$L(\theta)$ = likelyhood function

⊛ To find the model parameters we need to maximize $L(\theta)$.

⊛ $L(\theta)$ is a complex function to find maxima hence log-likelyhood function is used

$$\mathcal{L}(\theta) = \log[L(\theta)]$$

$$\Rightarrow \mathcal{L}(\theta) = \log\left[\prod_{i=1}^{N} h_\theta(x_i)^{y_i} (1 - h_\theta(x_i))^{1-y_i}\right]$$

$$\Rightarrow \mathcal{L}(\theta) = \sum_{i=1}^{N} \left[y_i \log(\sigma(\theta^T x_i)) + (1 - y_i)(1 - \sigma(\theta^T x_i))\right]$$

⊛ $\max[\mathcal{L}(\theta)]$ ⇒ find using gradient ascent (as maximisation).

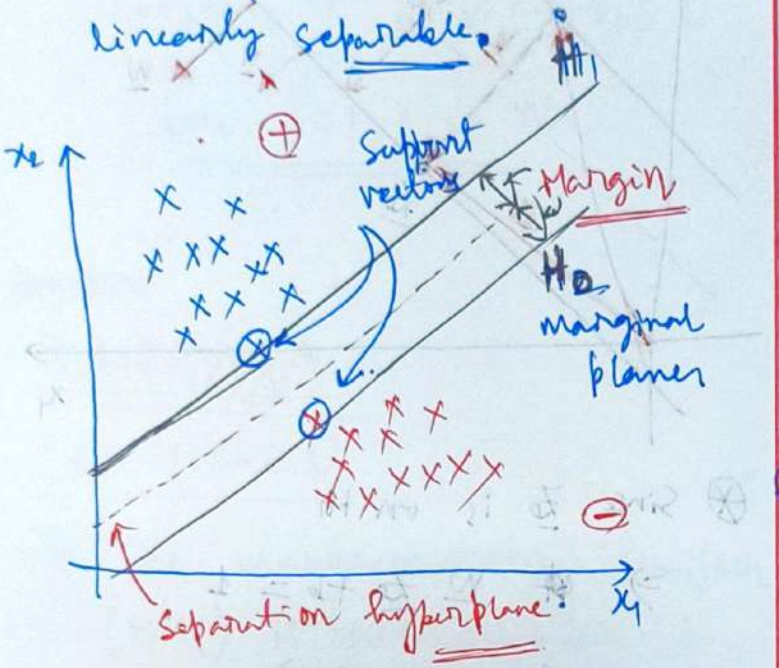
⊛ $\max[\mathcal{L}(\theta)]$

$$\boxed{\underline{\theta}_{n+1} = \underline{\theta}_n + \alpha \frac{\partial \mathcal{L}(\theta)}{\partial \theta}}$$

↳ +ve sign for maximisation

SUPPORT VECTOR MACHINE

- 1992, Vladimir Vapnik
- Can be used for both classification and regression
- Uses a nonlinear mapping function to transforms a data into a feature space that is linearly separable.



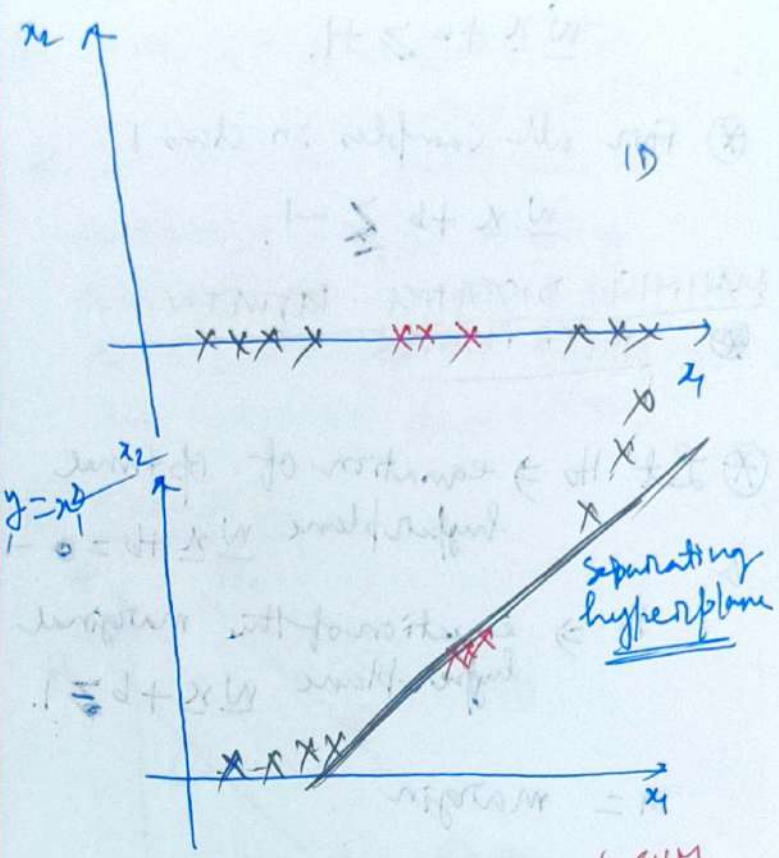
- Support vector machine creates a separation hyperplane with highest margin. (Optimal hyperplane)

dimension	Name of hyperplane
1D	point
2D	line
3D	plane
ND	hyperplane

Nonlinearly Separable cases

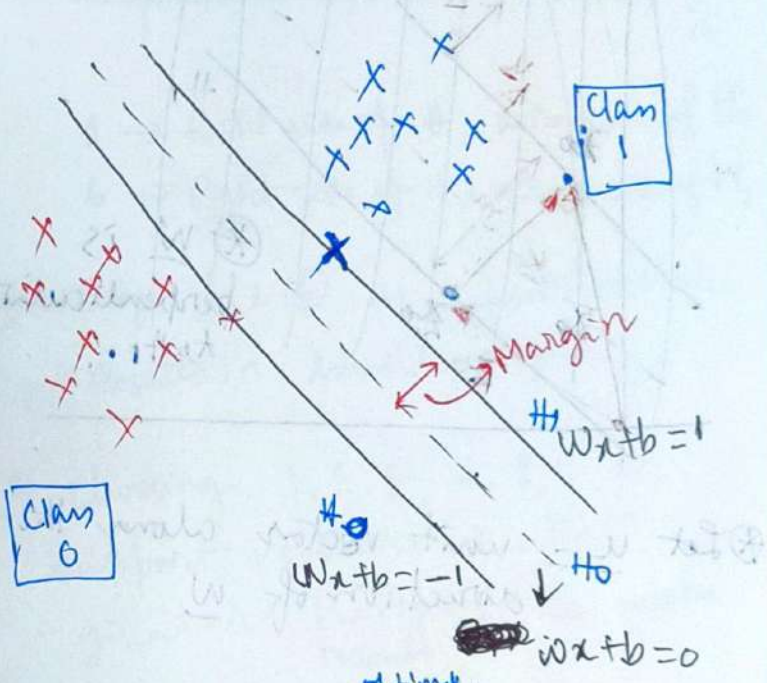
- We can transform the data to higher dimension by using a nonlinear function.
- Kernel SVM.

Example



MATHEMATICAL INTUITION IN SVM

- Consider the following case



- For 2D equation of Optimal plane $\Rightarrow wx+b=0$

equation of Marginal plane

$wx+b=1$

$wx+b=-1$

- For 3D.

Optimal plane $\Rightarrow \underline{w}\underline{x}+b=0$

$\underline{w}\underline{x}+b=1$

$\underline{w}\underline{x}+b=-1$

* For all samples in Class 0

$$\underline{w}\,\underline{x} + b \geq +1.$$

* For all samples in class 1

$$\underline{w}\,\underline{x} + b \leq -1.$$

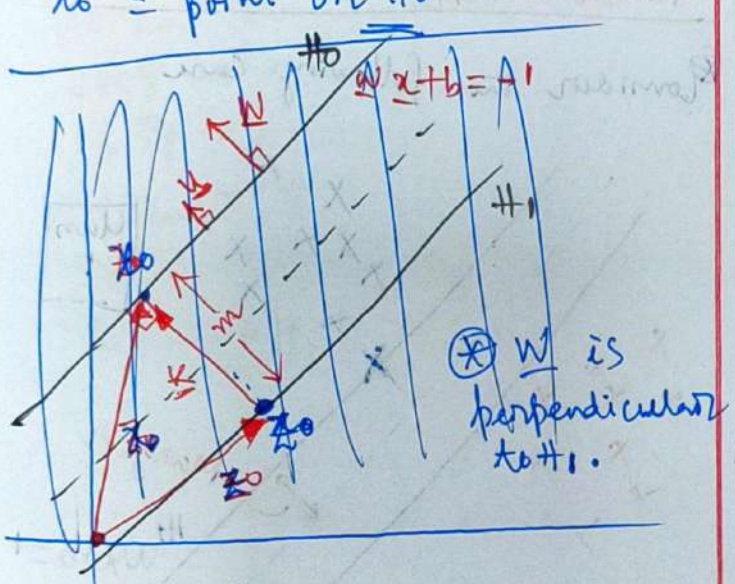
MAXIMISE DISTANCE BETWEEN HYPER PLANES

* Let H_0 ⇒ equation of optimal hyperplane $\underline{w}\,\underline{x} + b = 0$ ... (1)

H_1 ⇒ equation of the marginal hyperplane $\underline{w}\,\underline{x} + b \geq 1$.

m = margin

x_0 = point on H_0



* $\underline{w}$ is perpendicular to H_1.

* Let $\underline{u}$ = unit vector along the direction of $\underline{w}$

$$\Rightarrow \underline{u} = \frac{\underline{w}}{\|\underline{w}\|}$$

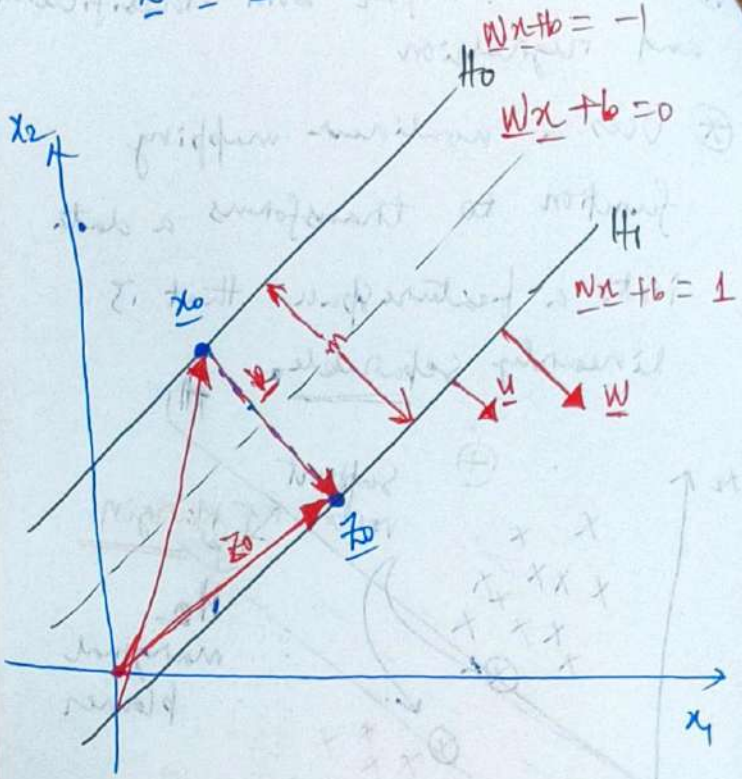
* Thus $m\underline{u}$ will be a vector in the direction of $\underline{w}$.

* $\underline{k} = m\underline{u} = m\,\frac{\underline{w}}{\|\underline{w}\|}$

$\|k\| = m$

* We can see that

$$\underline{z_0} = \underline{x_0} + \underline{k}.$$



* Since $\underline{z_0}$ is on H_1

$$\Rightarrow \underline{w}\,\underline{z_0} + b = 1$$

$$\Rightarrow \underline{w}\left(\underline{x_0} + \frac{m\underline{w}}{\|\underline{w}\|}\right) + b = 1$$

$$\Rightarrow \left[\underline{w}\,\underline{x_0} + m\,\frac{\underline{w}\,\underline{w}}{\|\underline{w}\|}\right] + b = 1$$

$$\Rightarrow \left[\underline{w}\,\underline{x_0} + m\,\frac{\|\underline{w}\|^2}{\|\underline{w}\|}\right] + b = 1$$

$$\Rightarrow \left[\underline{w}\,\underline{x_0} + m\|\underline{w}\|\right] + b = 1$$

$$\Rightarrow \underbrace{\underline{w}\,\underline{x_0} + b}_{=-1 \text{ since } \underline{x_0} \text{ is in } \underline{H_0}} + m\|\underline{w}\| = 1$$

$$\Rightarrow \boxed{m = \frac{2}{\|\underline{w}\|}}$$ ... (2)

* Bigger $\|\underline{w}\|$ ⇒ lower margin

...s to maximize the ...rgin m, we have to minimize $\|W\|$.

⊛ We minimize $\|w\|^2$ (MMC) as it is differentiable

⊛ Optimization step:

Minimize in $(\underline{W}, b)$

$\frac{1}{2}\|\underline{W}\|^2$

Subject to $y_i(\underline{W}.\underline{x_i} + b) \geq 1$

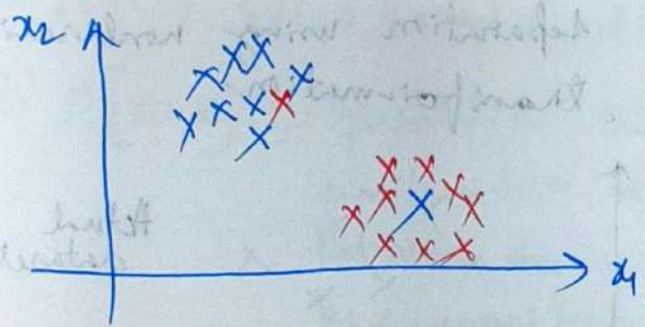
for $i = 1, 2 \ldots N$.

KERNEL SVM

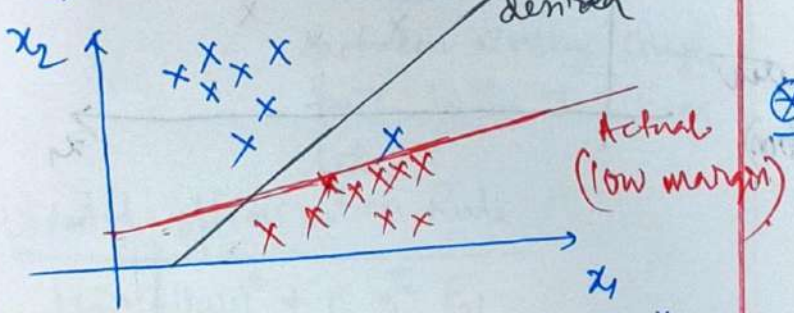
SOFT MARGIN

⊛ The maximum margin classifier (MMC) is not suitable if

ⓐ data is non separable.



ⓑ When the data is noisy

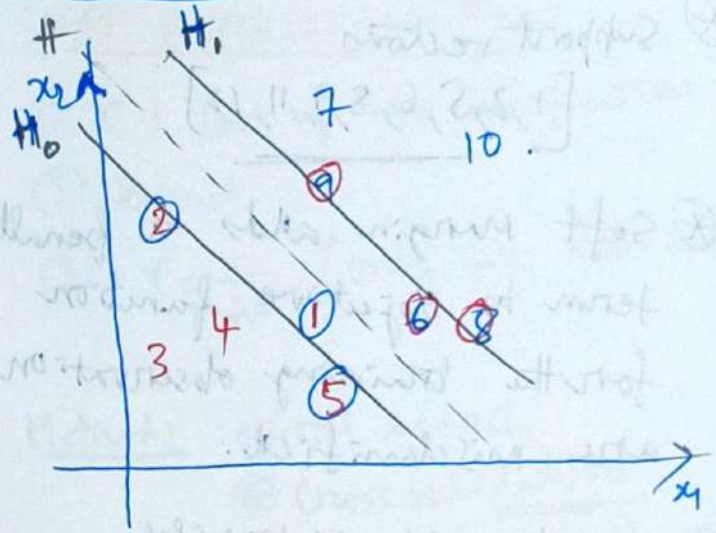


⊛ Solution: we can extend the concept of a separating hyperplane using soft margin.

⊛ MMC + SM ⇒ support vector classifier (SVC)

⊛ It could be worthwhile to misclassify a few training observations in order to do a better job in classifying the remaining observations.

⊛ Example 1



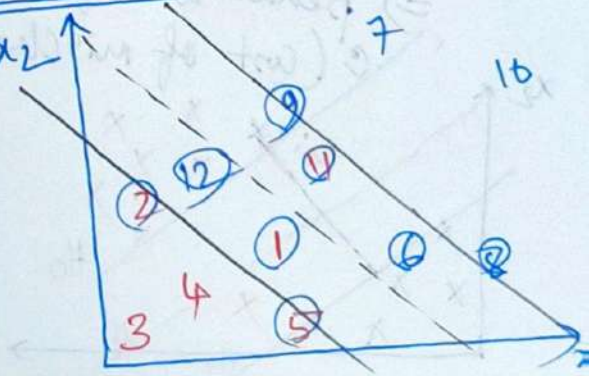
1 → Right side of H, wrong side of H_0

6 → Right side of H, wrong side of H

⊛ Choosing ① & ⑥ as support vector Results in small margin

⊛ Choosing 1, 2, 5, 6, 8, 9 as support vectors results in greater Margin, so more robust.

⊛ Example 2



So here

1 → Wrong side of H_0 → Right side of H

6 → Wrong side of H_1 → Right side of H

11 → Wrong side of H_0 → Wrong side of H

12 → Wrong side of H_1 → Wrong side of H

↓

Misclassified

Support vectors

[1, 2, 5, 6, 8, 9, 11, 12]

Soft margin adds a penalty term to objective function for the training observation are misclassified.

Updated optimization step

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i$$

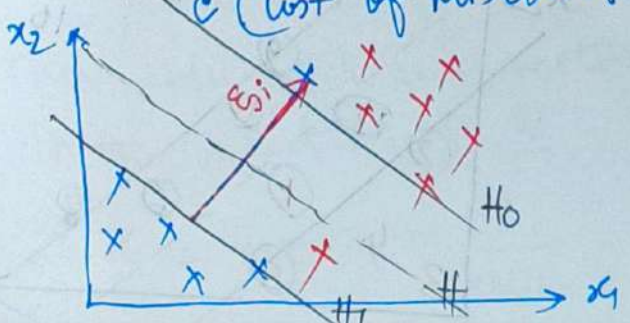
Such that $y_i(\underline{w}\,\underline{x}_i + b) \geq 1 - \xi_i$

$\xi_i \geq 0 \ \forall i$

ξ_i ⇒ Slack parameter

⇒ allows some observations to fall on the wrong side of margin.

⇒ penalizes by a parameter C (cost of misclassification)



$\underline{w}\,\underline{x} + b \geq 1 - \xi_i \quad y_i = +1$

$\underline{w}\,\underline{x} + b \leq -1 + \xi_i \quad y_i = -1$

Small C ⇒ wide margin

high bias, low variance

large C ⇒ narrow margin.

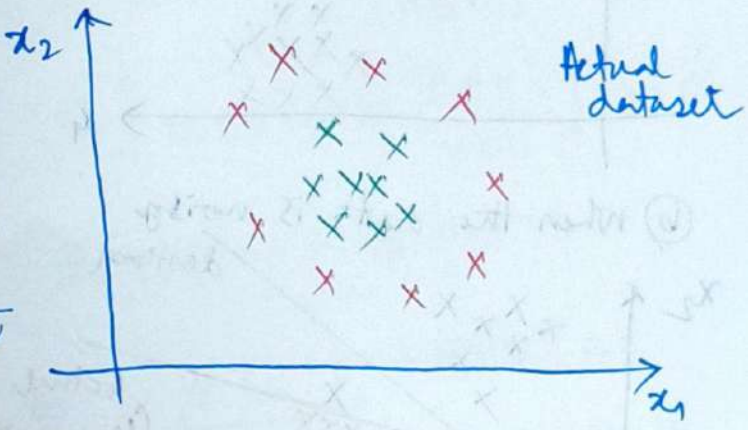
low bias, high variance

KERNEL TRICKS

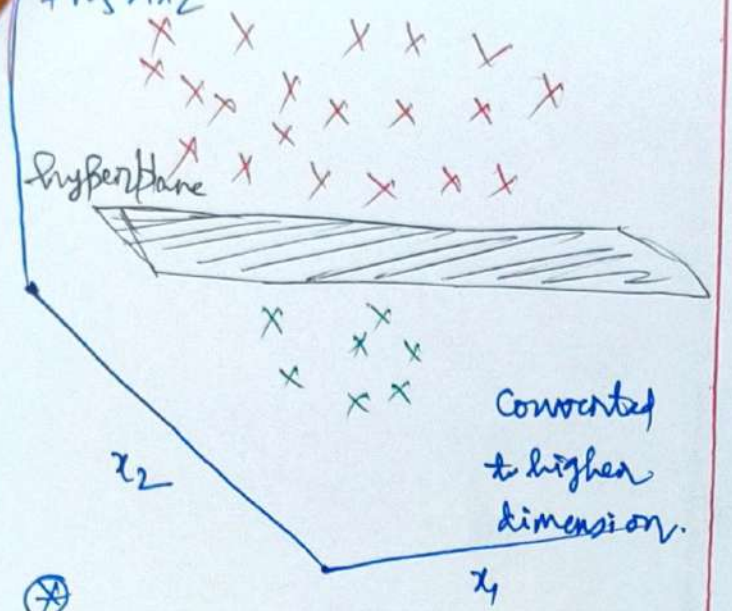
If the dataset is not linearly separated then the data needs to be transformed into higher dimensions, hoping for a linear separability in the higher dimension

works but will require higher computation

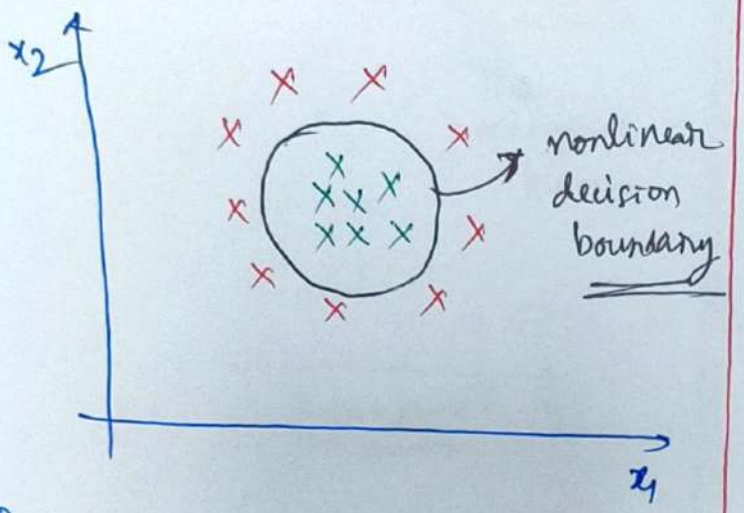
Kernel trick tries to achieve separation using nonlinear transformation.



$+w_1x_1+w_2x_2$
$+w_3x_1^2+w_4x_2^2$
$+w_5x_1x_2$



⊛



⊛ SVM ⇒ SVC + kernel function ϕ

⊛ $\phi[x_i, k]$ = quantifies the similarities of between observations by summarizing the relationship between every single pair in the training set.

Updated optimization Rule

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

such that $y_i\left(\underline{w}\,\underline{\phi}(x) + b\right) \geq 1 - \xi_i$

$\xi_i \geq 0 \quad \forall i$

③

MMC ⇒ Hard Margin (linear kern

SVC ⇒ soft Margin (,,

SVM ⇒ SVC + nonlinear kernel

⊛ Different types of kernels

ⓐ linear
ⓑ Gaussian RBF
ⓒ Polynomial
ⓓ Sigmoid

SVM HYPERPARAMETERS

① C : Cost of misclassification
② Kernel
③ $\gamma = \frac{1}{2\sigma^2}$ (only for RBF kernel)

Methods : ① Grid search
② Cross validation

Kernel	linear, RBF, poly, Sigmoid ④
C	0.1, 1, 10, 100 ... ④
γ	0.001, 0.0, 0.1, 1. ... ④
CV	5/10 .. (no. of folds) ⑤

Total = $4\times4\times4\times5 = 320$

DIMENSIONALITY REDUCTION

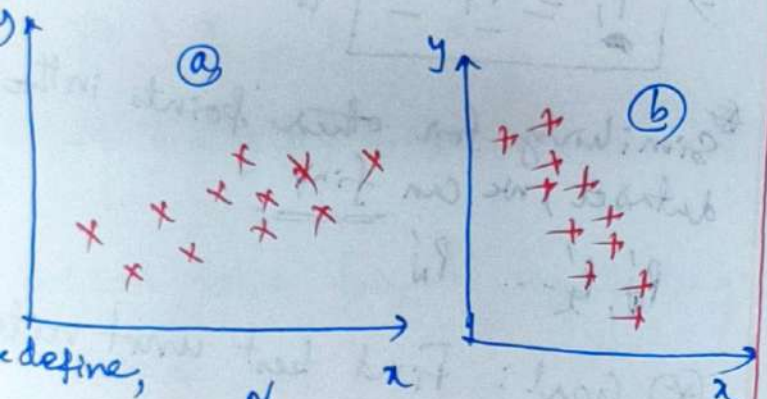
- Dimensionality reduction is used to reduce the number of features without while preserving as much of significant information as possible.
- DR reduces computational burden, simplifies models, reduces overfitting
- Common DR methods

linear patterns:
1. Principal component Analysis (PCA)
2. linear Discriminant Analysis (LDA)

nonlinear patterns:
3. t-distributed stochastic neighbour Embedding
4. Autoencoders.
5. Feature Selection: Chooses a subset of original features.

Feature selection

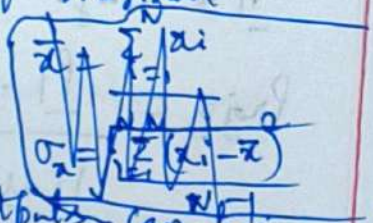
- Input and outputs can have below mentioned correlations.

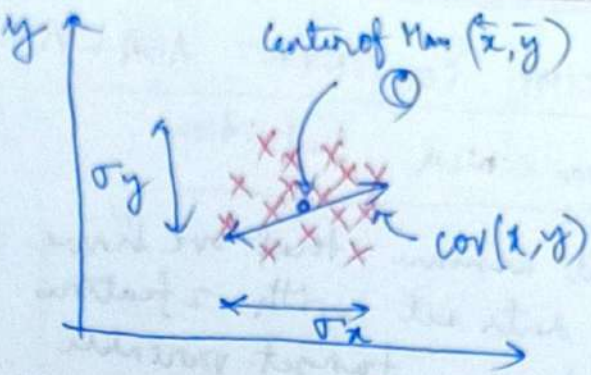


define,

$$cov(x,y) = \frac{\sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

cov(x,y) ~~+ve for fig (a)~~
~~-ve for fig (b)~~





$cov(x,y)$ → +ve for fig (a)
→ -ve for fig (b)
→ ≈0 for fig (c)

$$\text{Pearson Correlation} = \frac{cov(x,y)}{\sigma_x \sigma_y}$$

$$-1 \le PC \le +1$$

- $PC \to +1 \Rightarrow$ more +ve correlation
- $PC \to -1 \Rightarrow$ more -ve correlation
- $PC \approx 0 \Rightarrow$ no correlation

Here $\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} \Rightarrow$ Mean

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N-1}} \Rightarrow SD$$

- Feature selection : Drops the features with low covariance with the target variable.

Feature Extraction

- Feature extraction involves in transforming raw data into a set of features.
- Involves in creation of new features from the original data.

PRINCIPAL COMPONENT ANALYSIS

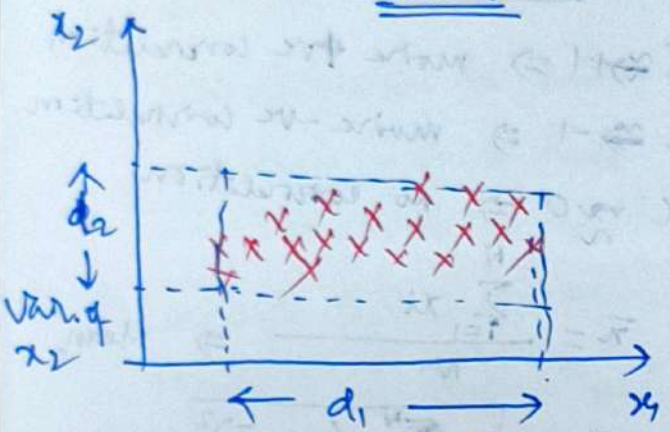
⊛ Geometrical Intuition

⊛ Let assume that we have a dataset with 2 features and one target variable

x_1	x_2	y
:	:	:
:	:	:
:	:	:

⊛ Scatter plot

Case - I

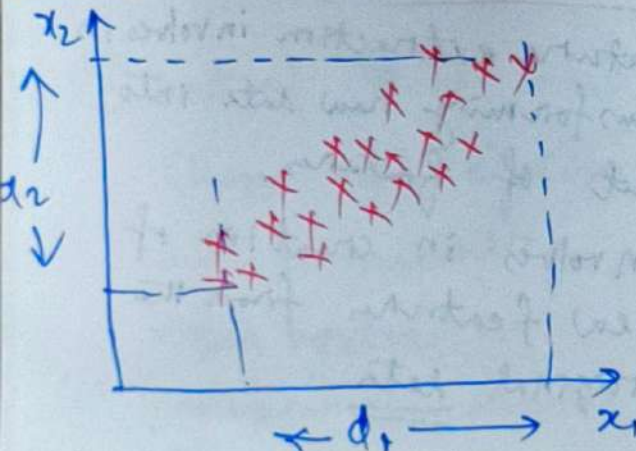


variance of x_1

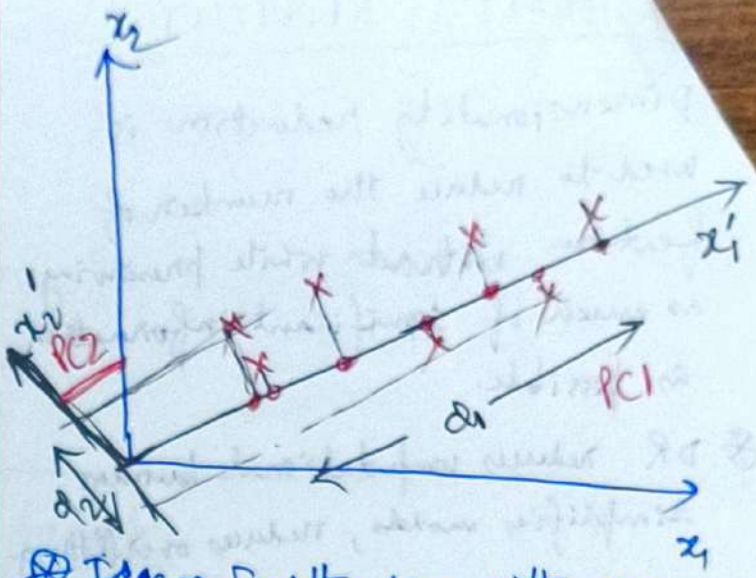
Since $d_1 > d_2 \Rightarrow x_1$ is a better feature than x_2.

⊛ we can drop x_2 (feature selection)

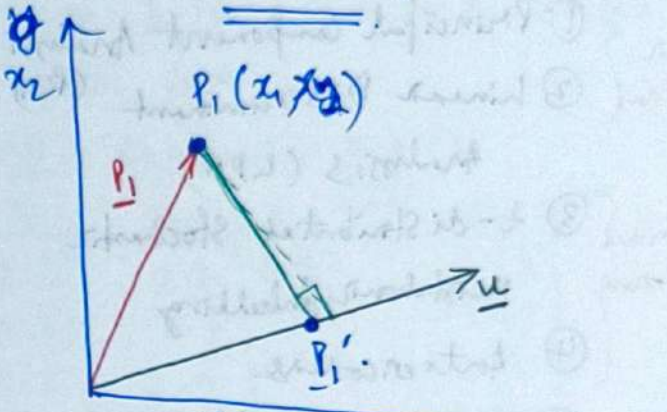
Case - II



$d_1 \approx d_2 \Rightarrow$ which one to drop??



⊛ Idea ⇒ find the line with max variance.



$u \Rightarrow$ unit vector

$P_1' \Rightarrow$ Projection of P_1 on u

$$Proj_{P_1} = \frac{P_1 \cdot u}{\|u\|} = P_1 \cdot u$$

$$\Rightarrow \boxed{P_1' = P_1 \cdot u} \quad \text{W}$$

⊛ Similarly for other points in the dataset, we can find.

$P_1', P_2' \ldots P_N'$

⊛ Goal: Find best unit vector that captures maximum variance in $P_1', P_2' \ldots P_N'$

⊛ we can find out the same by using Eigen vectors & Eigen values.

VALUES AND EIGEN VECTORS

Let $\underline{A}$ is an $N \times N$ matrix

* A scalar λ is called eigen value of $\underline{A}$ ~~corresponding to~~ such that

$$\underline{A}\,\underline{x} = \lambda \underline{x}$$

$A \rightarrow$ Matrix
$x \rightarrow$ Eigen vector
$\lambda \rightarrow$ Eigen value

Example ①

$$\underline{A} = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix}, \quad \underline{x} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

$\lambda = 6$

$$\underline{A}\,\underline{x} = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix}\begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 24 \end{bmatrix}$$

$$= 6\begin{bmatrix} 1 \\ 4 \end{bmatrix} = 6\,\underline{x}$$

* **How to find Eigen values and eigen vectors :**

$$\underline{A}\,\underline{x} = \lambda \underline{x}$$

$$\Rightarrow \underline{A}\,\underline{x} = \lambda \cdot \underline{I} \cdot \underline{x}$$

$$\Rightarrow \left[\underline{A} - \lambda \underline{I}\right] = 0$$

We can solve for λ by setting

$$\det\left[\underline{A} - \lambda \underline{I}\right] = 0$$

Example : 2

* Find $\det\left(\underline{A} - \lambda \underline{I}\right)$

$$\left|\begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix} - \lambda\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right| = 0$$

$$\Rightarrow \begin{vmatrix} -6-\lambda & 3 \\ 4 & 5-\lambda \end{vmatrix} = 0$$

$$\Rightarrow (-6-\lambda)(5-\lambda) - 12 = 0$$

$$\Rightarrow -30 + 6\lambda - 5\lambda + \lambda^2 - 12 = 0$$

$$\Rightarrow \lambda^2 + \lambda - 42 = 0$$

$$\Rightarrow \lambda + 7\lambda - 6\lambda - 42 = 0$$

$$\Rightarrow (\lambda + 7)(\lambda - 6) = 0$$

$$\Rightarrow \lambda = -7 \text{ or } \lambda = +6$$

Eigen vector for $\lambda = 6$

$$\underline{A}\,\underline{x} = \lambda \underline{x}$$

$$\Rightarrow \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 6\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} -6x_1 + 3x_2 \\ 4x_1 + 5x_2 \end{bmatrix} = \begin{bmatrix} 6x_1 \\ 6x_2 \end{bmatrix}$$

$-6x_1 + 3x_2 = 6x_1 \Rightarrow 12x_1 + 3x_2 = 0$

$4x_1 + 5x_2 = 6x_2 \Rightarrow 4x_1 - x_2 = 0.$

$\Rightarrow x_2 = 4x_1.$

Thus $\underline{x} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$.

Covariance Matrix

for 2 features

$$\Sigma = \begin{bmatrix} \text{Var}(x_1) & \text{Cov}(x_1, x_2) \\ \text{Cov}(x_1, x_1) & \text{Var}(x_2) \end{bmatrix}$$

for 3 features

$$\Sigma = \begin{bmatrix} \text{Var}(x_1) & \text{Cov}(x_1, x_2) & \text{Cov}(x_1, x_3) \\ \text{Cov}(x_1, x_2) & \text{Var}(x_2) & \text{Cov}(x_2, x_3) \\ \text{Cov}(x_1, x_3) & \text{Cov}(x_2, x_3) & \text{Var}(x_3) \end{bmatrix}$$

STEPS IN PCA

① Standardize the data.

$$X_{std} = \frac{X - \mu}{\sigma}$$

⇒ each feature has zero mean & unit var.

② Compute the covariance matrix $\Sigma = \left(\frac{1}{N-1}\right) X_{std}^T X_{std}$

③ Calculate the eigen values and eigen vectors of the covariance matrix Σ.

$$\Sigma \underline{x} = \lambda \underline{x}$$

④ Sort the eigen values and eigen vectors

⑤ out of 'N' eigen values Choose top 'k' eigen values and corresponding eigen vectors.

⑥ Transform the original data by using the selected eigen vectors (Principal components)

$$X' = X_{std} \cdot \underline{X}_K$$

$\underline{X}_K \Rightarrow$ Matrix of top k eigen vectors.

Example

Let the dataset of 2 features is given below. Compute the 1D PCA of the same.

Step 1: Standardize x_1 & x_2

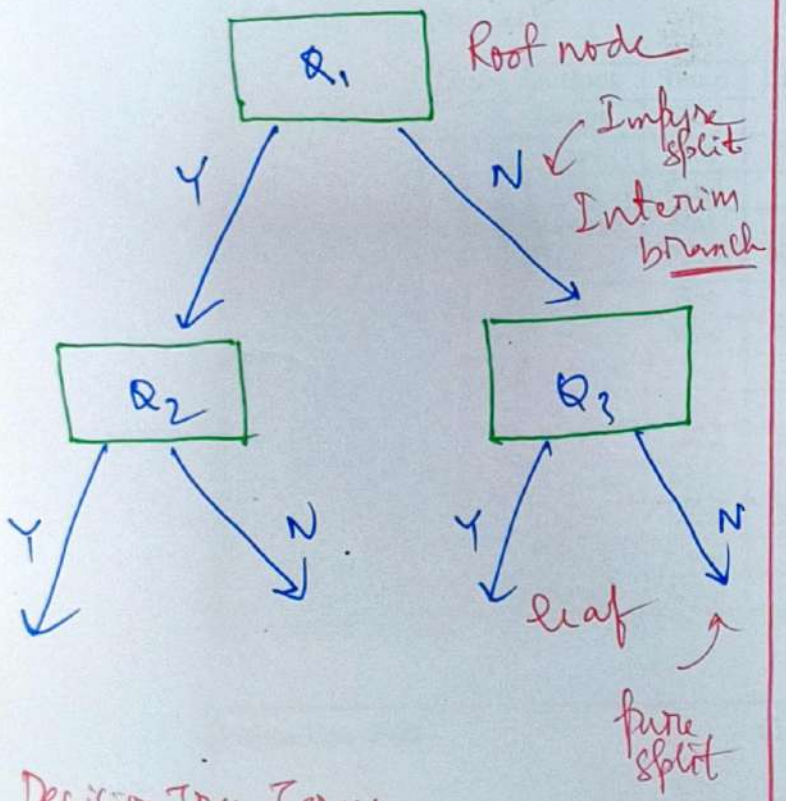
x_1	x_2	x_1 (Std)	x_2
2.5	2.4	0.87	0.58
0.5	0.7	-1.66	-1.42
2.2	2.9	0.49	1.16
1.9	2.2	0.11	0.34
3.1	3.0	1.63	1.28
2.3	2.7	0.62	0.93
2.0	1.6	0.24	-0.36
1.0	1.1	-1.03	-0.95
1.5	1.6	-0.39	-0.36
1.1	0.9	-0.90	-1.19

	X_{std}
$\bar{x}_1 = 1.81$	0
$\bar{x}_2 = 1.91$	0
$\sigma_1 = 0.79$	≈ 1
$\sigma_2 = 0.85$	≈ 1

DECISION TREES

- Decision trees are a class of powerful ML algorithms used for both classification and prediction.
- Decision trees are Interpretable models, since they work like if-else rules.
- DTs are used to classify / predict the output variable by using boolean (usually, but more options can also be employed).

Decision Tree Structure



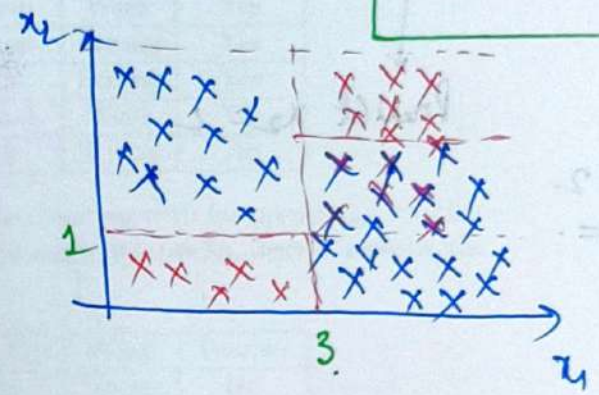
Decision Tree Terms

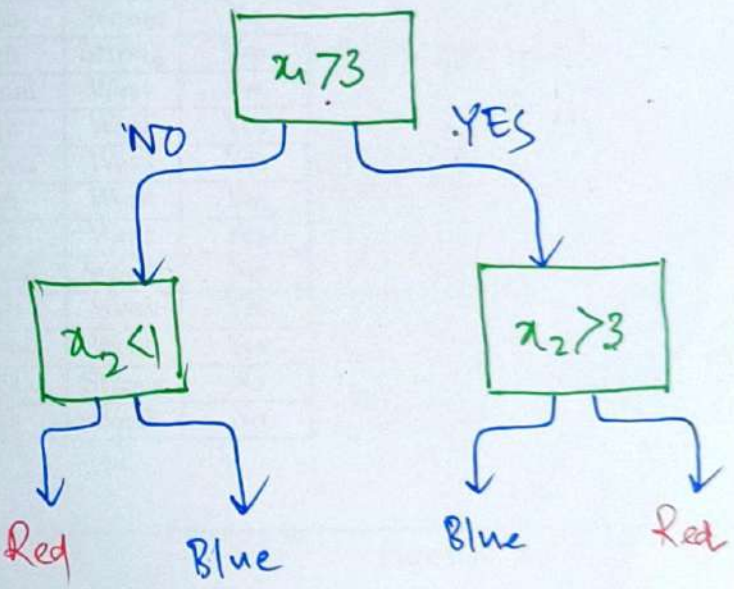
1. Root Node: node at the top of the tree
- Leaf / Terminal Node: Nodes at the bottom which do not get splitted further.
- Branch / Subtree: A subsection of the entire tree.
- Splitting: process of dividing 1 node to 2 (or more) sub nodes

Divide & conquer
→ split into subsets
→ pure? Y → stop
N → repeat.

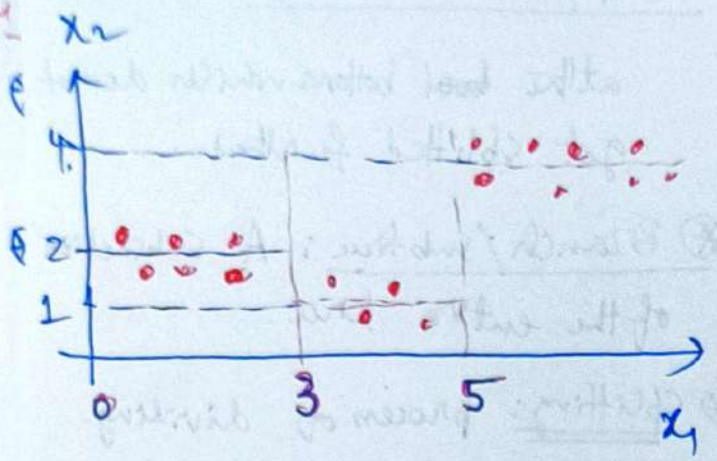
Example:

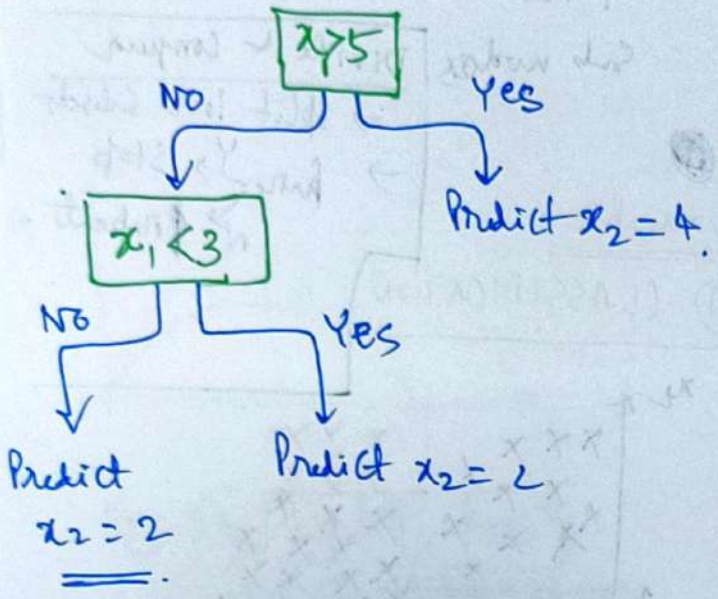
1. CLASSIFICATION





REGRESSION


x_2
4
2
1
0
3
5
x_1


$x_1 < 5$
NO
Yes
Predict $x_2 = 4$.
$x_1 < 3$
NO
Yes
Predict $x_2 = 2$
Predict $x_2 = 2$

We will use the following example as a running example in this unit.

Example: Jeeves is a valet to Bertie Wooster. On some days, Bertie likes to play tennis and asks Jeeves to lay out his tennis things and book the court. Jeeves would like to predict whether Bertie will play tennis (and so be a better valet). Each morning over the last two weeks, Jeeves has recorded whether Bertie played tennis on that day and various attributes of the weather (training set).

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

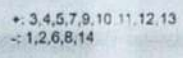
Jeeves would like to evaluate the classifier he has come up with for predicting whether Bertie will play tennis. Each morning over the next two weeks, Jeeves records the following data (test set).

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Mild	High	Strong	No
2	Rain	Hot	Normal	Strong	No
3	Rain	Cool	High	Strong	No
4	Overcast	Hot	High	Strong	Yes
5	Overcast	Cool	Normal	Weak	Yes
6	Rain	Hot	High	Weak	Yes
7	Overcast	Mild	Normal	Weak	Yes
8	Overcast	Cool	High	Weak	Yes
9	Rain	Cool	High	Weak	Yes
10	Rain	Mild	Normal	Strong	No
11	Overcast	Mild	High	Weak	Yes
12	Sunny	Mild	Normal	Weak	Yes
13	Sunny	Cool	High	Strong	No
14	Sunny	Cool	High	Weak	No

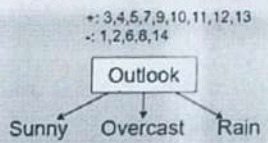
Problem: Construct a (full) decision tree for the Jeeves data set using the following order of testing features.

- First, test Outlook.
- For Outlook = Sunny, test Temp.
- For Outlook = Rain, test Wind.
- For other branches, test Humidity before testing Wind.

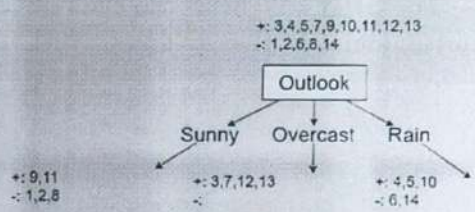
Solution: Here is the process to generate the decision tree by the given order.



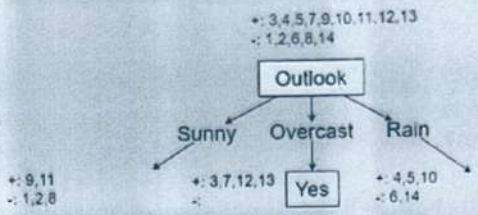
We have 9 positive and 5 negative examples. They are not in the same class, so we will have to choose a feature to test.



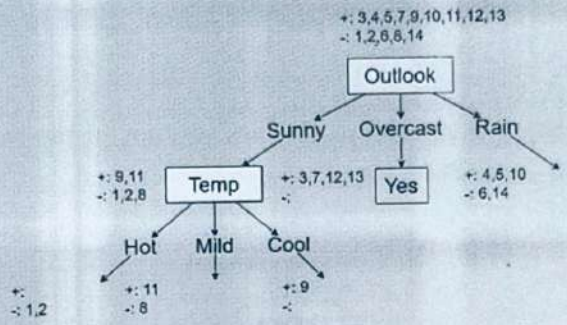
Based on the given order, we will test Outlook first. Outlook has three values: Sunny, Overcast, and Rain. We split the examples into three branches.



The three sets look like this. Example 1 has Outlook equal to Sunny, so it goes into the left branch. Example 3 has Outlook equal to Overcast, so it goes into the middle branch, etc.

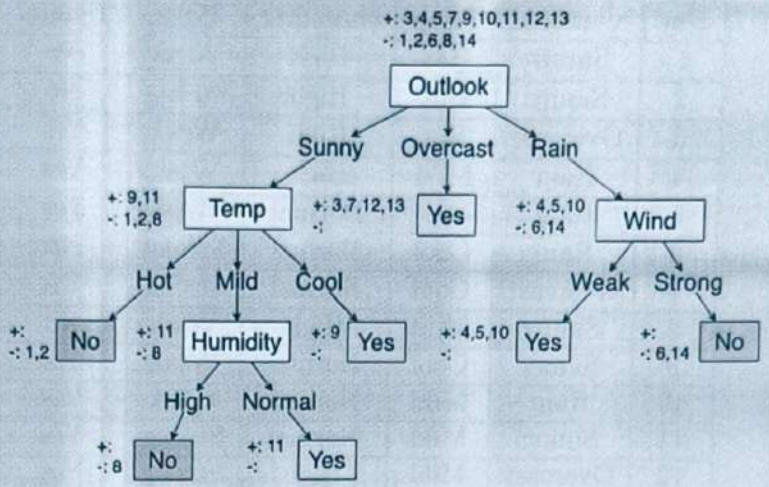


In the middle branch, all the examples are positive. There is no need to test another feature, and so we may make a decision. We create a leaf node with the label Yes, and we are done with this branch.



Looking at the left branch next, there are two positive and three negative examples. We have to test another feature. Based on the given order, we will test Temp next. Temp has three values — Hot, Mild, and Cool. We create three branches again. The five examples are split between these branches.

We will repeat this process at every node. First, check if all the examples are in the same class. If they are, create a leaf node with the class label and stop. Otherwise, choose the next feature to test and split the examples based on the chosen feature.



The above is the final decision tree. Each internal node represents a test—Not all of these are Boolean. Beside each node, the training examples are partitioned into two classes based on whether Bertie played tennis that day: Yes (+) and No (−).

4.3 When do we stop?

There are three possible stopping criteria for the decision tree algorithm. For the example in the previous section, we encountered the first case only: when all of the examples belong to the same class. In this case, we make the decision of that class and then we're done.

4.4 Base case 2: no features left

Let's look at the second case: what should we do if there are no more features to test?

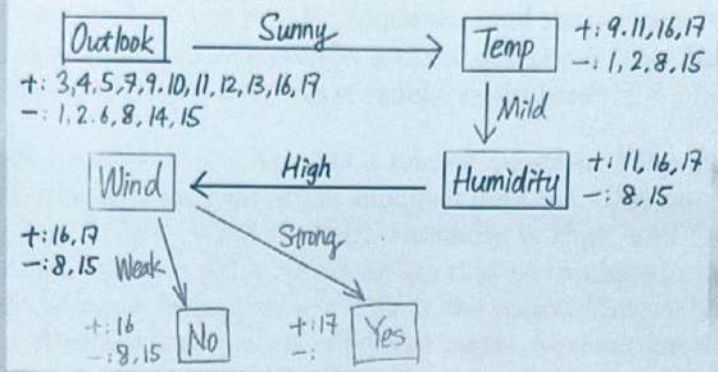
Problem: I took our training set and added a few examples. It now has 17 instead of 14 examples. For this modified training set, let's construct one branch of the decision tree where Outlook is Sunny, Temperature is Mild, and Humidity is High.

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No
15	Sunny	Mild	High	Weak	No
16	Sunny	Mild	High	Weak	Yes
17	Sunny	Mild	High	Strong	Yes

Solution: After testing Humidity is High, what should we do next? We have tested three of the four input features. To continue, testing Wind is the only option. When Wind is Strong, we have one positive example, and the decision is Yes. When Wind is Weak, we have three examples: one positive and two negative examples. The examples are mixed, so we cannot make a decision. But, we have tested all four features — there's no more feature to test. What should we do in this case?

Let's take another look at our data set. There are three examples when Wind is Weak. Note that, for the three examples, the values of all input features are all the same — Sunny, Mild, High, and Weak, but they have different labels, No for 8 and 15 and Yes for 16. This is an example of a noisy data set. With noisy data, even if we know the values of all the input features, we are still unable to make a deterministic decision. One reason for having a noisy data set is that the decision may be influenced by some features that we do not observe. Perhaps, another factor not related to weather influences Bertie's decision, but we don't know what that factor is.

What should we do when we run out of features to test? There are a few options. One option is to predict the majority class. In this case, the majority decision is No. Another option is to make a randomized decision. We can think of the three examples as a probability distribution. There is a 1/3 probability of predicting Yes and a 2/3 probability of predicting No. We will make the decision based on a random draw from the distribution. For this example, let's use the majority decision.



4.5 Base case 3: no examples left

Let's look at the last possible stopping criteria: what should we do if there are no examples left?

Problem: Let's consider another modified Jeeves training set. Complete one branch of the decision tree where Temperature is Hot, Wind is Weak, and Humidity is High.

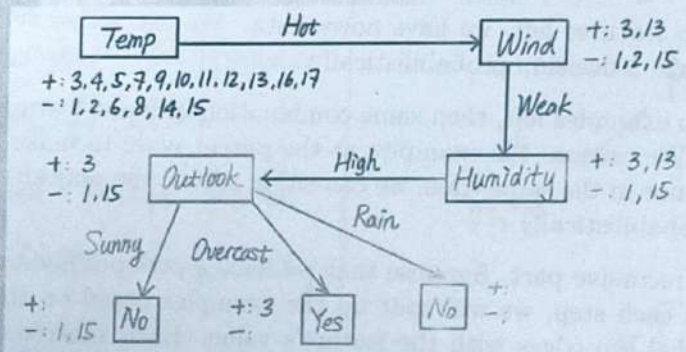
Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No
15	Sunny	Hot	High	Weak	No

Solution: After testing the three features, we have three examples left: one positive and two negative. The only feature left is Outlook. Outlook has three values. Let's split up the examples into three branches.

If Outlook is Sunny, we have two negative examples, and the decision is No. If Outlook is Overcast, we have one positive example, and the decision is Yes. Finally, if Outlook is Rain, there are no examples left. What should we do here?

Before we decide on what to do, let's ask a related question. Why did we encounter this case? Let's take another look at the modified data set. The case we are looking for is: Temperature is Hot, Wind is Weak, Humidity is High, and Outlook is Rain. After going through the data set, you will realize that no example in this data set has this combination of input feature values. This is the reason that we had no examples left. If we never observe a combination of feature values, we don't know how to predict it.

Now that we understand why this happened, how should we handle this case? One idea is to try to find some examples that are close to this case. If we go up to the parent node, the parent has some examples for different values of Outlook. Arguably, these are the closest examples to our case. Therefore, we can use the examples at the parent node to make a decision. At the parent node, the examples are likely to be mixed. Most likely, we cannot make a deterministic decision. Similar to the previous case, we can either make the majority decision or decide based on a random draw from a probability distribution.



4.6 Pseudo-code for the decision tree learner algorithm

Algorithm 1 Decision Tree Learner (examples, features)

```
1: if all examples are in the same class then
2:     return the class label.
3: else if no features left then
4:     return the majority decision.
5: else if no examples left then
6:     return the majority decision at the parent node.
7: else
8:     choose a feature f.
9:     for each value v of feature f do
10:        build edge with label v.
11:        build sub-tree using examples where the value of f is v.
```

Here is the pseudocode for the algorithm. Since a tree is recursive, we will naturally use a recursive algorithm to build it.

The algorithm starts with three base cases.

1. If all the examples are in the same class, we will return the class label.
2. If there are no features left, we have noisy data. We can either return the majority decision or make a decision probabilistically.
3. If there are no examples left, then some combination of input features is absent in the training set. We can use the examples at the parent node to make a decision. If the examples are not in the same class, we can either return the majority decision or make a decision probabilistically.

Next, we have the recursive part. Suppose that we have a pre-specified order of testing the input features. At each step, we will split up the examples based on the chosen feature's values. We will label the edges with the feature's value. Each subtree only has examples where the value of the feature corresponds to the value on the edge.

There's one crucial step left. So far, we have assumed that a pre-defined order of testing the input features. Where does this order come from? In practice, we have to choose this order ourselves.

MINE THE ORDER OF TING FEATURES

complex trees ⇒ overfitting

- objective ⇒ Build a simple & shallow tree.

- Need for finding optimal order of testing features. which will minimize the tree size ⇒ complex, computationally expensive

- Alternative Approach:
  At each step:
  - Choose a feature that makes the biggest difference to the classification. (feature that helps us to make a decision as quickly as possible).

- Choosing best feature ⇒ Reduce uncertainity.

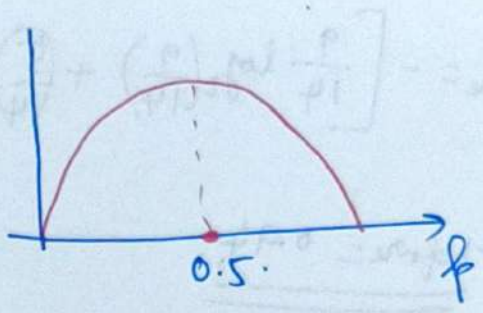
- Measure of uncertainity

Entropy: I (information content in an event)

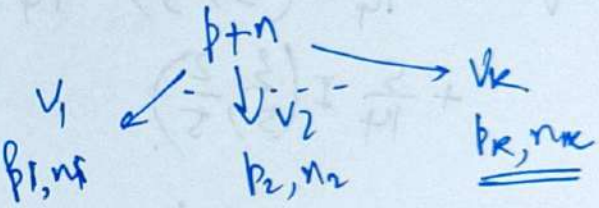
$$I = -\sum_{i=1}^{k} P(c_i) \log_2 [P(c_i)], \text{ bits}$$

$\{c_i\}$ ⇒ outcome.
$\{P(c_i)\}$ ⇒ probability of occurrence of c_i

- Certain events have low entropy
- uncertain events have high entropy.



- Expected information gain of testing a feature.
  - Consider k features $\{V_1, \ldots V_k\}$.
  - $\begin{cases} p \Rightarrow +ve \text{ values in } y \\ n \Rightarrow -ve \text{ values in } y \end{cases}$



$$I_{before} = I\left[\frac{p}{p+n}, \frac{n}{p+n}\right].$$

  - For each feature find the Expected entropy as follows:

$$E I_{after} = \sum_{i=1}^{k} \left(\frac{p_i+n_i}{p+n}\right) I\left[\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right].$$

  - Information Gain

$$IG = I_{before} - E I_{after}.$$

EXAMPLE

14 examples: 9p, 5n

$$I_{before} = -\left[\frac{9}{14}\log_2\left(\frac{9}{14}\right) + \frac{5}{14}\log_2\left(\frac{5}{14}\right)\right]$$

$\Rightarrow I_{before} = 0.94$

E_I after $= \frac{5}{14} I\left(\frac{2}{5}, \frac{3}{5}\right) + \frac{4}{14} I\left(\frac{4}{4}, \frac{0}{4}\right) + \frac{5}{14} I\left(\frac{3}{5}, \frac{2}{5}\right)$

outlook
- Sunny (2p, 3n)
- overcast (4p, 0n)
- Rain (3p, 2n)

$\Rightarrow E_I$ after $= 0.694$

G(outlook) $= 0.94 - 0.694 = 0.247$

Similarly

G (Humidity) $= 0.151$

G (Temp) $= 0.029$

G (wind) $= 0.048$

(*) Highest information gain feature $\Rightarrow$ outlook

(*) outlook is chosen as Root node.

outlook = Sunny

Gain (temp) = 0.57

Gain (Humidity) = 0.97.

Gain (wind) = 0.019

(*) we pick humidity under Sunny.

outlook = overcast

4p, 0N $\Rightarrow$ leaf node

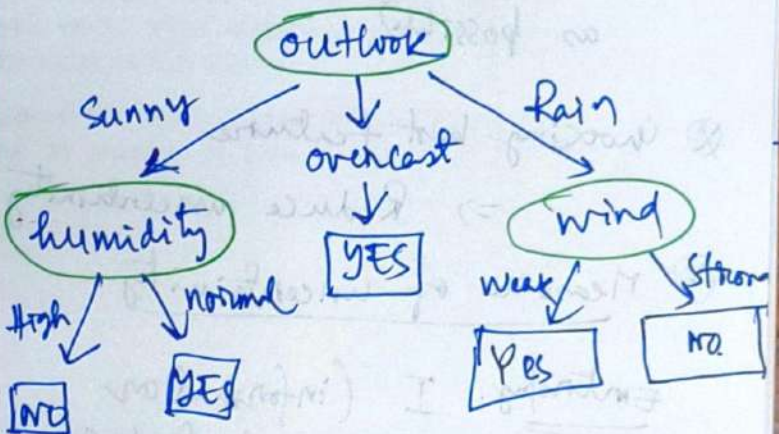
outlook = Rain

Gain (Temp) = 0.019

Gain (Humidity) = 0.019

Gain (wind) = 0.97.

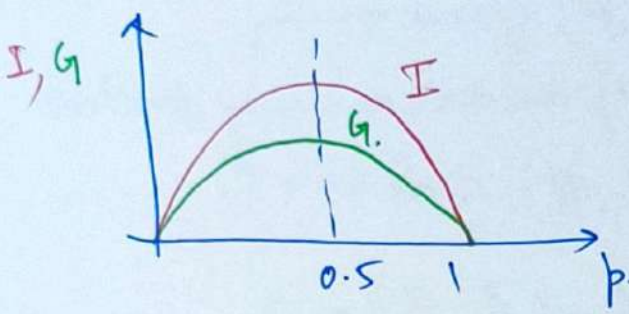
(*) we pick wind under Rain



GINI INDEX (IMPURITY)

- Used to measure the purity of a decision tree.
- evaluates how often a randomly chosen element would be incorrectly

$$Gini = 1 - \sum_{i=1}^{N} P(i)^2$$



$G = 0 \Rightarrow$ perfect purity

~~all elements belong to leaf node~~ ⇒ all elements are of same class

$G = 0 \Rightarrow$ perfect purity

↓

all elements are of same class

↓

leaf node ✓

- High Gini Index ⇒ higher Impurity
- Gini Index is a simpler expression ⇒ thus faster.

TYPES of DTs

TYPE	PURPOSE	
Classification tree	categorical outcome	
Regression Tree	continuous outcome	
CART (Classification & Regression tree)	Can be used for both Classification & Regression	Classification ↳ Gini Index Regression ↳ least Squares deviation
ID3 (Iterative dichotomiser 3)	Classification	entropy & information gain
C4.5	Improved ID_3	Handles both categorical & numerical data Gain Ratio
CHAID (Chi squared Automatic Interaction detector)	Classification & Regression	uses Chi-squared statistics.

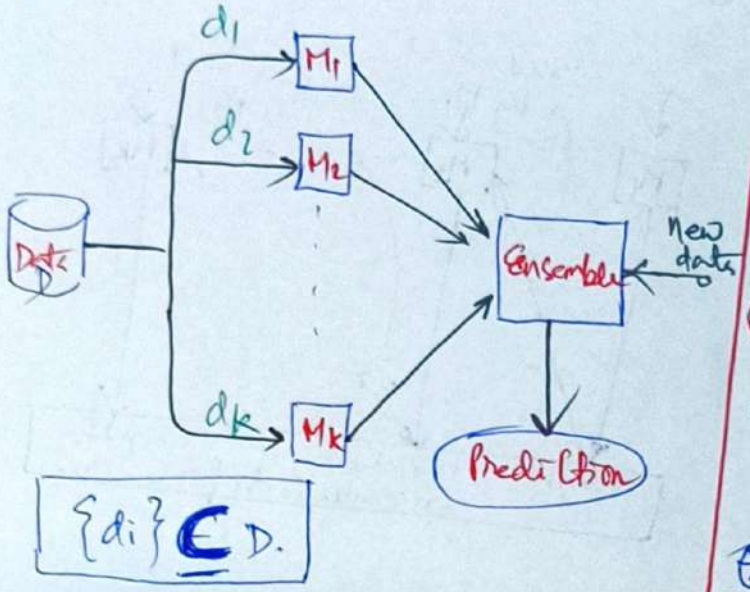
Gain Ratio

$$GR = \frac{\text{Information Gain}}{\text{Entropy}}$$

- Small entropy ⇒ High GR
- High entropy ⇒ low GR

ENSEMBLE TECHNIQUES

* Ensemble models use a combination of models to improve the overall performance.



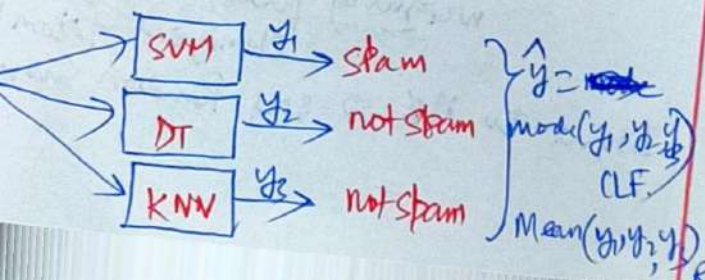
Types of Ensemble

* There are four types of ensemble techniques
  - (a) Bagging
  - (b) Boosting
  - (c) Stacking
  - (d) voting.

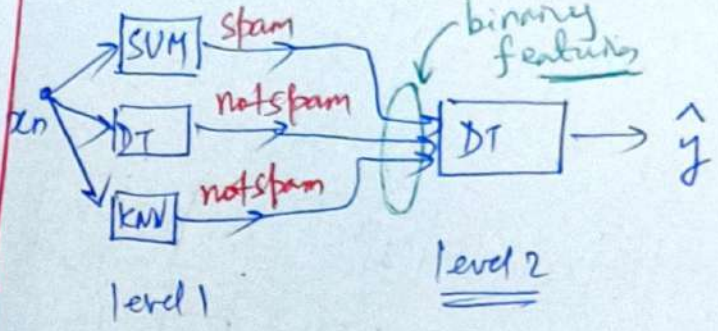
Majority voting

→ Classification → voting
→ Regression → Averaging.

* Example ⇒ whether email is spam or not spam



(b) Stacking



* Use predictions of multiple models as features to new model

Bagging

* ~~Short~~ Short form for bootstrap aggregation.
* Original data D
  no. of samples N
* Create k copies of D as $\{\tilde{D}_i\}_{i=1}^{k}$
  - Each $\{\tilde{D}_i\}$ is generated from D by sampling with Replacement.
  - no. of samples in $\tilde{D}_i = N$
  - The data sets are logically different from each other.
  - Train models $M_1, M_2 \ldots M_k$ using $\tilde{D}_1, \tilde{D}_2 \ldots \tilde{D}_k$.
  - Use majority voting (classification) or averaging (Regression.
  - Useful for high variance &

Original Dataset

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8

Bootstrap 1

x_8	x_6	x_2	x_6	x_8	x_4	x_2	x_6

x_1	x_5	x_3

Bootstrap 2

x_4	x_5	x_3	x_4	x_7	x_5	x_3	x_1

x_8	x_2	x_6

Training set

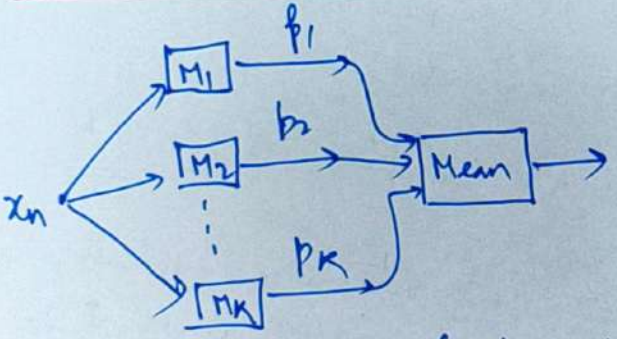
Test set

Advantages & disadvantages

- Reduces overfitting (reduces variance)
- easy to parallelize
- looses interpretability compared to singe DT.

RANDOM FORESTS

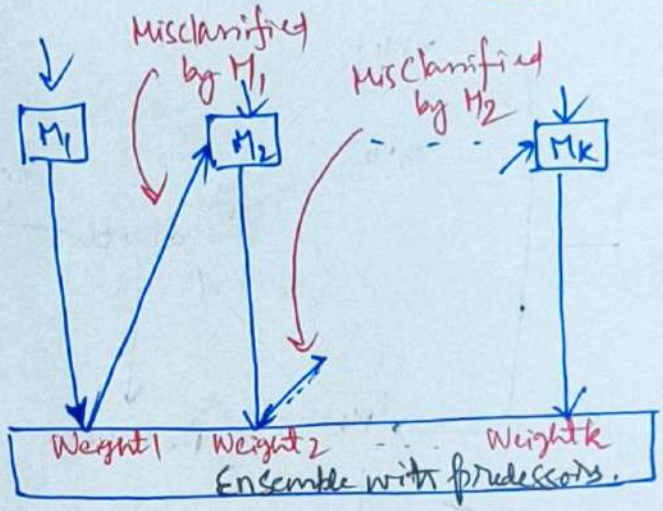
MAJORITY VOTING (SOFT VOTING)



$\{p_i\}$ ⇒ probability (not decision)

BOOSTING

- The general idea of boosting methods is to train predictors sequencially, each trying to correct its predecessor.
- Reduces bias and variance.



STEPS

1. Initialize weights
2. Train a weak learner
3. Calculate error.
4. update weights.
5. Combine weak learners.
   - final output by majority voting.

Input to M_1 ⇒ 1st Random Subset without Replacement.

Input to M_2 ⇒ 2nd Random Subset weighted + 50% examples that were mis-classified from M_1.

Input to M_K ⇒ 3rd Random Subset

+