Machine Learning

Machine learning is a subset of artificial intelligence (AI) that enables computers to learn from data and improve their performance over time without being explicitly programmed. Imagine teaching a child to recognize different types of fruits by showing them examples. Similarly, in ML, we train a computer model using data (like pictures of fruits) so it can learn to recognize patterns and make decisions (like identifying fruits).

Types of Machine Learning:

1. **Supervised Learning:** This is like learning with a teacher. You provide the model with data that is labeled. For example, you give it pictures of fruits with their names, and it learns to associate the image with the fruit.
2. **Unsupervised Learning:** Here, the learning is without a teacher. The model is given data but without labels. It tries to find patterns or groupings on its own. For example, giving a bunch of fruit images and letting the model categorize them without knowing their names.
3. **Reinforcement Learning:** This is like learning by trial and error. The model learns by making decisions and receiving feedback based on those decisions. Think of it as teaching a dog new tricks and rewarding it with treats for correct actions.

Difference between Batch Learning and Incremental Learning:

Batch Learning:

In batch learning, the system is trained using the entire training data set at once. This is a time-consuming process and requires a lot of memory and computational resources. Once trained, the model doesn't learn from new data unless it's retrained with a new batch including the old and new data. It's useful when the data is static and doesn't change over time.

Incremental Learning (Online Learning):

Incremental learning, on the other hand, allows the model to learn continuously as new data comes in. The model updates its parameters with each new data point or small batches of data points. This approach is more flexible and can adapt to changing data over time. It's ideal for situations where data is continuously generated or changes frequently.

Difference between Inductive Learning and Deductive Learning:

Inductive Learning:

Inductive learning is about learning general patterns from specific examples. It's like observing several specific instances and then making a broader generalization. Most machine learning, especially supervised learning, is inductive – you train a model on specific examples, and it learns to generalize from these to make predictions on unseen data.

Deductive Learning:

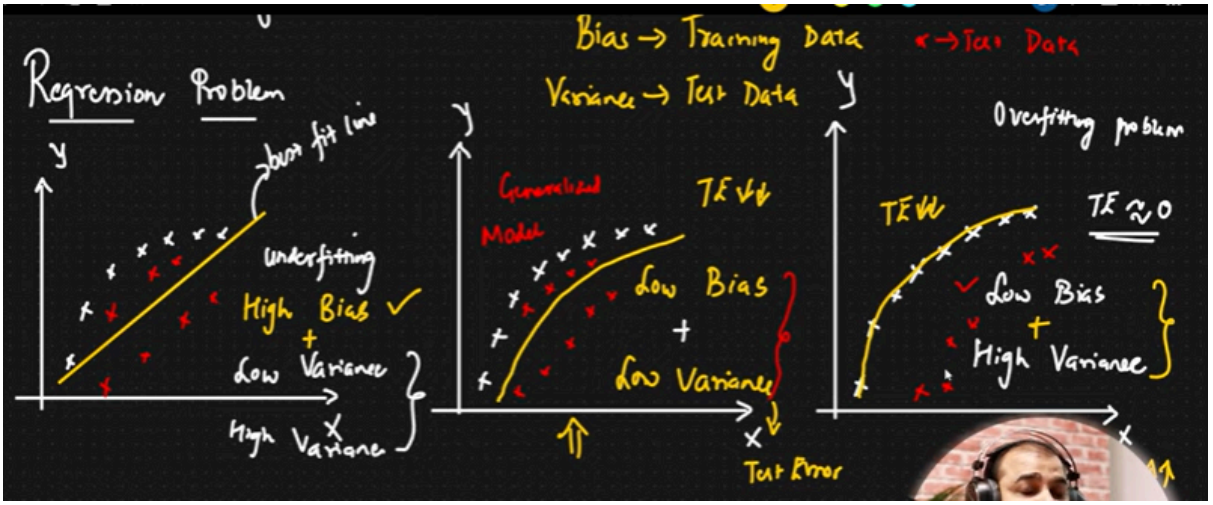
Deductive learning is the reverse process. It starts with a general rule or hypothesis and deduces specific facts or conclusions from it. It's more like traditional logic or programming, where you apply general rules to specific situations to deduce outcomes.

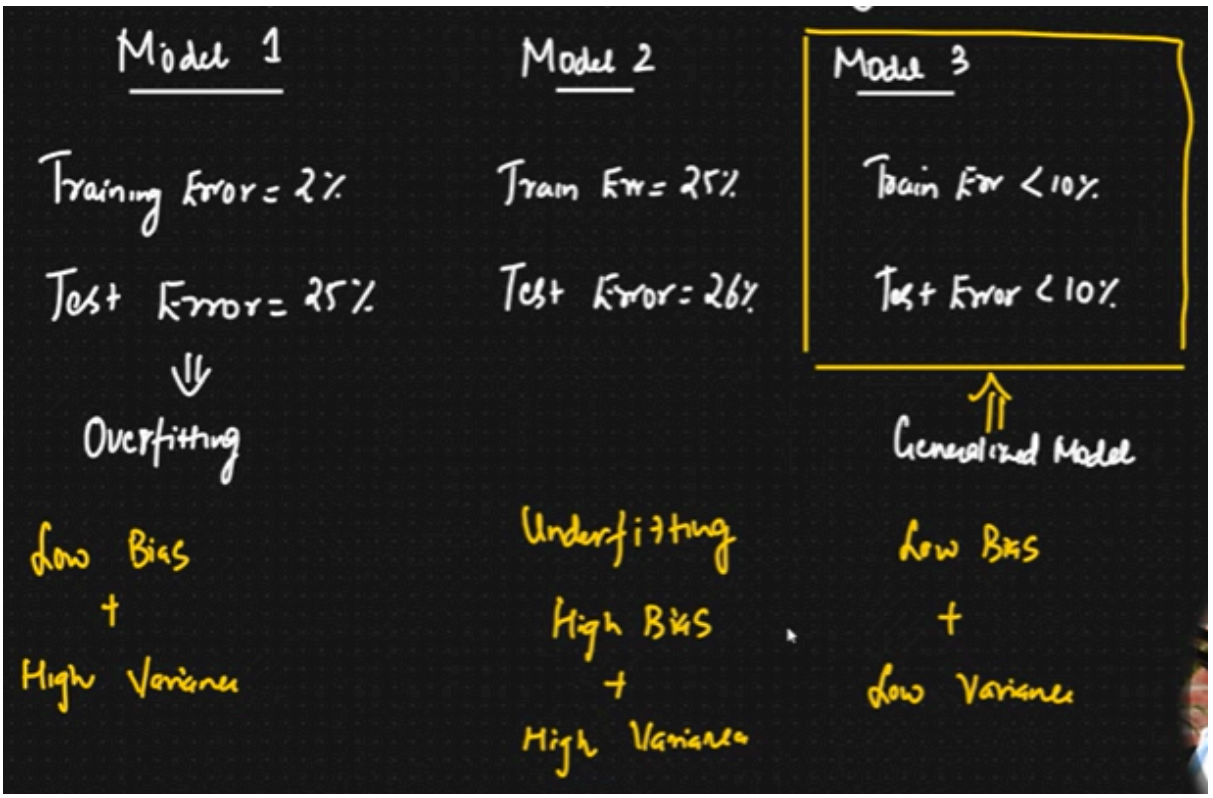
Main Challenges of Machine Learning:

Data Quality and Quantity: Having enough quality data is a major challenge. Poor quality data or insufficient data can lead to inaccurate models.
Overfitting and Underfitting: Balancing the model's complexity so it performs well on new, unseen data, without memorizing the training data or being too simplistic.
Bias and Fairness: Ensuring that the model doesn't inherit or amplify biases present in the training data.
Computational Resources: Some ML models, especially deep learning models, require significant computational power, which can be a limitation.
Interpretability and Explainability: Creating models that can be easily understood and interpreted by humans.
Generalization: Ensuring that a model trained on one set of data can perform well on data it hasn't seen before.

Difference between Overfitting and Underfitting of Data:

Overfitting:
Overfitting occurs when a model learns the training data too well, including the noise and outliers. It performs excellently on the training data but poorly on new, unseen data because it has essentially memorized the training data rather than learning to generalize. It's like a student who memorizes facts for a test but fails to understand the concepts for practical use.
Underfitting:
Underfitting happens when a model is too simple to capture the underlying patterns in the data. It doesn't perform well even on the training data. This might be due to an overly simplistic model, not enough training, or poor-quality data. It's like a student who hasn't studied enough for the test and therefore doesn't grasp the subject well.



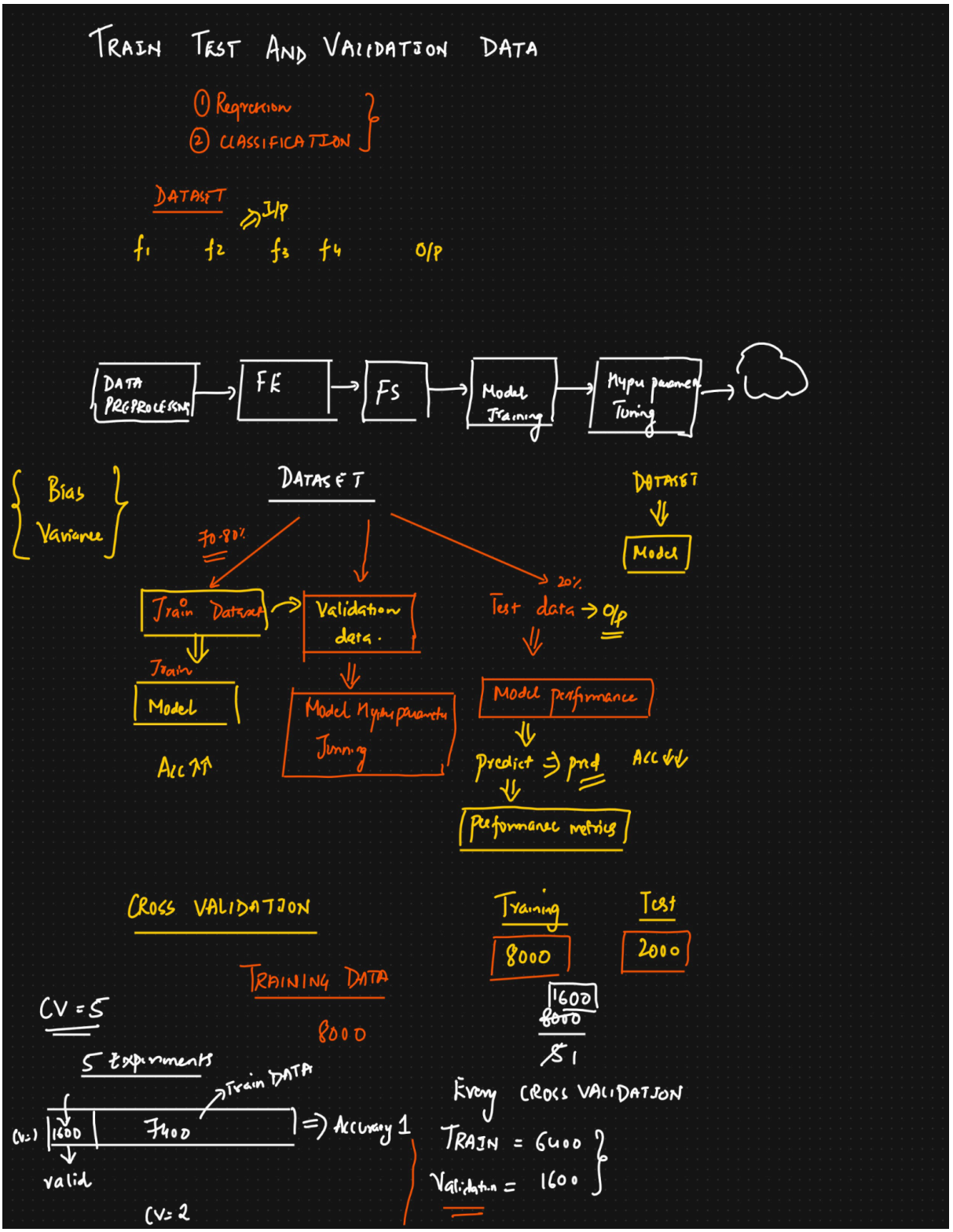


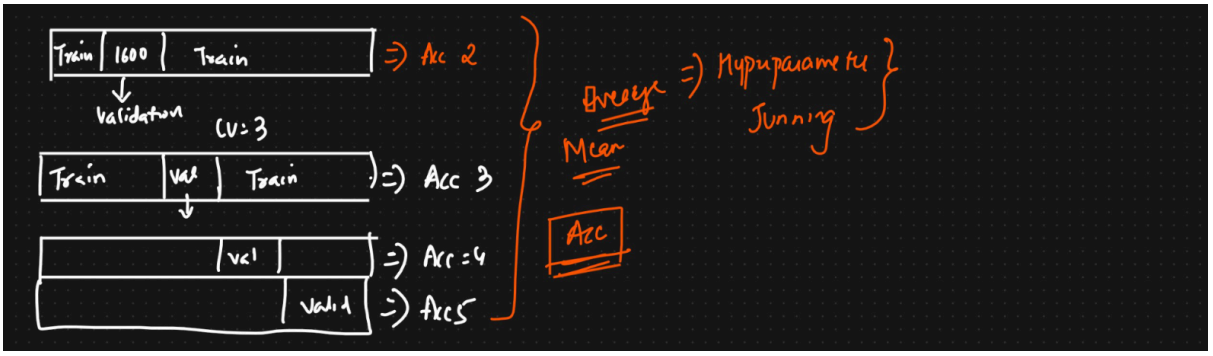
Difference between Testing and Validation:

Validation:
The validation set is a portion of your data set aside to assess the performance of your model during the training phase. It helps in tuning the parameters of the model, selecting the best model architecture, and avoiding overfitting. Essentially, it's used to validate the performance of your model on unseen data during the training process.

Testing:
The test set is another separate portion of your data used to evaluate the performance of your model after the training and validation phases are complete. It's only used once the model is completely trained and provides an unbiased

evaluation of the final model fit on the training dataset. The key here is that the test data has never been used in the training or validation process and serves as a proxy for new, unseen data.





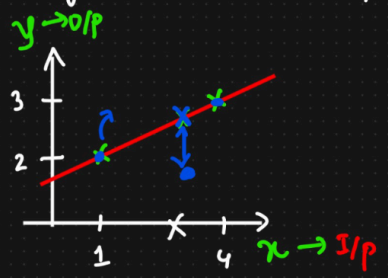
Regularization:

Regularization is a technique used to prevent overfitting in machine learning models by adding a penalty to the loss function. The idea is to simplify the model to make it more generalizable to new, unseen data. Common regularization techniques include L1 (lasso) and L2 (ridge) regularization.

Choosing Regularization Value:

The value for regularization (often represented as a lambda or alpha parameter) is typically chosen through trial and error, using techniques like cross-validation. The goal is to find a balance where the model is sufficiently complex to learn the underlying patterns in the data but not so complex that it overfits. A higher value for regularization increases the penalty, leading to simpler models, while a lower value allows the model to be more complex.

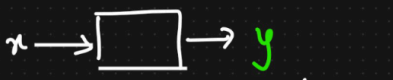
Ridge and Lasso Regression Math Intuition



Training Set

x	y
1	2
4	3

⇒ Linear Regression



→ Generalized Model

Model

Overfitting

Train Accuracy = 90%

Test Accuracy = 70%

Low Bias

High Variance

Underfitting

Train Accuracy = 60%

Test Data = 62%

High Bias

High Variance

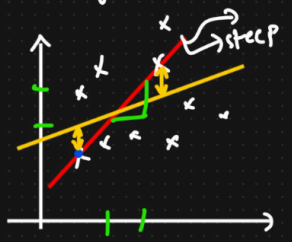
Train Acc = 90%

Test Acc = 89%

Low Bias &

Low Variance

Ridge Regression (l_2 Regularization)



$\lambda = 1$

Cost function → Residual Error

$$= \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)}\right)^2$$

$$\Rightarrow \left(h_\theta(x)^{(i)} - y^{(i)}\right)^2 + \lambda (\text{slope})^2$$

penalizing

$$= 0 + 1(2)^2$$

$$= 4$$

① Overfitting is removed

$y = mx + c$

$\Rightarrow$ {small value} $+ 1(1.3)^2$ $\lambda(m^2)$

$\Rightarrow \simeq 2.05$

$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$

small

Lasso Regression (l_1 Regularization)

$$\left(h_\theta(x)^{(i)} - y^{(i)}\right)^2 + \lambda |\text{slope}| \qquad \lambda |m|$$

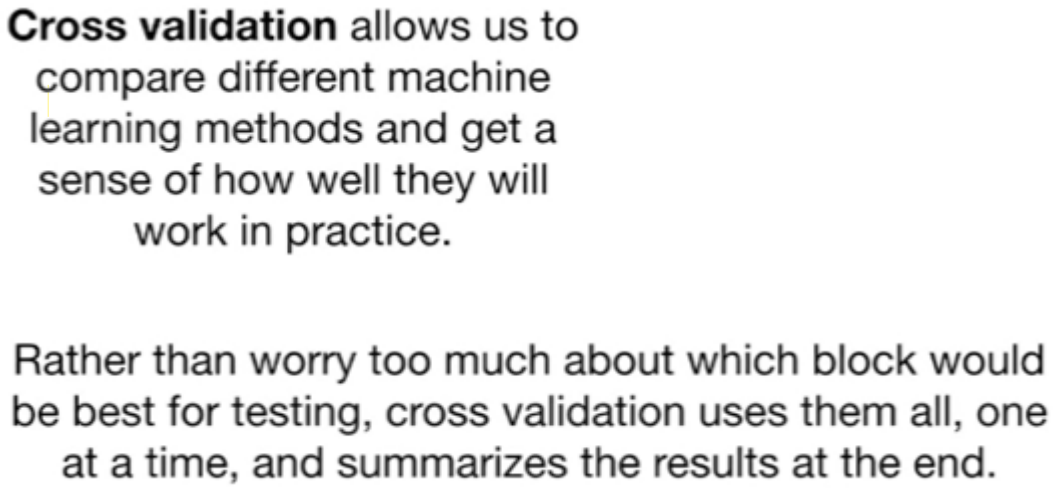
$\hookrightarrow \lambda |m_1 + m_2 + m_3 + \cdots m_n|$

① Overfitting prevent ✓

② Feature selection ✓

Cross-Validation:

Cross-validation is a technique used to assess the generalizability of a statistical model. It involves partitioning the data into subsets, training the model on some subsets (training set) and evaluating it on the remaining subsets (validation set). The most common method is k-fold cross-validation, where the data is split into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the validation data, and the other k-1 subsets form the training data. This process results in k different performance scores which can be averaged to produce a single estimation.



Types of CROSS Validation

1. LOOCV (Leave One Out Cross Validation)
2. Hold out Cross Validation
3. K-Fold Cross Validation
4. Stratified K-Fold CRoss-Validation
5. Time Series Cross-validation

⇒

1. LOOCV

Leave One Out

1300 records

DATASET

1000 records → TRAIN

TEST 300 records

{DATA Leakage}

TRAIN → Train, Validation

Disadvantage

1. Time Consuming Task
2. Overfitting ←

CV=1 Validation | TRAIN

CV=2 Validation | TRAIN

Imbalanced DATASET

2. Hold out CROSS VALIDATION

random_state

1000 records

TRAIN → TRAINING, Validation

70% 30%

⇒ 87%

⇒ 84%

⇒ 89%.

3. K Fold Cross Validation

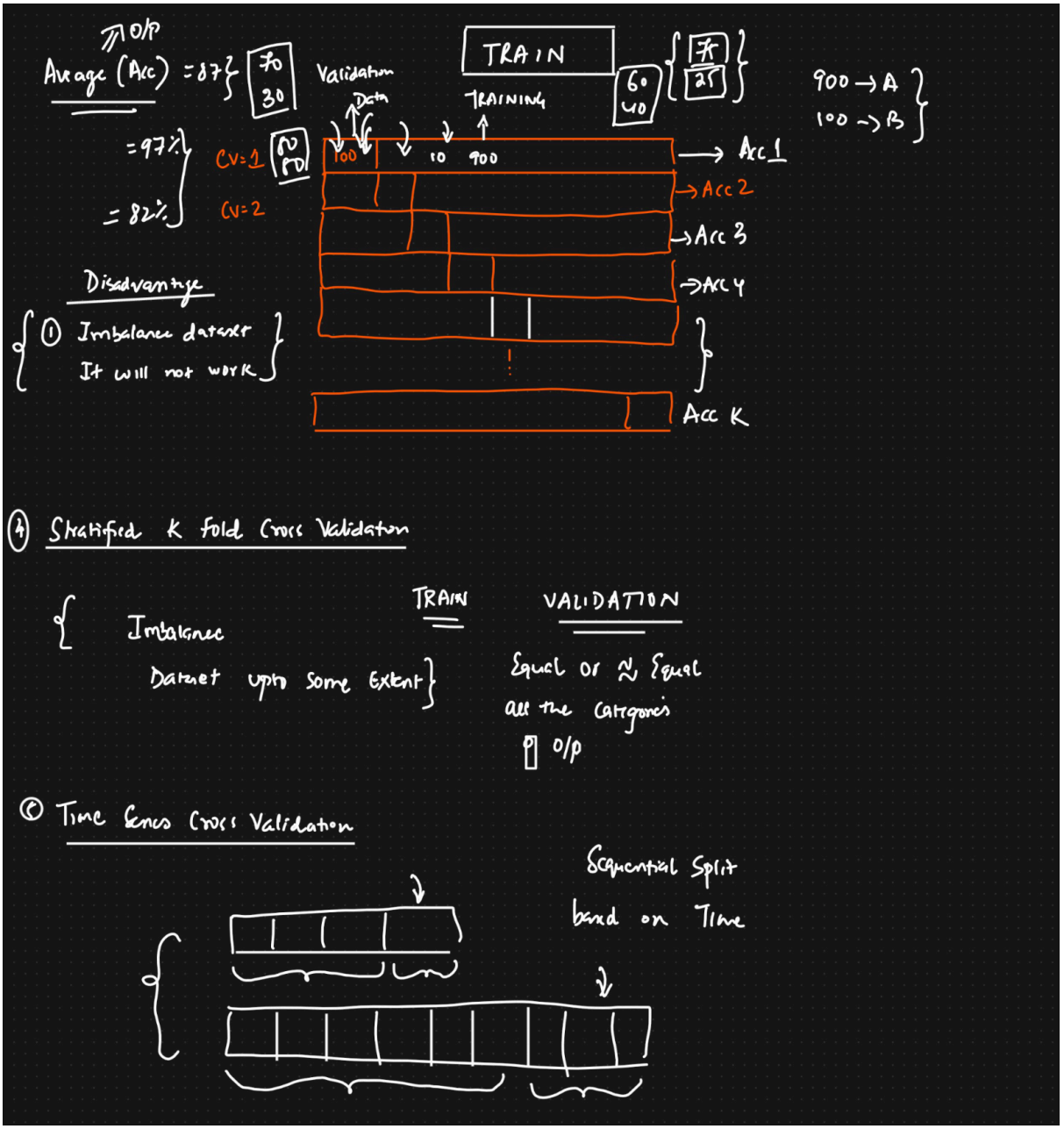
K=10 → 10 Experiments

1000/10 = 100

1000 records

900 → A

100 → B



No Free Lunch Theorem:

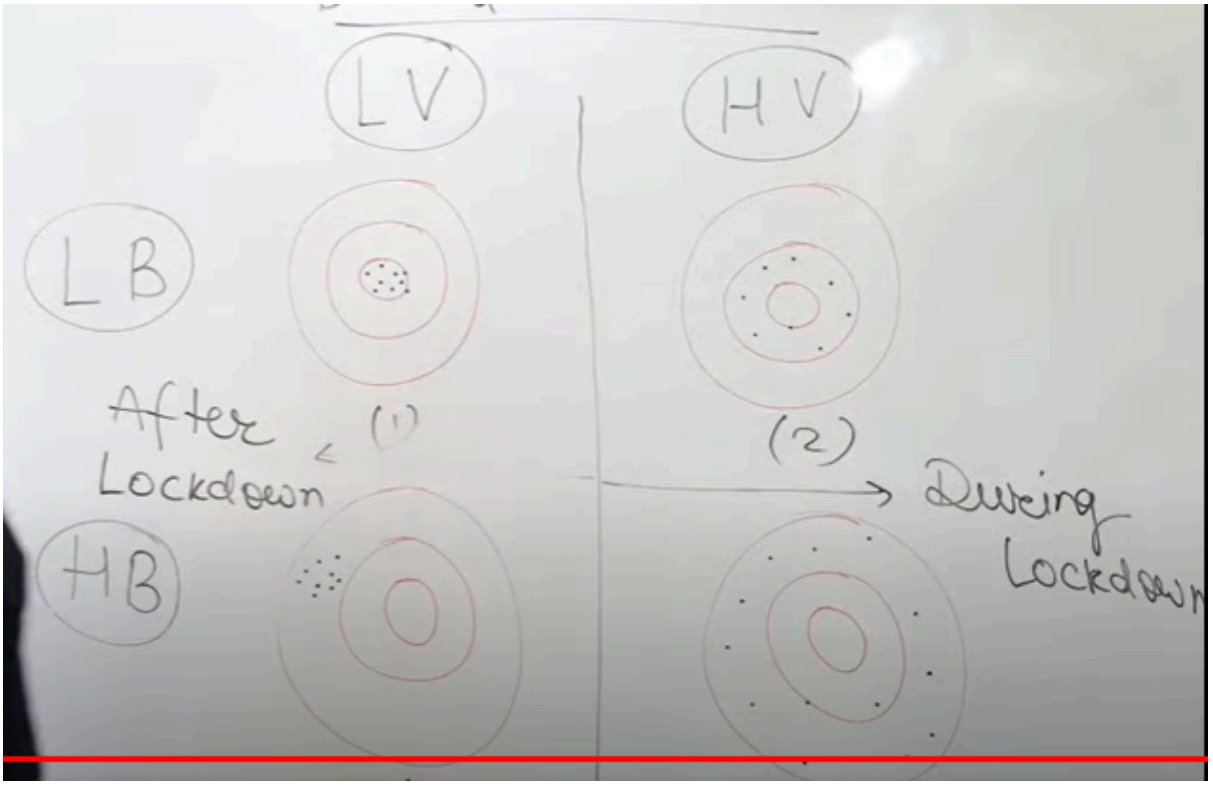
The No Free Lunch (NFL) theorem for optimization and machine learning states that no algorithm is universally better than any other algorithm when averaged across all possible problems. In simpler terms, there's no one-size-fits-all solution in machine learning. An algorithm that works well on one type of problem might perform poorly on another. This theorem highlights the importance of understanding the specificities of your data and problem context when choosing or designing a machine learning algorithm.

Bias and Variance

In machine learning, bias and variance are two critical concepts that describe errors from different sources in a model.

1. **Bias:**
   - Bias refers to the error due to overly simplistic assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting). In other words, a high-bias model is likely to be too simple, focusing on the main trends but missing the finer details of the data.

2. **Variance:**
   - Variance refers to the error due to too much complexity in the learning algorithm. High variance can cause an algorithm to model the random noise in the training data (overfitting). Essentially, a high-variance model captures noise as if it were a legitimate part of the data's pattern, leading to less generalization on new data.



Three commonly used methods for finding the sweet spot between simple and complicated models are: **regularization**, **boosting** and **bagging**.

Bias-Variance Tradeoff

The bias-variance tradeoff is a fundamental problem in supervised learning. Ideally, one wants to choose a model that simultaneously minimizes bias and variance, leading to the most accurate predictions on unseen data. Unfortunately, this is often a tradeoff:

- A model with high bias and low variance will be consistent but potentially inaccurate on average.
- A model with high variance and low bias can be very accurate on average but inconsistent across different sets of training data.

In practice, this tradeoff means that one has to balance the complexity of the model (variance) with its ability to perform well on unseen data (bias). This balance is crucial for creating models that perform well not only on the training data but also on new, unseen data.

Matrix Representation

When it comes to matrix representation, bias and variance don't directly translate into a standard matrix form. However, they can be understood in the context of the expected prediction error:

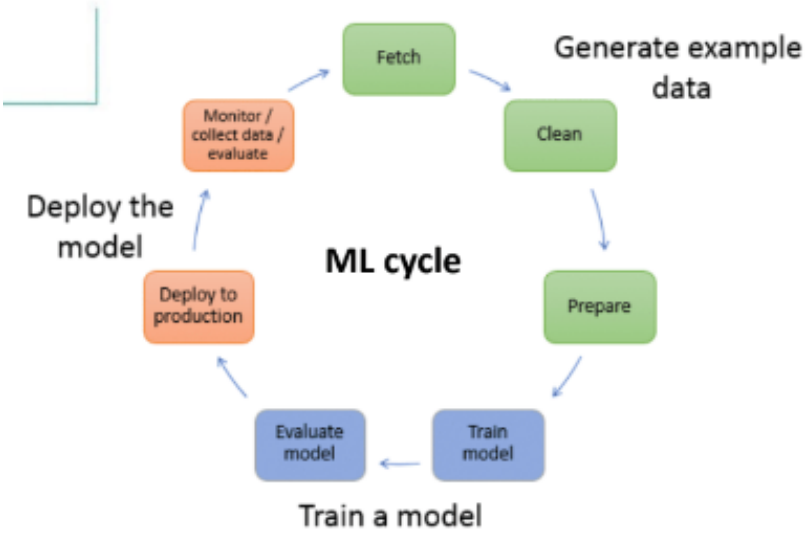
- **Expected Prediction Error:** $E[(Y - \hat{f}(X))^2]$
  - Here, Y is the true output, and $\hat{f}(X)$ is the model's prediction.
  - This error can be decomposed into bias, variance, and irreducible error components.

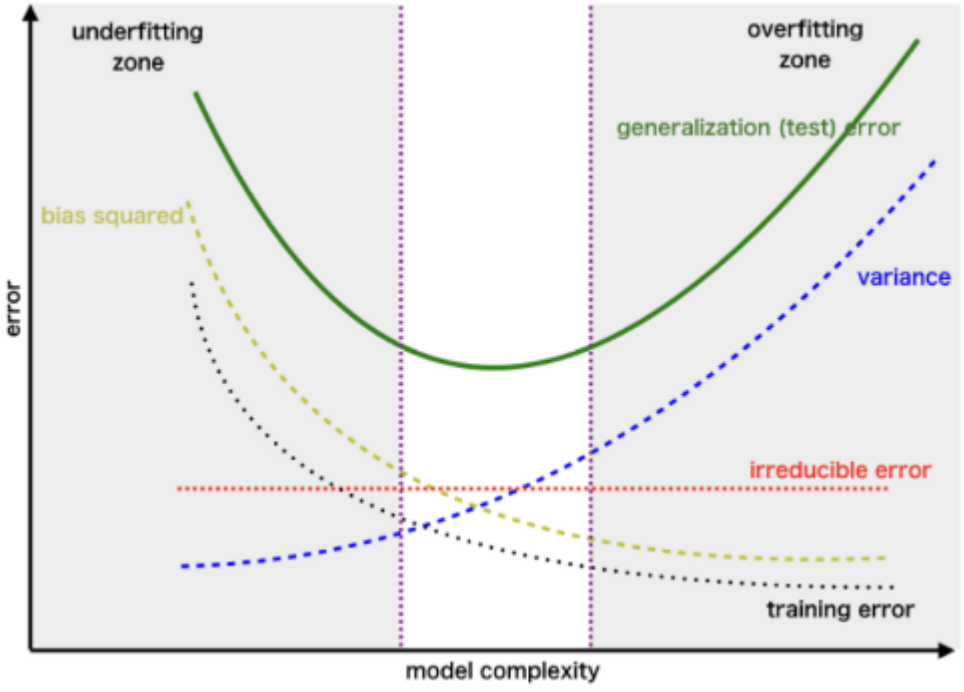
Mathematically, the expected prediction error for a model can be decomposed as:

$$\text{Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

- **Bias Term:** $(\text{Bias}[\hat{f}(X)])^2$
  - This is the squared difference between the expected prediction and the true output.
- **Variance Term:** $\text{Variance}[\hat{f}(X)]$
  - This represents the variability of model prediction.
- **Irreducible Error:**
  - This part of the error is due to the noise in the data itself and cannot be reduced by any model.

Understanding this tradeoff is crucial for building effective models that are neither too simple (high bias) nor too complex (high variance).


Fetch
Generate example data
Clean
Prepare
Train model
Evaluate model
Deploy to production
Monitor / collect data / evaluate
Deploy the model
ML cycle
Train a model


underfitting zone
overfitting zone
generalization (test) error
bias squared
variance
irreducible error
training error
error
model complexity

- When the model is in the initial phase of training, training, and test error are very high.
- When the model is enough trained, the training error is very low and the test error is high.
- The phase where training and test error is high is Underfitting.
- The phase where training error is low and the test error is high is Overfitting.
- The phase where there is a balance between training and testing error is Best fitting.
- An Underfit model has low variance and high bias.
- An Overfit model has high variance and low bias.

Confusion Matrix

A confusion matrix is a table used to evaluate the performance of a classification model. It visualizes the actual vs. predicted values in a matrix format, helping to understand how well the classifier is performing.

Here's a basic layout of a confusion matrix:

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

True Positives (TP): These are cases in which we predicted positive, and the actual value is also positive.
True Negatives (TN): These are cases in which we predicted negative, and the actual value is also negative.
False Positives (FP): These are cases in which we predicted positive, but the actual value is negative.
False Negatives (FN): These are cases in which we predicted negative, but the actual value is positive.

Evaluation Metrics Related to Confusion Matrix

1. **Accuracy:**
   - Measures the overall correctness of the model.
   - Formula: $\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$
2. **Precision:**
   - Measures the correctness achieved in positive prediction.
   - Formula: $\text{Precision} = \frac{TP}{TP+FP}$
3. **Recall (Sensitivity):**
   - Measures how well the model is detecting positive instances.
   - Formula: $\text{Recall} = \frac{TP}{TP+FN}$
4. **F1 Score:**
   - The F1 Score is the harmonic mean of precision and recall.
   - Formula: $F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}}$
   - Example: If precision is 0.8 and recall is 0.6, then $F1 = 2 \times \frac{0.8 \times 0.6}{0.8+0.6} = 0.69$
5. **Specificity:**
   - Measures the proportion of actual negatives that are correctly identified.
   - Formula: $\text{Specificity} = \frac{TN}{TN+FP}$

Precision and Recall Tradeoff

The precision and recall tradeoff is a common issue in classification models, especially when dealing with imbalanced datasets. Improving precision typically reduces recall and vice versa.

High precision means that the model is conservative in predicting positives; it's more confident about its positive predictions, but it may miss out on some actual positives (lower recall).

High recall means the model predicts most of the positive values correctly, but it can also include more negative values as positives, reducing precision.

The choice of prioritizing precision or recall depends on the specific requirements of the task. For example, in medical diagnostics, high recall (sensitivity) is often more crucial than precision.

This tradeoff highlights the importance of understanding the business or real-world context in which the model is applied, as it directly impacts the decision on which metrics to optimize.

Types of Supervised Learning Problems:

Supervised learning problems can be broadly categorized into two types:

Classification: The output variable is a category, such as "spam" or "not spam", "dog" or "cat". The goal is to predict which category the new data will fall into.

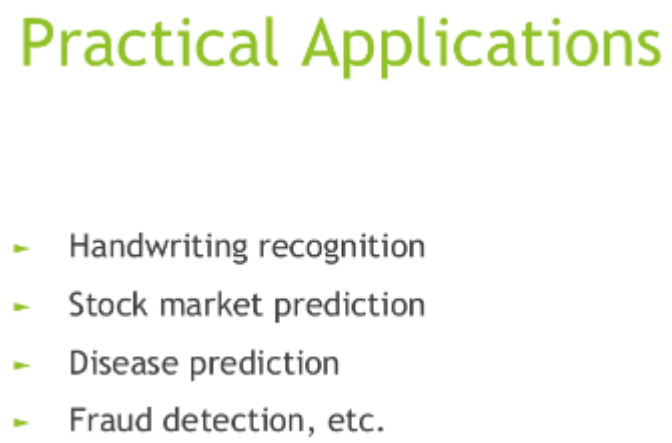
Regression: The output variable is a real value, such as "dollars" or "weight". Regression problems aim to predict a continuous output.

Hypothesis Space:

The hypothesis space in machine learning refers to the complete set of possible models that can be formulated by a learning algorithm given its input data. Essentially, it represents all the possible solutions (hypotheses) that a machine learning algorithm can choose from when trying to solve a problem.

Inductive Bias:

Inductive bias refers to the set of assumptions a machine learning algorithm makes to generalize beyond the training data. Since it's impossible for the training data to cover all possible scenarios, these assumptions (or biases) guide the algorithm in predicting outcomes for unseen data. The inductive bias is crucial for the effectiveness of a learning algorithm, as it influences how well the model can generalize from the training data to new, unseen data.



Part 1: Introduction to Supervised Learning

Layman: Imagine teaching a computer to recognize whether a picture contains a cat or a dog. You show it lots of pictures, and for each one, you tell it whether it's a cat or a dog. Over time, the computer starts to notice patterns and can guess on new pictures it's never seen before. That's what supervised learning is about: teaching computers to make predictions based on past examples.

Formal: Supervised learning is a machine learning paradigm where a model is trained on a labeled dataset, which means that each training example is paired with the correct output. The model learns from this data to make predictions or decisions, rather than discovering patterns on its own.

Part 2: Types of Supervised Learning Problems

Layman: There are two main types: guessing a category (like cat or dog, called classification) and guessing a number (like the price of a house, called regression).

Formal: Supervised learning problems are categorized into classification tasks, where the output is categorical, and regression tasks, where the output is a continuous value.

Part 3: Inductive Learning

Layman: It's like making an educated guess. If you see that every time it rains, your neighbor wears a hat, you might guess that they'll wear a hat next time it rains. You're using specific examples to make a general rule.

Formal: Inductive learning refers to the process of learning a general rule from specific examples, aiming to generalize well beyond the training data.

Part 4: Features and Feature Space

Layman: Think of features as characteristics or details about something. For instance, if you're trying to identify fruits, features might include color, size, and taste. These features help in making decisions.

Formal: Features are individual measurable properties or characteristics of a phenomenon being observed. The feature space is the n-dimensional space defined by the features, where each dimension corresponds to a feature.

Part 5: Hypothesis Space

Layman: Imagine all the possible guesses you could make about something, like all the reasons you think your friend is late. The space of all these guesses is your hypothesis space.

Formal: The hypothesis space encompasses all the possible models that could be learned by a machine learning algorithm. It is defined by the set of functions that can be chosen during the learning process.

Part 6: Function Representation and Inductive Bias

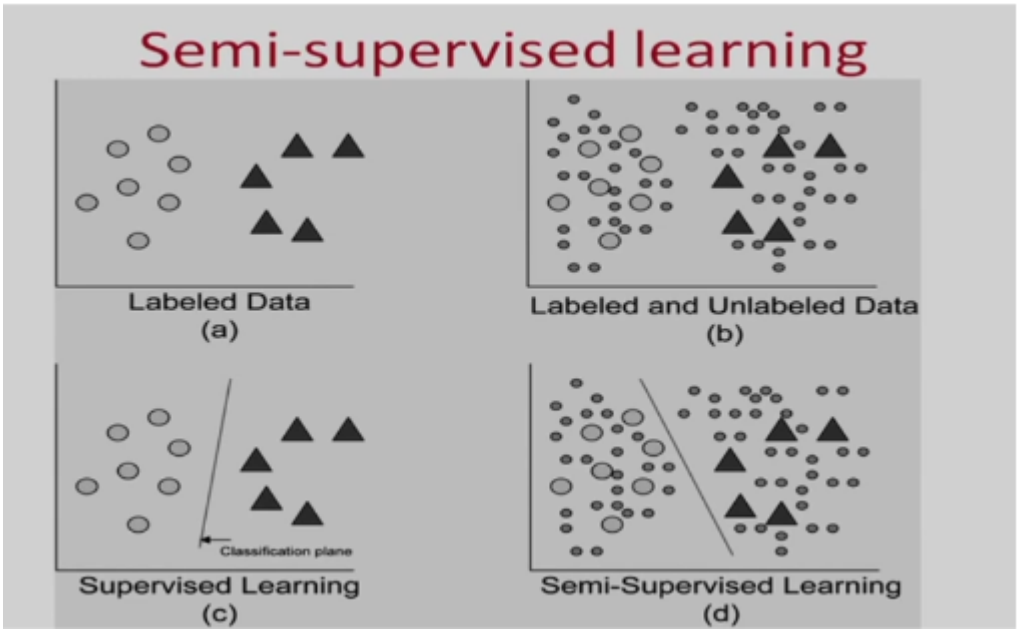
Layman: Deciding on the kind of guesswork you use to make predictions, like deciding whether to consider all possible reasons or just the simplest ones for your friend's lateness, is about choosing your approach or bias.

Formal: Function representation involves selecting the form and complexity of the model, influenced by inductive bias, which are assumptions made to generalize beyond the training data effectively.

Part 7: Important Issues in Machine Learning

Layman: When teaching a computer to make predictions, we need to think about how to pick the best way to guess, how to make sure our guesses stay accurate over time, and how confident we are in our predictions.

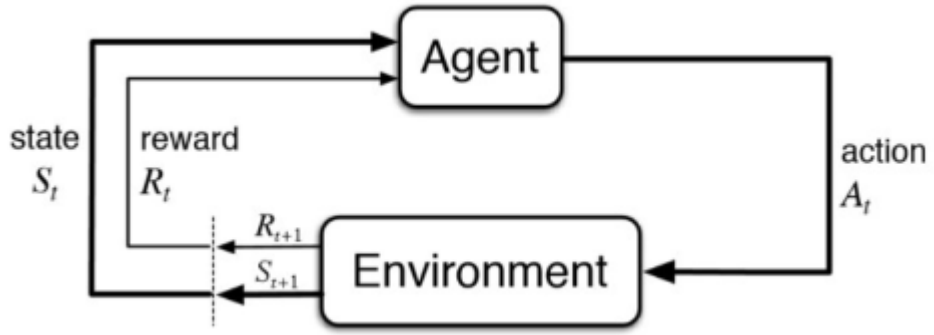
Formal: Key considerations in machine learning include defining appropriate hypothesis spaces, selecting algorithms that efficiently search through these spaces, addressing overfitting to enhance future prediction accuracy, and ensuring sufficient training data for model reliability.



Semi-supervised learning is a type of machine learning that falls between supervised and unsupervised learning. It involves training a model on a dataset that includes both labeled and unlabeled data. Typically, a small amount of labeled data is combined with a large amount of unlabeled data. The idea is that

even though the unlabeled data does not have explicit outputs, it can still provide valuable structure and distribution information that can be used to better understand and model the underlying patterns in the data. This approach is particularly useful when acquiring labeled data is expensive or time-consuming, but there's an abundance of unlabeled data available. Semi-supervised learning is often applied in scenarios like image and speech recognition, where unlabeled data is plentiful but labeling is costly.

Reinforcement learning is a type of machine learning where an agent learns to make decisions by performing actions in an environment to achieve a goal. The agent receives rewards for certain actions or sequences of actions and uses this feedback to learn over time which actions yield the most reward.



1. The agent is in a certain state, S_t, within the environment.
2. Based on this state, the agent takes an action, A_t, to affect the environment.
3. The environment responds to this action by transitioning the agent to a new state, S_{t+1}, and giving a reward, R_{t+1}, that indicates the success of the action.
4. The agent then uses this reward to update its policy, which is a strategy for choosing future actions based on the state.

The goal of the agent is to maximize the total reward it receives over time. This is often a balance between exploring new actions to find more rewarding outcomes and exploiting known actions that already give good rewards.

Types of Reinforcement Learning Problems

- can be used for a variety of **planning problems** including
  - travel plans,
  - budget planning and
  - business strategy.

Most Important Reinforcement Learning Algorithms

- Q-learning
- SARSA (State–action–reward–state–action) is an algorithm for learning a Markov decision process policy

Practical Applications

- Self-driving cars
- Intelligent robots
- AlphaGo Zero (the latest version of DeepMind's AI system playing Go)

Unsupervised learning is a type of machine learning that deals with input data without labeled responses. The goal is to model the underlying structure or distribution in the data in order to learn more about the data.

Most Important unsupervised Learning Algorithms

- Clustering
  - K-means
  - Hierarchical Cluster Analysis (HCA)
  - Expectation maximization
- Visualization and dimensionality reduction
  - Principal Component Analysis (PCA)
  - Kernel PCA
  - Locally-Linear Embedding (LLE)
  - T-distributed Stochastic Neighbor Embedding (t-SNE)
- Association rule learning
  - Apriori
  - Eclat

Types and examples of unsupervised learning include:

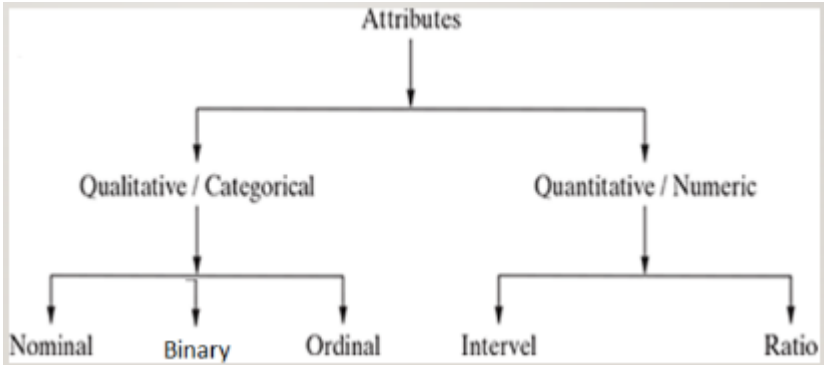
Clustering: Grouping data points into clusters of similar characteristics. For example, market segmentation in business applications.
Association: Discovering rules that describe large portions of your data, such as people that buy X also tend to buy Y. An example is market basket analysis in retail.
Dimensionality Reduction: Reducing the number of random variables to consider. An example is feature extraction to reduce the input variable space for visualization or further analysis.

Unsupervised learning is useful for exploratory analysis, finding hidden patterns, or compressing the data for further processing.

Practical Applications

- Market basket analysis
- Recommender system
- Customer segmentation, etc.

Batch learning is a type of machine learning approach where the system is incapable of incremental learning and must be trained using all available data at once. In this approach, the model is trained on the complete dataset and finalized. After training, it doesn't learn anymore; to incorporate new data, the model must be retrained from scratch with the old and the new data combined. This method can be resource-intensive and less adaptive, but it's straightforward and can be more stable since the model doesn't change over time unless retrained.



Qualitative / Categorical:

Nominal: Categories without any intrinsic ordering. Examples: Countries, colors, gender.

Binary: A special case of nominal with only two categories. Examples: Yes/No, True/False.
Ordinal: Categories with a clear ordering or ranking. Examples: Rating scales (poor, fair, good), education level (high school, bachelor's, master's).

Quantitative / Numeric:

Interval: Numeric scales with meaningful intervals but no true zero point. Examples: Temperature in Celsius, calendar years.
Ratio: Numeric scales with a meaningful zero point, allowing for the comparison of absolute magnitudes. Examples: Height, weight, duration of time.
This classification is crucial for determining the appropriate statistical analysis or machine learning techniques to apply to the data.

Data preprocessing is a crucial step in the data analysis process. It involves preparing and cleaning the data to make it suitable for a machine learning model. This usually means converting raw data into a clean data set.

Features in data preprocessing refer to the input variables, or predictors, used by a machine learning model. They are the attributes or properties that can be measured or identified in the data set.

STEPS FOR PREPROCESSING
Data Quality Assessment: Evaluating the quality of data to identify any issues with accuracy, completeness, or relevance. For example, checking if there are missing values or outliers in a dataset.

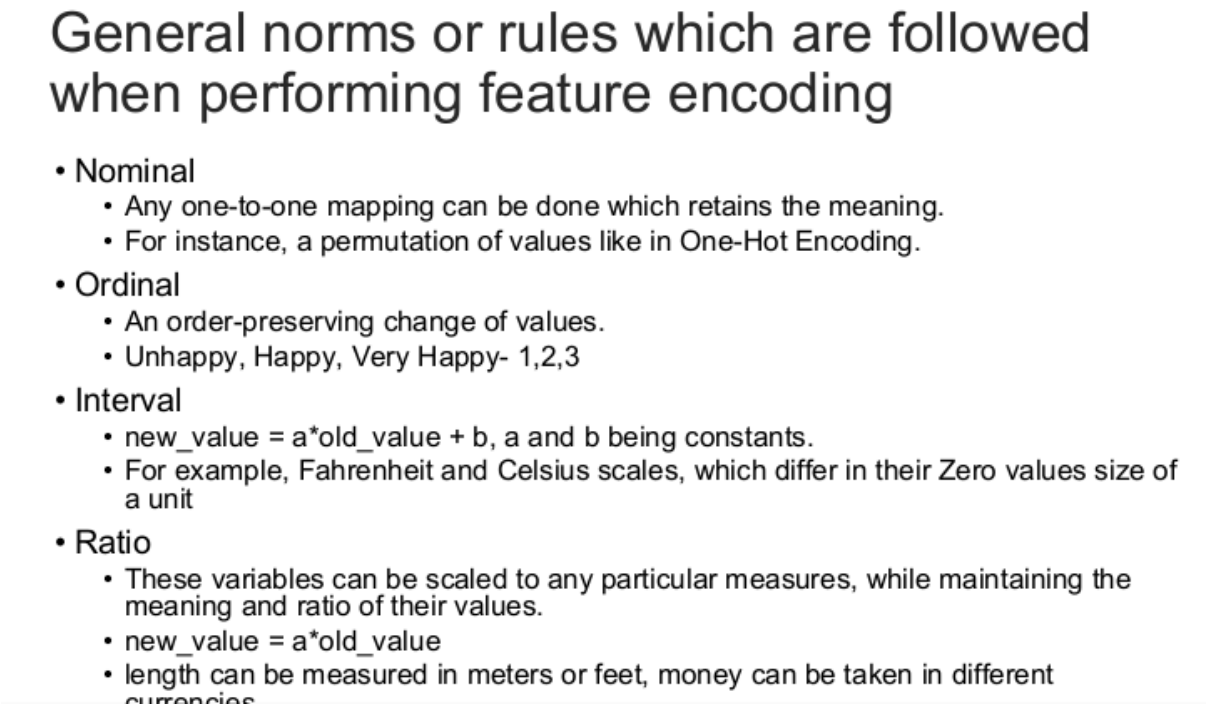
Feature Aggregation: Combining two or more attributes (features) to form a new feature, which can have a more significant impact on the predictive models. For instance, aggregating 'day' and 'month' into a 'date' feature.

Feature Discretization: Transforming continuous features into discrete ones. This can be done by binning, which involves dividing a continuous feature into intervals. For example, age can be discretized into '0-20', '21-40', '41-60', etc.

Feature Sampling: Selecting a subset of data to reduce the size of the dataset for analysis, which can be helpful in dealing with large datasets. For example, using a random sample of a large dataset for preliminary model training to save computational resources.

Dimensionality Reduction: Reducing the number of features in the dataset to prevent overfitting and to reduce computational costs, while retaining the essential information. Principal Component Analysis (PCA) is a common technique used for this purpose.

Feature Encoding: Converting categorical data into a numerical format so that it can be provided to ML algorithms. One example is using one-hot encoding to transform the 'gender' feature with values 'male' and 'female' into a binary format.



Feature Scaling: Adjusting the scale of features so that they have a common scale. This is important for algorithms that depend on the distance between features, like SVM or KNN. Standardization (z-score scaling) and normalization (min-max scaling) are common methods used.

Comes Under Feature Encoding

One Hot Encoding is a process used to convert categorical variables into a form that could be provided to ML algorithms to do a better job in prediction. For example, if you have a 'color' feature with three categories (red, green, blue), one hot encoding will create three new features called 'red', 'green', and 'blue'. Each feature will have a binary response (0 or 1) indicating the absence or presence of the color.

Dummy Variable Trap is a scenario where independent variables are highly correlated. In the case of one hot encoding, it happens when the binary indicator variables (dummy variables) are used for each category level and include all those dummy variables in a regression model simultaneously. For the 'color' example, including red, green, and blue dummy variables in the model could lead to multicollinearity because knowing the value of red and green will give you the value of blue. To avoid the dummy variable trap, one category level should be

dropped and used as a baseline (e.g., omit 'blue' and only include 'red' and 'green' in the model).

Certainly! Let's use a practical example to clarify the concept of the dummy variable trap in the context of one-hot encoding.

Imagine you have a dataset of cars with a categorical feature 'Color' that can take three possible values: Red, Blue, and Green. You want to use one-hot encoding to create dummy variables for this categorical variable so that you can include it in a regression model.

Without Avoiding Dummy Variable Trap:

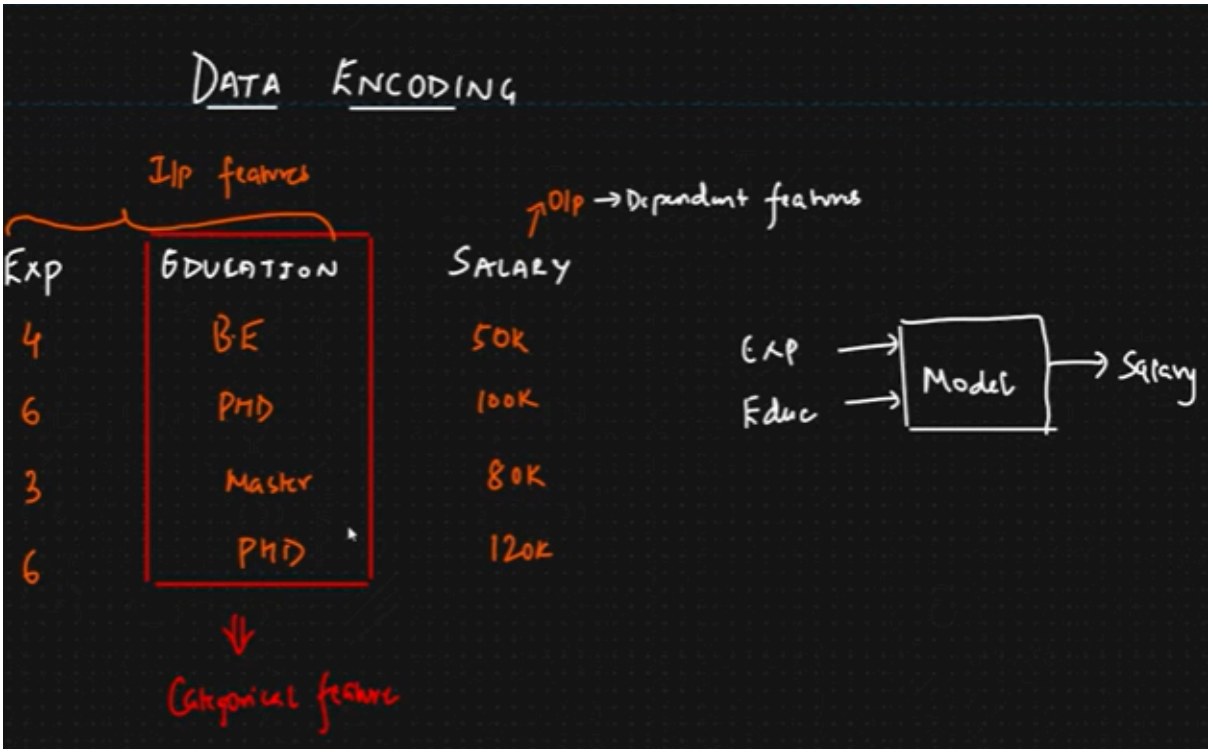
Car	Color	Dummy_Red	Dummy_Blue	Dummy_Green
A	Red	1	0	0
B	Blue	0	1	0
C	Green	0	0	1

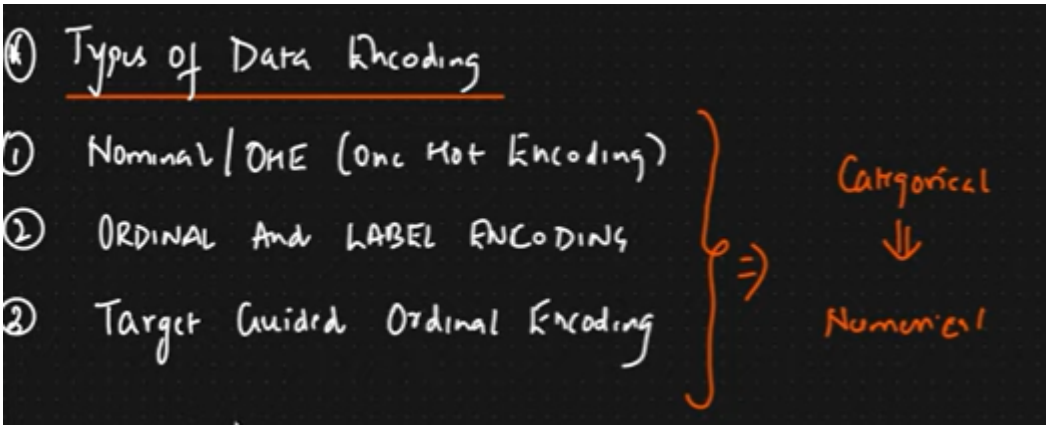
Here, `Dummy_Red`, `Dummy_Blue`, and `Dummy_Green` are the dummy variables created through one-hot encoding. However, if you include all three dummy variables in the model, you introduce multicollinearity because knowing the value of two dummy variables will automatically give you the value of the third. For instance, if `Dummy_Red` and `Dummy_Blue` are both 0, `Dummy_Green` must be 1.

Avoiding Dummy Variable Trap:

Car	Color	Dummy_Red	Dummy_Blue
A	Red	1	0
B	Blue	0	1
C	Green	0	0

To avoid the dummy variable trap, you drop one of the dummy variables (in this case, `Dummy_Green`). Now, `Dummy_Red` and `Dummy_Blue` indicate whether a car is Red or Blue, respectively, and if both are 0, it indicates that the car is Green. This way, you avoid multicollinearity and the dummy variable trap, making the model coefficients interpretable and reliable.





Nominal encoding is a technique used to transform categorical variables that have no intrinsic ordering into numerical values that can be used in machine learning models. One common method for nominal encoding is one-hot encoding, which creates a binary vector for each category in the variable.

House Price (DATASET)

No. of Rooms	House Size	Location	Price
		Bangalore	
		Noida	
		Dehi	
		Bangalore	
		Dehi	
		Noida	

Location → OHE

Bangalore	Delhi	Noida
1	0	0
0	0	1
0	1	0
1	0	0
0	1	0

Disadvantage [10 location]

① Sparse matrix [overfitting]

② 1000 features [1000 categories]

⇓

It will increase the number of the features

One popular method for converting discrete variables, or features, into numbers is to use something called **One-Hot Encoding**.

Favorite Color	Height (m)	Loves Troll 2
Blue	1.77	Yes
Red	1.32	No
Green	1.81	Yes
Blue	1.56	No
Green	1.64	Yes
Green	1.61	No
Blue	1.73	No

Blue	Red	Green	Height (m)	Loves Troll 2
1	0	0	1.77	1
0	1	0	1.32	0
0	0	1	1.81	1
1	0	0	1.56	0
0	0	1	1.64	1
0	0	1	1.61	0
1	0	0	1.73	0

But when we have a lot of options, for example, if we had a column of **Postal Codes**, and there are **41,683** postal codes in the United States…

…then we would end up replacing the one **Postal Code** column with **41,683** new columns, which might make the data difficult to work with.

Simply converting the discrete values to random numbers, like we did here, is called **Label Encoding**.

Favorite Color	Height (m)	Loves Troll 2
0	1.77	1
1	1.32	0
2	1.81	1
0	1.56	0
2	1.64	1
2	1.61	0
0	1.73	0

Because we used the **Target**, the thing we want to predict, to determine what values to replace the discrete options, this method is called **Target Encoding**.

Favorite Color	Height (m)	Loves Troll 2
0.33	1.77	1
0	1.32	0
0.67	1.81	1
0.33	1.56	0
0.67	1.64	1
0.67	1.61	0
0.33	1.73	0

Feature scaling is applied in machine learning to normalize the range of independent variables or features of data. It is generally performed during the data preprocessing phase before running a machine learning algorithm.

When to Use Feature Scaling:

Feature scaling is used when features have different scales and we want to apply algorithms that are sensitive to the scale of the data, like SVMs, k-nearest neighbors, and gradient descent-based algorithms (like linear regression or neural networks).

Why Perform Feature Scaling:

The primary reason is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges, which can negatively affect the performance of a model.
It also helps gradient descent converge more quickly.

How to Perform Feature Scaling:

Standardization: It involves rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1. It's not bounded and is suitable for features with a Gaussian distribution.

Normalization: This technique scales the features to a range of [0, 1], which is a special case of min-max scaling. This scaling is beneficial for non-Gaussian distributions and is bound within a range, making it useful for algorithms that require input data within bounds.

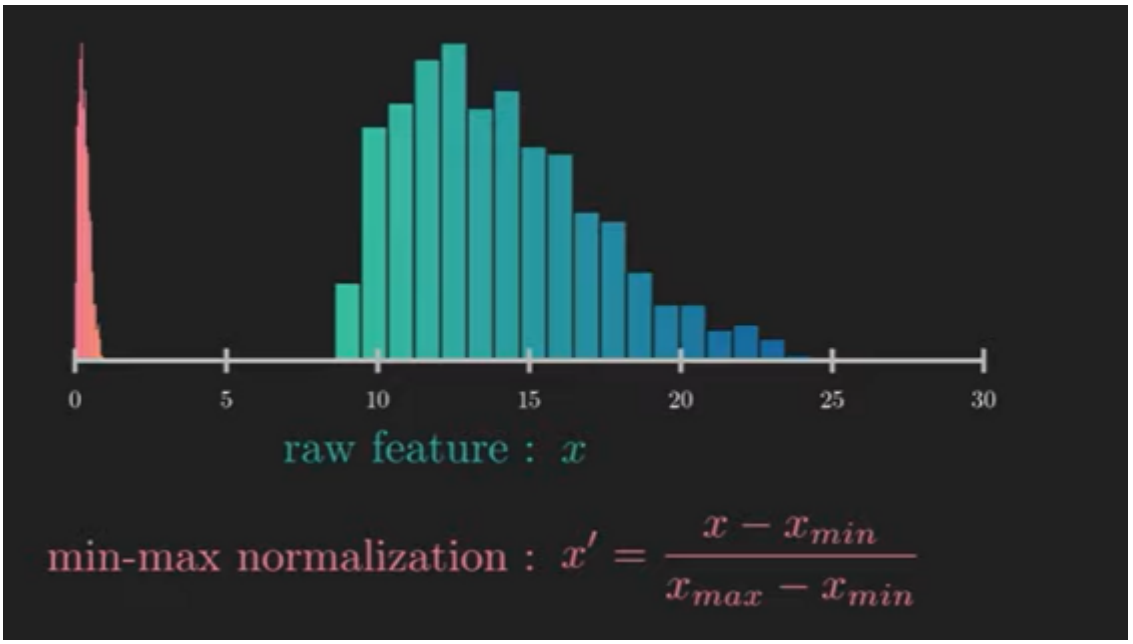
Standardization

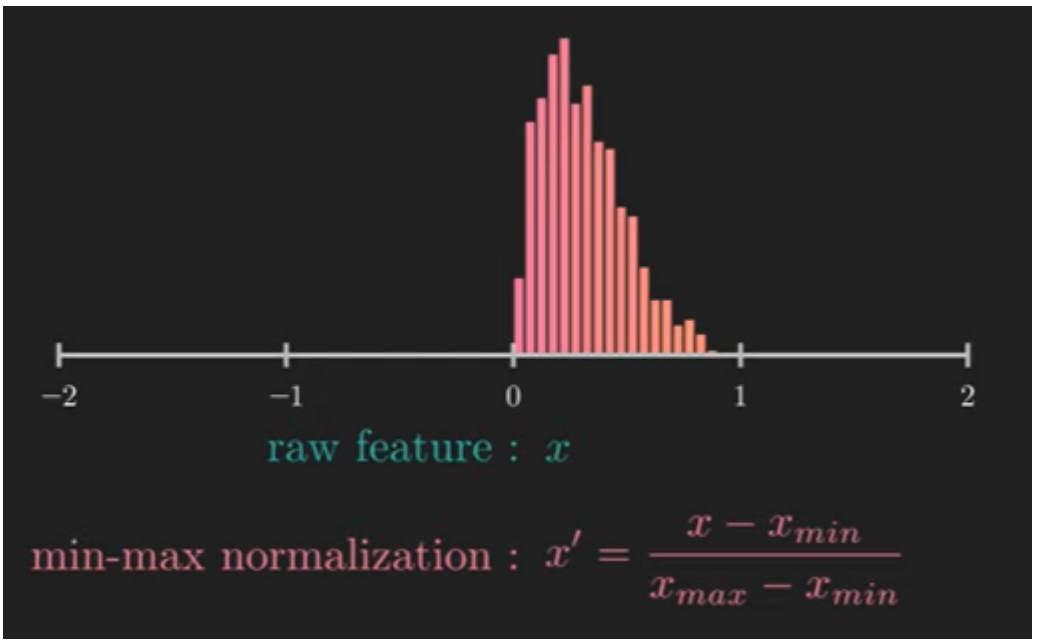
- $X' = \frac{x-\mu}{\sigma}$
- Values are shifted and rescaled so that they end up ranging between -3 and +3.
- the values are centered around the mean with a unit standard deviation.
- converts the data to mean of 0 and standard deviation of 1.
- Works all the time

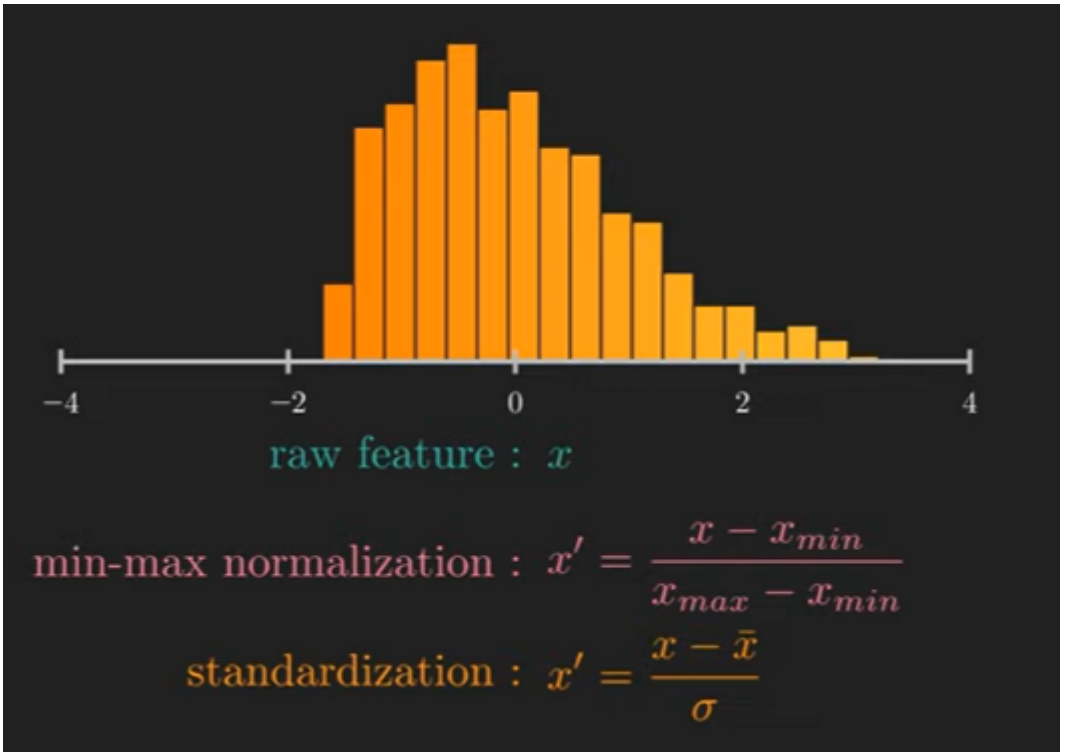
Normalization

- $X' = \frac{X-Xmin}{X_{max}-Xmin}$
- values are shifted and rescaled so that they end up ranging between 0 and 1.
- It is also known as Min-Max scaling.
- Is recommended when we have normal distribution in most of the features.

Normalize Non-Gaussian features and Standardize Gaussian-like features

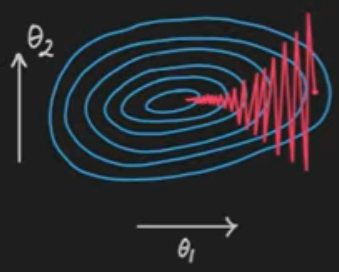



−2
−1
0
1
2
raw feature : x
min-max normalization : $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$


−4
−2
0
2
4
raw feature : x
min-max normalization : $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$
standardization : $x' = \frac{x - \bar{x}}{\sigma}$

Why Feature Scaling is Important?

- Faster Convergence



$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum (\theta_1 x_1 + \theta_2 x_2 - y)\, x_1$$

(small)

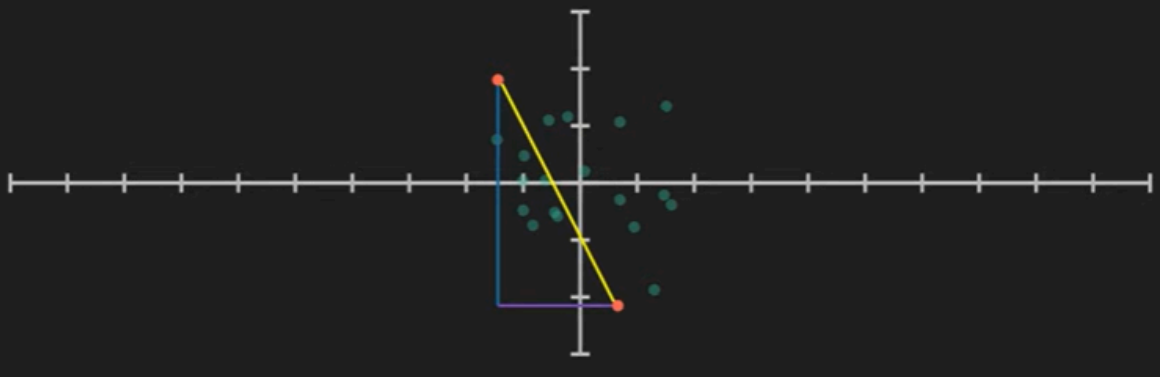
$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum (\theta_1 x_1 + \theta_2 x_2 - y)\, x_2$$

(big)

$$J(\theta_1, \theta_2) = \frac{1}{2m} \sum (\theta_1 x_1 + \theta_2 x_2 - y)^2$$

$$x_1 \in (0, 1);\ x_2 \in (100, 1000)$$

- Computing Distance Appropriately



Gaussian Distribution — BMI, BloodPressure, Glucose.
Non-Gaussian Distribution — Age, DiabetesPedigreeFunction, Insulin, Pregnancies, SkinThickness

Normalize Non-Gaussian features and Standardize Gaussian-like features

Feature Scaling Requirement

- Gradient Descent Based Algorithms
  - linear regression, logistic regression, neural network, etc. that use gradient descent as an optimization technique require data to be scaled
  - Having features on a similar scale can help the gradient descent converge more quickly towards the minima
- Distance-Based Algorithms
  - KNN, K-means and SVM
  - are most affected by the range of features.
  - they are using distances between data points to determine their similarity.
  - there is a chance that higher weightage is given to features with higher magnitude.
- Tree-Based Algorithms
  - are fairly insensitive to the scale of the features.

[The Bias-Variance Tradeoff. In this post, we will explain the... | by Giorgos Papachristoudis | Towards Data Science]

When is F1 score an important measure?/ Why we take harmonic mean in case of F1 score?

The F1 score is a measure that combines precision and recall of a test into a single metric by taking their harmonic mean. This measure is important in situations where you need to balance precision and recall, and particularly when the class distribution is imbalanced—that is, when one class is significantly less frequent than the other.

Precision is the ratio of true positive predictions to the total positive predictions (including false positives). It is a measure of the quality of the model's predictions:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall (also known as sensitivity) is the ratio of true positive predictions to the total actual positives (including false negatives). It is a measure of the model's ability to detect positive instances:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

The **F1 score** is the harmonic mean of precision and recall and is calculated as:

$$\text{F1 score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The reason the harmonic mean is used (and not the arithmetic mean) is that the harmonic mean tends to be closer to the smaller number when dealing with two numbers. This means that if either precision or recall is low, the F1 score will also be low. It gives a more realistic measure of the performance of a test compared to the arithmetic mean, which could give an unduly high average if one of the measures (precision or recall) is very high, masking the poor performance of the other.

For example, consider a medical test for a disease that is very rare (imbalanced classes). If you predict 'no disease' for all patients, you may end up with very high precision but extremely low recall, as you'll never correctly identify those few patients with the disease. The F1 score would rightly reflect the poor performance of such a model in this case, emphasizing the need for a balance between precision and recall.

Show different cases where Normalization versus Standardization should be used as feature scaling method.

Normalization is usually done when you want to scale the data between a particular range (typically 0 to 1). It is useful when you know the approximate minimum and maximum values that your input features could have. For instance, pixel intensities in an image are always in the range of 0 to 255; normalizing these values can bring them to a scale from 0 to 1. This technique is useful when you have specific knowledge about the minimum and maximum values that data can take.

Use Cases for Normalization:

- When the data does not follow a Gaussian distribution.
- When you need bounded values on the input.
- For algorithms that assume data is on the same scale, like k-nearest neighbors with a Euclidean distance measure, neural networks, or when using gradient descent-based optimization.

Standardization is a scaling technique that assumes that the distribution of the data is normal and scales the data to have a mean of zero and a standard deviation of one. It is less affected by outliers than normalization. Standardization is useful when you want to compare features that have different units or different scales.

Use Cases for Standardization:

- When the data follows a Gaussian distribution (or when you want to make this assumption).
- For algorithms that assume data is centered around zero, like support vector machines or principal component analysis.
- When you want to mitigate the influence of outliers.

What will happen in the case of m<<n, where m is the number of instances and n is the number of features?

this is a well-known scenario in machine learning known as the "curse of dimensionality" or having a dataset with more features than samples, and it can lead to several issues:

Overfitting: When you have more features than instances, the learning algorithm can start to "memorize" the training data instead of learning to generalize from it. This means that while it may perform very well on the training data, it will likely perform poorly on unseen data.

Computational Complexity: More features mean that the complexity of the model increases, which can significantly increase the time and computational resources required to train models.

Noise: With a large number of features, there's an increased chance that some of the features are irrelevant or noisy, which can mislead the learning algorithm and result in poorer performance.

Feature Selection: It becomes crucial to apply feature selection techniques to reduce the number of features to those that are most relevant to the prediction task.

Regularization: Techniques such as L1 (Lasso) and L2 (Ridge) regularization become more important as they can help to prevent overfitting by penalizing the model for having too many large coefficients.

Dimensionality Reduction: Techniques like Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) may be used to reduce the number of dimensions while retaining most of the variability in the data.

Sparse Data: In cases where the data is also sparse (most feature values are zero), specialized algorithms that can handle sparsity effectively should be considered.

In summary, when the number of features significantly exceeds the number of instances, special care must be taken to avoid overfitting, manage computational complexity, and ensure that the model can generalize well to new data. Feature selection, regularization, and dimensionality reduction are key strategies in dealing with such scenarios.

Simple Linear Regression

Simple linear regression is a statistical method used to model the relationship between a single independent variable and a dependent variable by fitting a linear equation to observed data. The steps for performing simple linear regression are:

1. **Model Specification**:
   Define the model as $Y = \beta_0 + \beta_1 X + \epsilon$, where:
   - Y is the dependent variable.
   - X is the independent variable.
   - β_0 is the intercept.
   - β_1 is the slope of the line.
   - ϵ is the error term.
2. **Estimation of Coefficients**:
   Calculate β_0 and β_1 using the least squares method, which minimizes the sum of squared residuals:
   - $\beta_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$
   - $\beta_0 = \bar{y} - \beta_1 \bar{x}$

     where $\bar{x}$ and $\bar{y}$ are the means of X and Y, respectively.

3. **Model Fitting**:
   Use the estimated coefficients to fit the model, which can then be used for making predictions.
4. **Model Assessment**:
   Evaluate the model's performance using metrics like R^2, which measures the proportion of variance in the dependent variable that is predictable from the independent variable.

Linear Regression

Linear regression is a common statistical tool for modeling the relationship between some "explanatory" variables and some real valued outcome. Cast as a learning problem, the domain set $\mathcal{X}$ is a subset of $\mathbb{R}^d$, for some d, and the label set $\mathcal{Y}$ is the set of real numbers. We would like to learn a linear function $h : \mathbb{R}^d \to \mathbb{R}$ that best approximates the relationship between our variables (say, for example, predicting the weight of a baby as a function of her age and weight at birth). Figure 9.1 shows an example of a linear regression predictor for $d = 1$.

The hypothesis class of linear regression predictors is simply the set of linear functions,

$$\mathcal{H}_{reg} = L_d = \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle + b : \mathbf{w} \in \mathbb{R}^d,\ b \in \mathbb{R}\}.$$

Next we need to define a loss function for regression. While in classification the definition of the loss is straightforward, as $\ell(h, (\mathbf{x}, y))$ simply indicates whether $h(\mathbf{x})$ correctly predicts y or not, in regression, if the baby's weight is 3 kg, both the predictions 3.00001 kg and 4 kg are "wrong," but we would clearly prefer the former over the latter. We therefore need to define how much we shall be "penalized" for the discrepancy between $h(\mathbf{x})$ and y. One common way is to use the squared-loss function, namely,

$$\ell(h, (\mathbf{x}, y)) = (h(\mathbf{x}) - y)^2.$$

For this loss function, the empirical risk function is called the Mean Squared Error, namely,

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} (h(\mathbf{x}_i) - y_i)^2.$$

Pros and Cons

- Works on any size of dataset, gives information about relevance of features
- The Linear Regression Assumptions need to be checked

Assumptions of linear regression include:

Linearity: The relationship between the independent and dependent variables should be linear.
Independence: Observations should be independent of each other.

Homoscedasticity: The residuals (differences between observed and predicted values) should have constant variance.
Normality: The residuals should be normally distributed.
No multicollinearity: Independent variables should not be too highly correlated with each other.

Ridge Regression adds a penalty equal to the square of the magnitude of coefficients to the loss function of linear regression. This method aims to prevent overfitting by shrinking large coefficients.

The formula for Ridge Regression is:

$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}))^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

where λ is the regularization parameter.

Lasso Regression is similar to Ridge Regression but adds a penalty equal to the absolute value of the magnitude of coefficients.

The formula for Lasso Regression is:

$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}))^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Both Ridge and Lasso can be used to improve the prediction accuracy and interpretability of statistical models.

A cost function, often used in machine learning, measures the performance of a model by quantifying the error between predicted values and actual values. It's a function that the model wants to minimize during training. For example, in linear regression, a common cost function is the Mean Squared Error (MSE), which calculates the average of the squares of the errors between predicted and actual values.

The difference between Ridge and Lasso regression lies in the penalty applied to the cost function:

Ridge Regression applies a penalty proportional to the square of the coefficient magnitudes (L2 penalty). This tends to shrink the coefficients for all features but does not set any to zero.
Lasso Regression applies a penalty proportional to the absolute value of the coefficient magnitudes (L1 penalty). This can shrink some coefficients to zero, performing feature selection.

Gradient Descent is an optimization algorithm used to find the minimum of a function. It iteratively moves towards the steepest descent as defined by the negative of the gradient.

Stochastic Gradient Descent (SGD) is a variation of gradient descent where the gradient is calculated and updated using just one randomly selected data point at a time. This can make SGD faster and better suited to large datasets, though it may introduce more variance in the path to convergence.

Here's an outline of the key topics presented across the slides:

Purpose of Regression Models: It discusses two main reasons for building regression models — explanatory and predictive purposes.

Simple Linear Regression: Explains the statistical method that summarizes and studies relationships between two continuous variables: an independent variable (predictor) and a dependent variable (response).

Model Equation: Introduces the linear equation $y = b0 + b1 * x$, where $b0$ is the intercept, $b1$ is the slope, and y and x are the dependent and independent variables, respectively.

Coefficient Meaning: Discusses the interpretation of the slope and constant in a real-world context, like the relationship between years of experience and salary.

Goal of Linear Regression: Aims to find the best estimates for the coefficients to minimize errors in prediction.

Assumptions about Error: Assumes errors in the data, represented as ϵ, are normally distributed with mean zero and some variance σ^2.

Least Squares Estimation: Describes the method to find the line that minimizes the sum of squared prediction errors.

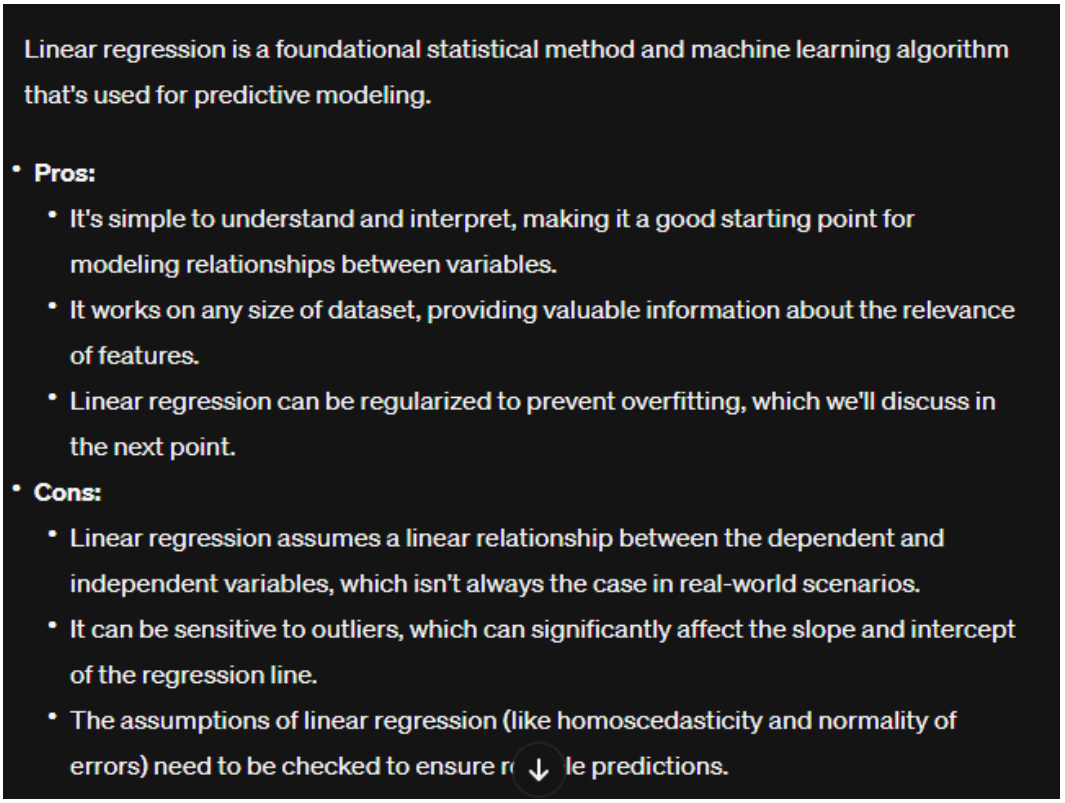
Assumptions of Linear Regression: Covers the necessary conditions for performing regression, such as linearity, homoscedasticity, multivariate normality, independence, and lack of multicollinearity.

Pros and Cons of Linear Regression: Addresses the scalability of linear regression and the need to check for assumptions.

Regularization Techniques: Introduces methods like Lasso and Ridge Regression to avoid overfitting.

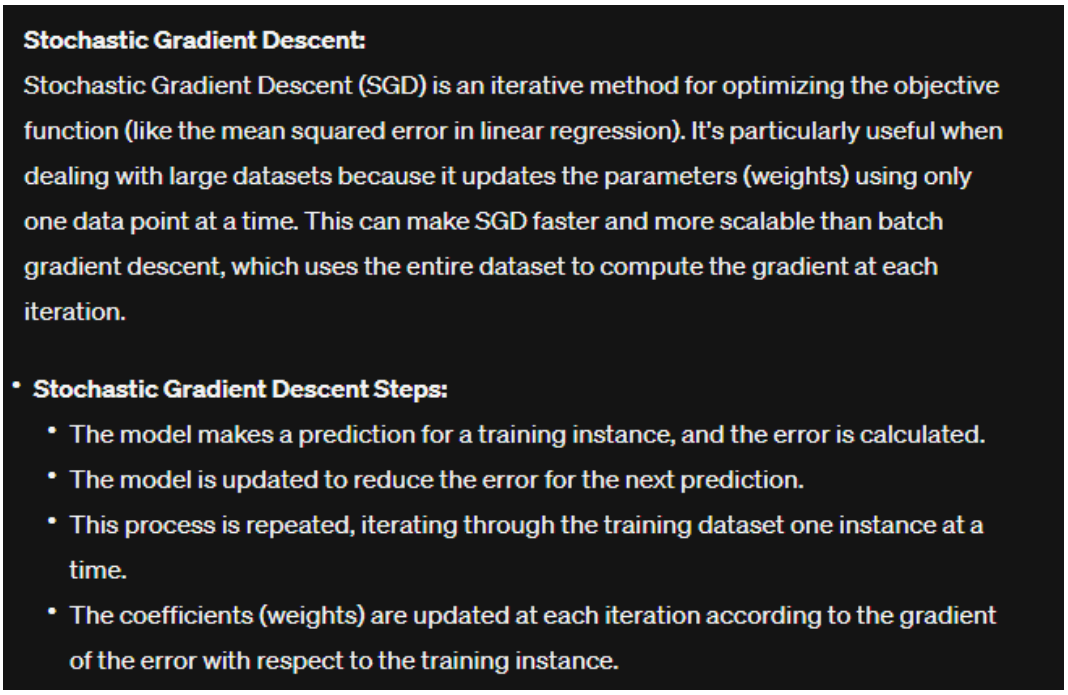
Stochastic Gradient Descent: Describes an iterative method for optimizing the parameters of the model.

Practical Application: Walks through the process of importing libraries, preparing data, splitting datasets, training the model, making predictions, and visualizing results.



Linear regression is a foundational statistical method and machine learning algorithm that's used for predictive modeling.

- **Pros:**
  - It's simple to understand and interpret, making it a good starting point for modeling relationships between variables.
  - It works on any size of dataset, providing valuable information about the relevance of features.
  - Linear regression can be regularized to prevent overfitting, which we'll discuss in the next point.
- **Cons:**
  - Linear regression assumes a linear relationship between the dependent and independent variables, which isn't always the case in real-world scenarios.
  - It can be sensitive to outliers, which can significantly affect the slope and intercept of the regression line.
  - The assumptions of linear regression (like homoscedasticity and normality of errors) need to be checked to ensure r le predictions.



Stochastic Gradient Descent:

Stochastic Gradient Descent (SGD) is an iterative method for optimizing the objective function (like the mean squared error in linear regression). It's particularly useful when dealing with large datasets because it updates the parameters (weights) using only one data point at a time. This can make SGD faster and more scalable than batch gradient descent, which uses the entire dataset to compute the gradient at each iteration.

- **Stochastic Gradient Descent Steps:**
  - The model makes a prediction for a training instance, and the error is calculated.
  - The model is updated to reduce the error for the next prediction.
  - This process is repeated, iterating through the training dataset one instance at a time.
  - The coefficients (weights) are updated at each iteration according to the gradient of the error with respect to the training instance.

Linear Regression Machine Learning Algorithms

Data Scientist → Linear Regression

Agenda

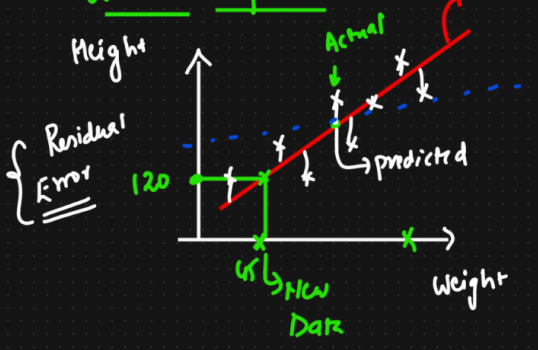
1. What problem we are solving
2. Geometric Intuition
3. Mathematical Intuition

① Machine Learning

- Supervised
  - Regression (Continuous) — Linear Regression
  - Classification
- Unsupervised

Aim : To find best fit line

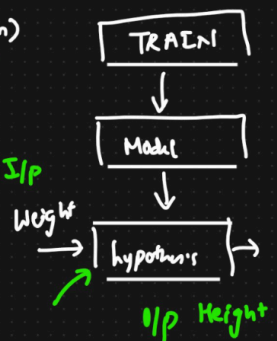
Linear Regression ✓



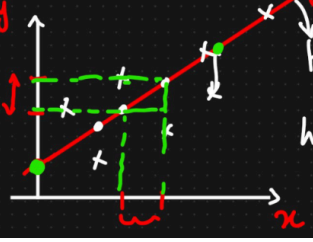
Minimal Error

Weight (kg)	Height (cm)
73	150
63	155
78	170
82	165

O/p



Maths {Basic Maths}



$\hat{y} = mx + c$

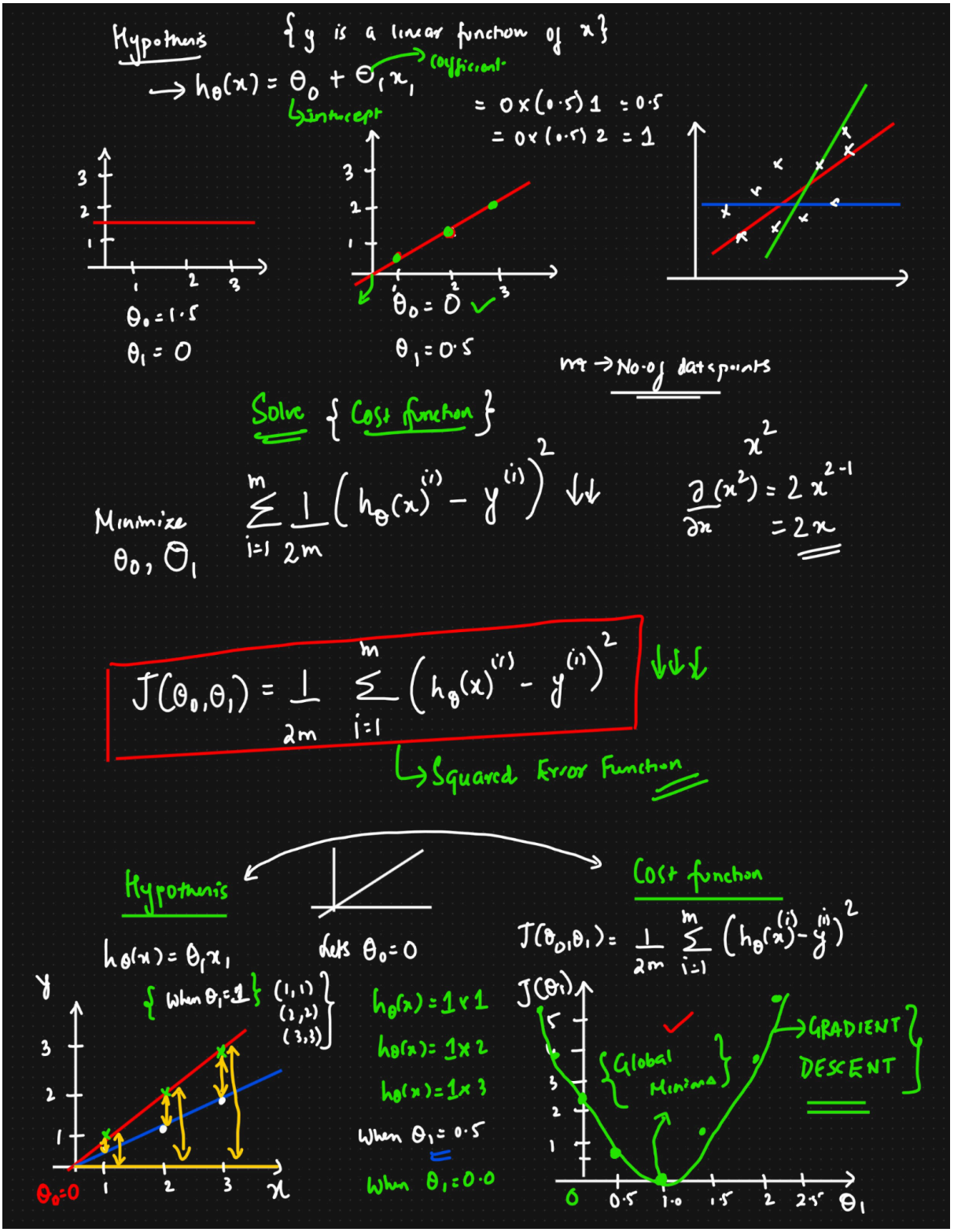
$h_\theta(x) = \theta_0 + \theta_1 x_1$

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots \theta_n x_n$

m = Slope or Coefficient

c = Intercept

Andrew NG


Hypothesis
{y is a linear function of x}
$h_\theta(x) = \theta_0 + \theta_1 x_1$
coefficient
y-intercept
$= 0 \times (0.5)\,1 = 0.5$
$= 0 \times (0.5)\,2 = 1$
$\theta_0 = 1.5$
$\theta_1 = 0$
$\theta_0 = 0$
$\theta_1 = 0.5$
m → No. of datapoints
Solve {Cost function}
Minimize θ_0, θ_1
$\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x)^{(i)} - y^{(i)}\right)^2$
$\frac{\partial}{\partial x}(x^2) = 2x^{2-1} = 2x$
$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x)^{(i)} - y^{(i)}\right)^2$
Squared Error Function
Hypothesis
$h_\theta(x) = \theta_1 x_1$
{When $\theta_1 = 1$} (1,1) (2,2) (3,3)
Lets $\theta_0 = 0$
$h_\theta(x) = 1 \times 1$
$h_\theta(x) = 1 \times 2$
$h_\theta(x) = 1 \times 3$
When $\theta_1 = 0.5$
When $\theta_1 = 0.0$
Cost function
$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$
J(θ1)
Global Minima
GRADIENT DESCENT
$\theta_0 = 0$
θ_1

$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x)^{(i)} - y^{(i)})^2$ when $\theta_1 = 1$

$= \frac{1}{6}[(0)^2 + (0)^2 + (0)^2]$

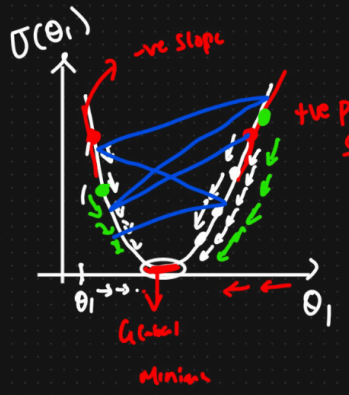
$= 0$

$J(\theta_1) = \frac{1}{6}[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2]$ when $\theta_1 = 0.5$

$= \frac{1}{6}(3.5) = \approx 0.58$

$J(\theta_1) = \frac{1}{6}[(0-1)^2 + (0-2)^2 + (0-3)^2]$

$= \frac{14}{6}$ or ≈ 2.3



$\theta_1 \rightarrow$ update $\rightarrow$ find derivation or slope

$\alpha = 1000$ { Learning Rate }

$\alpha = 0.01$

Repeat convergence theorem

$\{$

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j}(J(\theta_1))$ ← Slope

$\}$

$\theta_1 := \theta_1 - \alpha(+ve)$ } Decrease ✓

$\theta_1 := \theta_1 - \alpha(-ve)$ } Increase ✓ (Slope)

$\theta_1 = \theta_1 +$

Outline

1. Start with θ_0 & θ_1
2. Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we reach near global Minima.
3. Convergence Theorem

$\{$

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

$\}$

Multiple Linear Regression

Multiple Linear Regression (MLR) is an extension of simple linear regression to predict an outcome based on multiple explanatory variables. The model is given by the equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon$$

Here, Y is the dependent variable, $X_1, X_2, \ldots, X_n$ are the independent variables, β_0 is the intercept, $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients for each independent variable, and ϵ is the error term.

The coefficients are estimated using the least squares method, which minimizes the sum of squared residuals between the predicted and actual values.

The loss function commonly associated with MLR is the Mean Squared Error (MSE), which is the average of the squared differences between the observed actual outomes and the outcomes predicted by the model.

The learning rate is a hyperparameter in iterative training algorithms like gradient descent. It determines the size of the steps taken towards the minimum of the loss function. If the learning rate is too high, it may overshoot the minimum; if it's too low, the algorithm may take too long to converge or get stuck in a local minimum.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3}))^2$$

In building regression models, there are several strategies you can use to select the appropriate variables:

All-in: This approach involves using all available variables in the model. It's often used when you have prior knowledge that all variables are important, or when you are required to use all variables for some reason, such as theoretical considerations or regulatory requirements.

Backward Elimination: You start with all variables and remove the least significant variable (the one with the highest p-value greater than a chosen threshold) one by one until you're left with a set of only statistically significant variables.

Forward Selection: This is the opposite of backward elimination. You start with no variables and add the most significant variable step by step. At each step, you add

the variable that has the most statistical significance in explaining the response variable until no significant variables are left to add.

Bi-directional Elimination (Stepwise Regression): This is a combination of forward selection and backward elimination. You add variables as in forward selection, but after each addition, use backward elimination to remove any variable that does not meet a significance criterion in the updated model.

Score Comparison: This method involves building a set of models, each with a different subset of variables, and then comparing their performance using a particular criterion, such as the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC). You select the model with the best (usually the lowest) score.

Multiple linear regression is a statistical technique that models the relationship between two or more independent variables and a single continuous dependent variable. It is often used to understand the influence of several independent variables on the dependent variable.

Causal Relationships:

When discussing causal relationships in the context of multiple linear regression, it's important to clarify that regression alone typically cannot prove causation, but it can suggest possible causal relationships that are worth further investigation. Here are ways in which multiple linear regression can be helpful:

Control for Confounding Variables: By including multiple variables in the regression model, researchers can control for confounding factors that might otherwise skew the results. This allows for a clearer picture of the relationships between specific independent variables and the dependent variable.
Assess Effect Size: Multiple linear regression provides coefficients that quantify the size of the effect that each independent variable has on the dependent variable, holding other variables constant.
Interaction Effects: It can also identify interaction effects between variables, which can be indicative of complex causal mechanisms.
Predictive Modeling: Although not directly indicative of causation, a well-fitting model suggests that the independent variables have some predictive power over the dependent variable.

Limitations:

However, to suggest causality, certain conditions must be met, such as temporal precedence (the cause must precede the effect), covariation (changes in the cause lead to changes in the effect), and the elimination of alternative explanations (no other factors are causing the effect).

Further Analysis for Causation:

To establish causation, additional analyses beyond multiple linear regression are typically required, such as randomized controlled trials or longitudinal studies. Additionally, researchers must have a theoretical basis for any causal claims and should be cautious of spurious correlations that can arise due to omitted variable bias or reverse causality.

Multiple linear regression can be a powerful tool for identifying potential causal relationships, but it must be used as part of a broader analytical approach that includes careful study design, theoretical reasoning, and potentially experimental methods to confirm causation.

Polynomial Regression

Polynomial Linear Regression is a form of regression analysis in which the relationship between the independent variable ***x*** and the dependent variable ***y*** is modeled as an ***n***th degree polynomial. It works by transforming the original features into polynomial features of a specified degree and then applying linear regression on these transformed features.

When to Use It:

- It is used when the data shows a non-linear relationship that a simple linear model cannot fit accurately.

Pros:

- Can model non-linear relationships.
- Provides a better fit for some datasets than simple linear regression.

Cons:

- Can lead to overfitting if the polynomial degree is too high.
- The model becomes complex and computationally intensive with increasing degree.

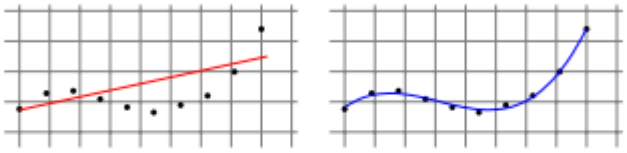
Polynomial Linear Regression is effective for non-linear datasets, but care must be taken to choose the right degree to avoid overfitting.

Linear Regression for Polynomial Regression Tasks

Some learning tasks call for nonlinear predictors, such as polynomial predictors. Take, for instance, a one dimensional polynomial function of degree n, that is,

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

where $(a_0, \ldots, a_n)$ is a vector of coefficients of size $n + 1$. In the following we depict a training set that is better fitted using a 3rd degree polynomial predictor than using a linear predictor.



We will focus here on the class of one dimensional, n-degree, polynomial regression predictors, namely,

$$\mathcal{H}^n_{poly} = \{x \mapsto p(x)\},$$

where p is a one dimensional polynomial of degree n, parameterized by a vector of coefficients $(a_0, \ldots, a_n)$. Note that $\mathcal{X} = \mathbb{R}$, since this is a one dimensional polynomial, and $\mathcal{Y} = \mathbb{R}$, as this is a regression problem.

One way to learn this class is by reduction to the problem of linear regression, which we have already shown how to solve. To translate a polynomial regression problem to a linear regression problem, we define the mapping $\psi : \mathbb{R} \to \mathbb{R}^{n+1}$ such that $\psi(x) = (1, x, x^2, \ldots, x^n)$. Then we have that

$$p(\psi(x)) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \langle \mathbf{a}, \psi(x) \rangle$$

and we can find the optimal vector of coefficients $\mathbf{a}$ by using the Least Squares algorithm as shown earlier.

Gradient Descent Algorithms

Gradient Descent is an optimization algorithm used for finding the minimum of a function. In the context of machine learning, it's often used to minimize a cost function.

Mathematical Notation:

The algorithm updates the parameters, represented as θ, of the model by moving them in the direction of the steepest descent of the cost function, denoted as $J(\theta)$. The update rule is given by:

$$\theta := \theta - \alpha \frac{\partial}{\partial \theta} J(\theta)$$

where α is the learning rate, and $\frac{\partial}{\partial \theta} J(\theta)$ is the gradient of the cost function.

Working of Gradient Descent:

- Start with initial guesses for the model parameters.
- Calculate the gradient of the cost function at the current parameter values.
- Update the parameters in the opposite direction of the gradient, scaled by the learning rate.
- Repeat the process until the cost function converges to a minimum.

TYPES

Batch Gradient Descent: This algorithm computes the gradient of the cost function for the entire training dataset. It's accurate but can be very slow and computationally expensive, especially with large datasets. Batch Gradient Descent can also have trouble escaping local minima.

Stochastic Gradient Descent (SGD): Instead of using the whole dataset, SGD updates parameters for each training example one by one. It's faster and can handle large datasets efficiently. However, its frequent updates result in significant fluctuations, which can help escape local minima but make convergence to the exact minimum difficult.

Stochastic Gradient Descent (SGD)

In stochastic gradient descent we do not require the update direction to be based exactly on the gradient. Instead, we allow the direction to be a random vector and only require that its *expected value* at each iteration will equal the gradient direction. Or, more generally, we require that the expected value of the random vector will be a subgradient of the function at the current vector.

Stochastic Gradient Descent (SGD) for minimizing $f(\mathbf{w})$

parameters: Scalar $\eta > 0$, integer $T > 0$
initialize: $\mathbf{w}^{(1)} = \mathbf{0}$
for $t = 1, 2, \ldots, T$
  choose $\mathbf{v}_t$ at random from a distribution such that $\mathbb{E}[\mathbf{v}_t \mid \mathbf{w}^{(t)}] \in \partial f(\mathbf{w}^{(t)})$
  update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t$
output $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

Mini-Batch Gradient Descent: This method is a compromise between Batch and Stochastic Gradient Descent. It updates parameters for a small subset of the training data at each step. This balances the efficiency of SGD with the precision of Batch Gradient Descent. It's typically the preferred method due to its robustness and efficiency.

Gradient Descent is a fundamental optimization algorithm used in machine learning to minimize a function. Here's a step-by-step explanation of how it works, using an example:

1. **Define the Cost Function**: In the context of linear regression, the cost function often used is the Mean Squared Error (MSE), which measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value.
2. **Initialize Parameters**: Choose initial values for the parameters you are trying to estimate. For simple linear regression, these would be the slope (m) and intercept (b) of the line $y = mx + b$.
3. **Calculate the Gradient**: The gradient is a vector that tells us in which direction the function increases most rapidly. To minimize the cost, we want to go in the opposite direction of the gradient. For MSE, you would partially differentiate the cost function with respect to each parameter to find the gradient.
4. **Update Parameters**: Adjust the parameters in the direction that reduces the cost function. This is done by subtracting a fraction of the gradient from the current parameters. The fraction is determined by the learning rate (α), a hyperparameter that you choose in advance.
5. **Repeat**: Perform steps 3 and 4 iteratively until the cost function stops decreasing or decreases very slowly. Each iteration is called a step, and the learning rate determines the size of each step.
6. **Convergence**: Ideally, the algorithm converges to the minimum cost, which corresponds to the best estimates for the parameters.

Example:

Assume we have a dataset of points (x_i, y_i), and we are trying to fit a line to these points. Our initial guess is that $m = 0$ and $b = 0$, so our initial line is $y = 0$.

- We calculate the error (or cost) for our line's predictions using MSE.
- We find the gradient of the MSE with respect to m and b.
- We update m and b by subtracting α times their respective gradients from them.

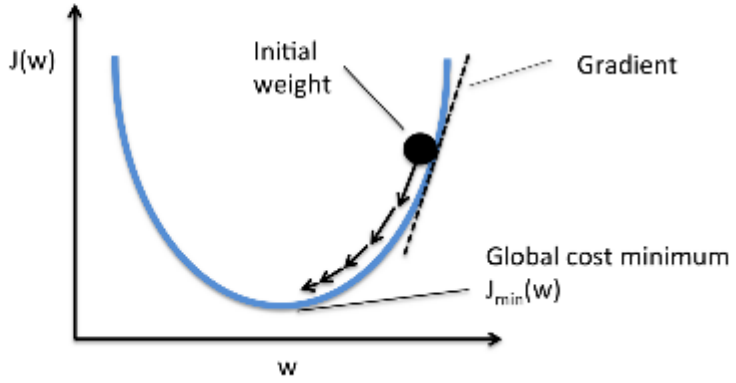
↓

For instance, if after calculating the gradients, we find that $\frac{\partial MSE}{\partial m} = 2$ and $\frac{\partial MSE}{\partial b} = 3$, and we choose $\alpha = 0.01$, then:

- New $m = m - \alpha \times \frac{\partial MSE}{\partial m} = 0 - 0.01 \times 2 = -0.02$
- New $b = b - \alpha \times \frac{\partial MSE}{\partial b} = 0 - 0.01 \times 3 = -0.03$

This process repeats, adjusting m and b each time, until the change in MSE between iterations is negligible, indicating that we've arrived at the minimum cost. The values of m and b at this point give us the line of best fit according to our data and the MSE cost function.

cost function is a parabola curve



Regression Splines

Layman Terms Explanation with Examples

Imagine you're trying to draw a line through a series of dots on a graph so that the line fits the dots as well as possible. But the dots don't form a straight line; they curve. If you tried to use a single straight line (like in simple linear regression) or even a simple curve (polynomial regression), you might not get a very good fit, especially if the dots have different patterns in different areas.

Regression Splines are like drawing a piece of a line or a gentle curve between some dots, then picking up a new piece of line or curve for the next few dots, and so on. The places where you switch from one piece to another are called "knots." This way, you can follow the pattern of the dots closely without making your drawing too wiggly or complex.

Example : Think of it like drawing a path through a park that has different types of scenery. In one area, the path might need to curve around a pond. In another area, it might go straight through a field. Instead of making one simple path for the entire park, you adjust your path to fit each section best.

Formal Explanation with Examples

In statistics and machine learning, Regression Splines are a sophisticated technique for modeling relationships between variables when the relationship is nonlinear. Unlike linear regression, which fits a single line to data, or polynomial regression, which fits a global polynomial curve, regression splines divide the dataset into sections and fit a simple model, like a linear or low-degree polynomial, within each section.

Knots are the points at which the data is divided, and within each of these intervals, a piecewise polynomial function is applied. This allows for flexible modeling that can adapt to different patterns in different segments of the data.

Piecewise Polynomials and Splines : By fitting low-degree polynomials within each segment divided by knots, we avoid the high variability and overfitting risks associated with high-degree polynomials. When these piecewise polynomials are made to join smoothly at the knots (ensuring continuity and smooth transitions), they form what are known as splines.

Example : A real-world application could involve modeling the growth rate of plants under varying light conditions. Instead of assuming a single growth pattern across all light intensities, splines allow us to model different growth patterns for low, medium, and high light intensities, with knots representing the transition points between these conditions.

Regression Splines are a tool used in regression to add non-linearity to a model. They involve dividing the data into distinct regions and fitting separate polynomials to each region. This approach maintains continuity and differentiability at the boundaries (knots).

The Solution-Regression Splines approach refers to the specific method of solving regression problems using splines. It involves choosing the number and positions of the knots, which can significantly impact the model's flexibility.

Studying Regression Splines is important because they provide a way to model complex relationships without assuming a global form for the entire data set.

Piecewise Step Functions divide the range of data into intervals and fit a different constant (step) in each interval. This method is simpler than polynomials but less smooth.

For forming piecewise polynomials, necessary conditions and constraints include ensuring continuity at the knots, smoothness, and often differentiability. The specific constraints depend on the degree of the spline and the desired smoothness at the knots.

Cubic Splines are piecewise-polynomial functions used in regression, divided into segments by knots, where each segment is a cubic polynomial. They ensure continuity, smoothness, and differentiability at the knots. Natural Cubic Splines further constrain the function to be linear beyond the boundary knots, which helps in handling the extrapolation problem and avoiding extreme variations at the boundaries.

Choosing the number and locations of the knots in cubic splines is crucial. Too few knots can oversimplify the model, while too many can lead to overfitting. The choice is often based on domain knowledge, data characteristics, or using methods like cross-validation. The knots are usually placed where there are significant changes in the behavior of the data, but finding their optimal placement can be challenging and requires careful consideration.

What are Regression Splines?
Regression Splines are methods in statistics to fit non-linear trends in data using piecewise polynomial functions.

How does Polynomial Regression differ from Linear Regression?
While Linear Regression assumes a straight-line relationship between variables, Polynomial Regression fits a curved line, which is more flexible.

What are Solution-Regression Splines?
Solution-Regression Splines use a mix of linear or polynomial functions to fit different segments (or bins) of the dataset.

What are 'Knots' in Regression Splines?
Knots are the points where the data is split into different segments or bins for separate model fitting.

What are Piecewise Step Functions?
These functions are constant within intervals and change abruptly at the boundaries, used to fit different constants in data bins.

How do Piecewise Polynomials work?
They involve fitting separate low-degree polynomials over different regions of the data instead of a single polynomial over the entire range.

What is a Piecewise Cubic Polynomial?

It fits different cubic polynomial functions to the data based on specific segments divided by knots.

Why use more knots in Piecewise Polynomials?
More knots allow for greater flexibility as different functions can be used for each data bin.

What are necessary conditions for forming piecewise polynomials?
They must be continuous at knots, with constraints for smoothness and derivative continuity.

What are Cubic and Natural Cubic Splines?
Cubic Splines use piecewise polynomials with extra constraints for continuity and derivatives, while Natural Splines are a type that is linear beyond boundary knots.

How do you choose the number and locations of knots?
Placement can be based on areas of high data variability, uniform distribution, or using methods like cross-validation for an objective approach.

What is Cross-Validation in this context?
It's a method to determine the best number of knots by iteratively fitting the spline, leaving out parts of the data, and calculating the error.

K-Nearest Neighbor

K-Nearest Neighbor (KNN) is a simple, versatile machine learning algorithm used for both classification and regression.

Why Called a 'Lazy Algorithm'?
KNN is considered 'lazy' because it doesn't learn a discriminative function from the training data but memorizes the dataset instead.

How Does KNN Perform Regression?
In regression, KNN predicts the output for a new data point by averaging the values of the K nearest neighbors.

Key Concepts:

- Non-parametric: KNN makes no assumptions about the functional form of the problem being solved, making it versatile for modeling complex relationships where the structure is unknown.

- Distance Metric: The algorithm uses a distance metric, typically Euclidean distance, to identify the 'K' training samples closest to the new point. Other metrics like Manhattan or Minkowski distance can also be used depending on the dataset.
- 'K' Value: The choice of 'K' (the number of nearest neighbors to consider) is crucial. A smaller 'K' can make the model sensitive to noise in the data, while a larger 'K' makes the boundary smoother and the model less sensitive to outliers but might blur distinctions between classes.
- Weighted Average: Sometimes, the contribution of each of the 'K' neighbors to the prediction is weighted by their distance to the query point, giving closer neighbors more influence on the prediction.

How Does It Work?

- It stores all the training data.
- For a new data point, it searches the entire training set for the K closest points.
- KNN uses a distance metric like Euclidean distance to find these points.
- For classification, it assigns the class based on the majority vote among the K neighbors, while for regression, it takes the average of their values.

KNN's simplicity is its strength, making it easy to implement and understand. However, its performance might decrease with very large datasets or in cases where dimensions are high (curse of dimensionality).

Layman's Terms Explanation

Imagine you have a bunch of apples and oranges mixed together, and you want to figure out if a new fruit from the basket is an apple or an orange. You decide to look at the 3 fruits closest to this new one. If 2 of them are apples and 1 is an orange, you'd guess the new fruit is an apple since most of its "neighbors" are apples. That's essentially how KNN works.

Example: Suppose you're at a pet adoption center with dogs and cats. You want to determine whether a new animal (let's say, an animal you found no information about) is more like the dogs or cats you see. You decide to look at the 5 nearest animals to this new one. If 4 out of these 5 are cats, you might guess this new animal is probably a cat too. This method of making a decision based on the 'nearest neighbors' is what KNN is all about.

Formal Explanation

K-Nearest Neighbors (KNN) is a type of supervised machine learning algorithm used for both classification and regression tasks, but it is more widely used for classification problems. The algorithm operates on the principle that similar things exist in close proximity, i.e., "birds of a feather flock together."

In KNN, the value of 'K' is a critical factor. 'K' refers to the number of nearest neighbors to include in the majority vote for classification or average for regression.

Distance Measure: To identify the nearest neighbors, KNN uses a distance measure — typically Euclidean distance, although other distances like Manhattan or Minkowski can also be used.
Majority Voting or Averaging: For classification, the algorithm assigns the class based on the majority class among the k-nearest neighbors. For regression, it calculates the average of the values of the k-nearest neighbors.
No Training Phase: Unlike other machine learning algorithms, KNN doesn't learn a discriminative function from the training data but memorizes the dataset instead.

Example: Consider a dataset with academic records of students categorized into "Passed" and "Failed" based on features like hours studied, attendance, etc. For a new student record, KNN would look at the 'K' nearest students (based on hours studied and attendance) to predict whether the new student passed or failed. If 'K' is set to 3, and two of the three nearest neighbors passed their exams, the algorithm would predict that the new student also passed.

KNN is simple and effective, especially in cases where the decision boundary is very irregular. However, its performance can be significantly affected by the choice of 'K' and the distance metric used. Additionally, it can become computationally expensive as the dataset grows, since it requires calculating the distance to every training example for each prediction.

Problem Statement:

Imagine we have a dataset with house sizes (in square feet) and their corresponding prices (in thousands of dollars). We want to predict the price of a new house (House X) with a size of 1,500 square feet using KNN for regression with $K = 3$.

Here's our small dataset of known house sizes and prices:

House	Size (sq ft)	Price ($k)
A	1400	300
B	1600	330
C	1700	360
D	1800	380
E	1900	400

Steps to Solve:

1. **Calculate Distances**: First, we need to calculate the Euclidean distance between House X and each of the houses in our dataset. The Euclidean distance between two points x and y is given by $\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$. Since we only have one dimension (size), this simplifies to $\sqrt{(size_x - size_y)^2}$.
2. **Identify Nearest Neighbors**: After calculating the distances, we identify the 3 nearest neighbors to House X based on these distances.
3. **Compute Average Price**: The predicted price for House X is the average price of its 3 nearest neighbors.

Distances Computed:

- Distance from House A (1,400 sq ft) to House X (1,500 sq ft): $|1400 - 1500| = 100$
- Distance from House B (1,600 sq ft) to House X: $|1600 - 1500| = 100$
- Distance from House C (1,700 sq ft) to House X: $|1700 - 1500| = 200$
- Distance from House D (1,800 sq ft) to House X: $|1800 - 1500| = 300$
- Distance from House E (1,900 sq ft) to House X: $|1900 - 1500| = 400$

Identify Nearest Neighbors: We then sorted these distances to find the 3 nearest neighbors to House X. The houses with the smallest distances are the nearest neighbors. In this case, Houses A and B are each 100 units away, and House C is 200 units away, making them the 3 nearest neighbors.

Compute Average Price: The final step was to calculate the average price of these 3 nearest neighbors as the prediction for House X's price.

- Prices of the nearest neighbors: House A ($300k), House B ($330k), and House C ($360k).
- Average price: $\frac{300+330+360}{3} = 330$.

Therefore, based on the KNN regression model with $K = 3$, the predicted price for House X is $330,000. This approach allows us to make predictions by leveraging the known outcomes of the nearest neighbors in our dataset.

Hindi-K Nearest Neighbour Indepth Intuition- Classification And Regression

Decision Tree

A Decision Tree is a flowchart-like tree structure used in machine learning for both classification and regression tasks. It breaks down a dataset into smaller subsets while an associated decision tree is incrementally developed.

Pros:

- Easy to understand and interpret.
- Requires little data preparation.
- Can handle both numerical and categorical data.
- Performs well with large datasets.

Cons:

- Prone to overfitting, especially with complex trees.
- Can become unstable with small variations in data.
- Biased with imbalanced datasets.

Algorithm:

- Starts at the tree's root and splits the data based on a feature that results in the largest information gain (IG) in classification or lowest MSE in regression.
- Repeats this process recursively until it meets a stopping criterion, like a maximum depth or minimum number of samples required to split a node.

Example (Regression):

Consider a dataset with house attributes (size, number of rooms) and their prices. A decision tree for regression might first split the data based on the number of rooms (e.g., fewer or more than 3 rooms), and then further split each subset based on size. The predicted price for a new house would be the average price of similar houses in the corresponding leaf node.

Numerical Question:

Suppose you are a real estate analyst trying to predict the price of houses based on their size (in square feet) and the number of bedrooms. You have collected data on several houses in a region, and now you want to build a Decision Tree Regression model to make your predictions.

The dataset you have is as follows:

House	Size (sq ft)	Bedrooms	Price ($)
1	2100	3	400,000
2	1600	2	330,000
3	2400	4	450,000
4	1416	2	232,000
5	3000	4	540,000

Your task is to predict the price of a new house (House 6) that is 1800 square feet and has 3 bedrooms.

Steps to Solve Using Decision Tree Regression:

Feature Selection: In this problem, the "Size (sq ft)" and "Bedrooms" are the independent variables (features), and "Price ($)" is the dependent variable (target).
Splitting the Dataset: Normally, you would split your dataset into a training set and a testing set. However, since this is a simple example, we'll use the entire dataset to train the model and make a prediction for the new house.
Building the Tree:

- Step 1: Start by considering the entire dataset as the root node.
- Step 2: Determine the best split based on minimizing variance within nodes. For simplicity, let's assume our model decides to first split by "Size (sq ft)" at 2000 sq ft, dividing our houses into two groups: one with houses smaller than or equal to 2000 sq ft and one with houses larger.

- Step 3: For each of these groups, it further splits based on "Bedrooms" or continues with "Size" if that reduces the variance in price more. Suppose it then splits the first group (≤2000 sq ft) at "2 Bedrooms", separating houses into those with 2 bedrooms or less and those with more.
- Step 4: Repeat the splitting process until further splits do not significantly reduce variance or the tree reaches a predetermined depth to prevent overfitting. In our simplified case, let's stop at these splits.

Making a Prediction:

- Step 1: To predict the price of House 6, follow the decisions in the tree based on its features: 1800 sq ft and 3 bedrooms.
- Step 2: First, it goes to the node for houses ≤2000 sq ft.
- Step 3: Within this node, it falls into the category of houses with more than 2 bedrooms.
- Step 4: The prediction for House 6 would be the average price of houses in this final node. Based on our dataset, it includes House 2 (330,000 $) and potentially others if we had more data. For this example, let's say the predicted price is 330,000 $.

Considerations:

- Real Application: In practice, you would use a software library like `scikit-learn` in Python to build the decision tree. It would automatically handle feature selection, tree construction, and prediction.
- Model Evaluation: Normally, you would also evaluate your model's performance using part of your dataset reserved for testing, using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or R-squared.

Part 5-Hindi- Decision Tree Regression Indepth Intuition With Variance Redu...

Support Vector Regression

A Support Vector Machine (SVM) is a powerful and versatile supervised machine learning model, used for both classification and regression tasks. It works by finding the hyperplane that best separates the classes in the feature space.

Why and How to Use SVM:

- It's effective in high dimensional spaces and when there is a clear margin of separation in the data.

- SVM is used by maximizing the margin between the data points of different classes, thus creating a well-generalized classification boundary.

When to Use SVM:

It's best used in classification problems with complex but small- or medium-sized datasets.

Support Vector Regression (SVR):

- SVR uses the same principles as SVM but for regression problems.
- Instead of fitting the widest possible street between classes, SVR fits as many instances as possible on the street while limiting margin violations.
- The goal is to estimate a function that deviates from the actual observed values by a value no greater than a specified tolerance.

A **hyperplane** in machine learning, especially in the context of Support Vector Machines (SVM), is essentially a decision boundary that separates different classes of data. It can be thought of as a line in two dimensions or a plane in higher dimensions.

Mathematical Representation:

A hyperplane in an n-dimensional space is represented as:

$$\sum_{i=1}^{n} w_i x_i - b = 0$$

where w is the weight vector, x represents the feature vectors, and b is the bias.

Difference from Ordinary Regression:

- In traditional regression, the goal is to minimize the error between the predicted and actual values.
- In SVM (using a hyperplane), the objective is to find the best margin (distance) that separates the classes with a maximum margin while minimizing classification errors.

This approach is fundamentally different in terms of the objective (maximizing margin vs. minimizing error) and the nature of the problem being solved (classification vs. regression).

Support Vector Regression (SVR) works by following these steps:

Selecting a Kernel and Regularization Parameter: Choose a kernel (like linear, polynomial, or RBF) and set the regularization parameter (C). The kernel transforms the data into a higher-dimensional space, while C balances the

tradeoff between a smooth decision surface and classifying training points correctly.

Creating a Buffer or Margin: Unlike in SVM for classification, in SVR, we create a margin of tolerance (epsilon) where no penalty is given to errors.

Fitting the Model: SVR tries to fit the error within this margin while also minimizing the model complexity.

Predicting Values: For new inputs, SVR predicts values based on the function it derived during training.

Example: Imagine predicting house prices based on features like size and location. SVR will determine a function based on training data (known house prices) that predicts the price of a house while keeping the errors within a certain margin. If the actual price of a house falls within this margin from the predicted price, it's considered acceptable.

The Kernel Trick is a technique used in SVMs to transform data that is not linearly separable in its original space into a higher-dimensional space where it becomes linearly separable. This allows SVM to create non-linear decision boundaries using methods designed for linear classifiers.

Common Applications of SVM:
- Image classification
- Text categorization
- Handwriting recognition
- Bioinformatics (like protein classification)

Pros and Cons of SVR:
- **Pros**:
  - Effective in high-dimensional spaces.
  - Works well with a clear margin of separation.
  - Robust against overfitting, especially in high-dimensional spaces.

- **Cons**:
  - Requires careful tuning of parameters.
  - Not suitable for larger datasets due to its high computational complexity.
  - Doesn't perform well with a lot of noise and overlapping classes.

SVM (Support Vector Machines) ML Algorithms

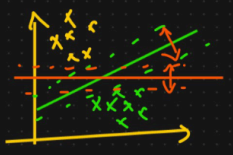
Tasks

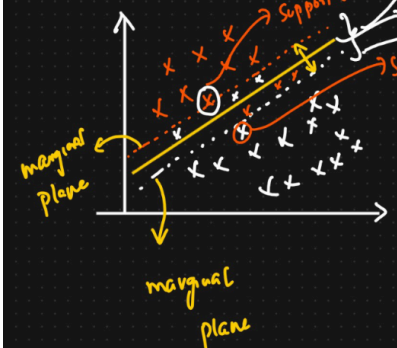
① Classification ✓ ② Regression

Support Vector Classifier → Hard marginal plane ✓

Support vectors, Hyperplane, Support vectors, Soft marginal plane

Arm





$y = mx + c$ $\quad y = \theta_0 + \theta_1 x \quad y = \beta_0 + \beta_1 x$

$y = w_1x_1 + w_2x_2 + w_3x_3 + b$

$y = w^T x + b$

$y = mx + c$

$\Updownarrow$

$ax + by + c = 0$ → Equation of a straight line

$by = -ax - c$

$y = -\frac{a}{b}(x) - \frac{c}{b}$

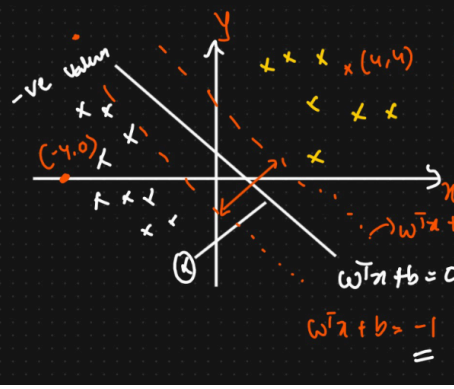
$m = \frac{-a}{b}$

$c = \frac{-c}{b}$

$3(4) + 2(4) + 4 = 0$

$12 + 8 + 4$

$24 \Rightarrow$ +ve



$3x + 2y + 4 = 0$

$3(-4) + 2(0) + 4$

$-12 + 0 + 4$

$= -8$

2 components

$w \rightarrow$ magnitude, vectors

$\vec{w}$

$w^T x_1 + b = +1$

$w^T x_2 + b = -1$

$(-) \quad 2 \quad (-) \quad (+)$

$\vec{w}$ $\quad \dfrac{w^T(x_1 - x_2)}{\|w\|} = \dfrac{2}{\|w\|} \Rightarrow$ Maximize

$\underset{(w,b)}{\text{Maximize}} = \dfrac{2}{\|w\|} \Rightarrow$ Marginal plane distance

Constraint $\quad$ Such that $\quad y_i = \begin{cases} +1 & w^T x + b \geq 1 \\ -1 & w^T x + b \leq -1 \end{cases}$

For all annuate datapoint

$$y * (w^T x_i + b) \geq 1 \Rightarrow \text{Constraint}$$

$\underset{(w,b)}{\text{Maximize}} = \dfrac{2}{\|w\|} \Rightarrow \underset{(w,b)}{\text{Minimize}} = \dfrac{\|w\|}{2}$

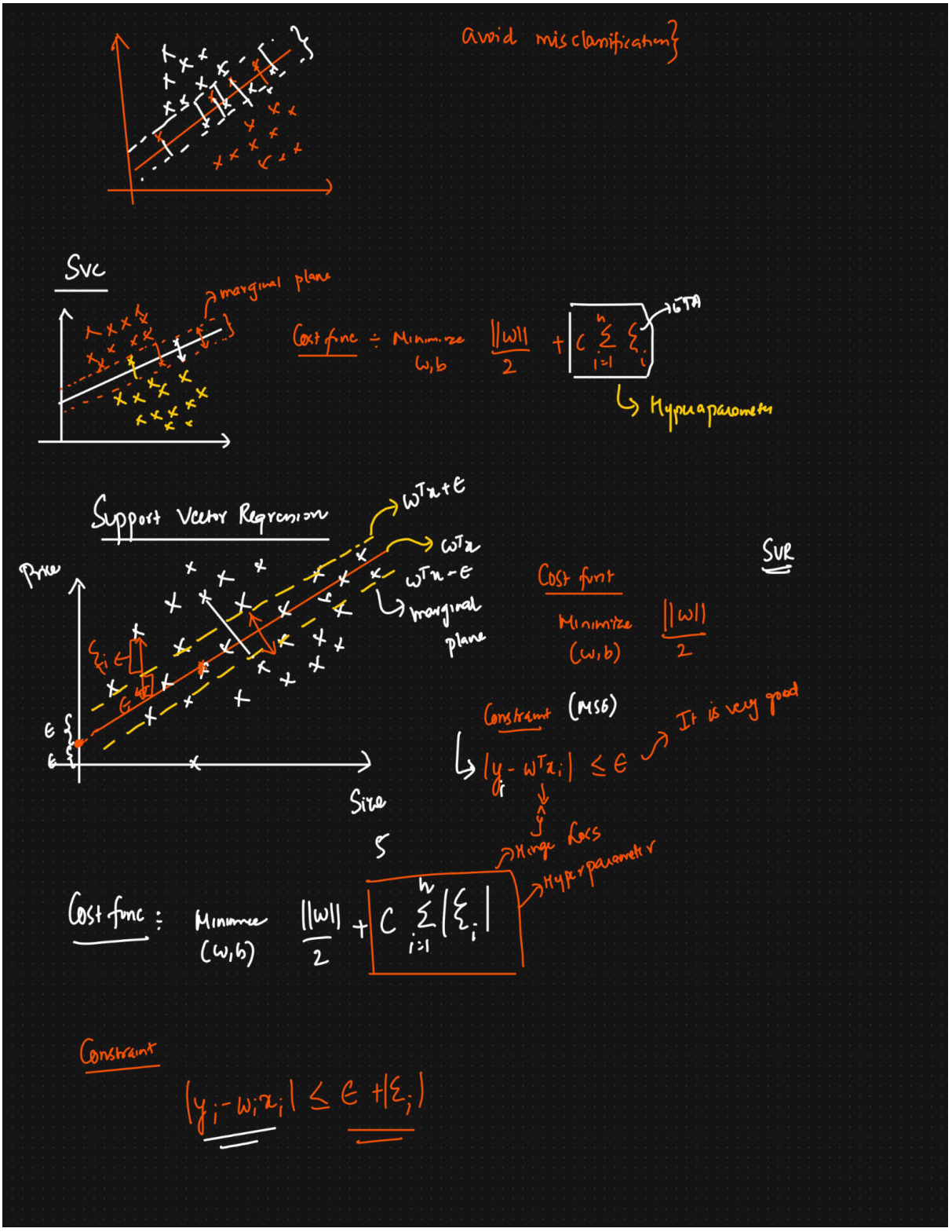
Cost function : $\underset{(w,b)}{\text{Minimize}} \dfrac{\|w\|}{2} + C_1 \sum_{i=1}^{n} \xi_i^{\text{pow}}$

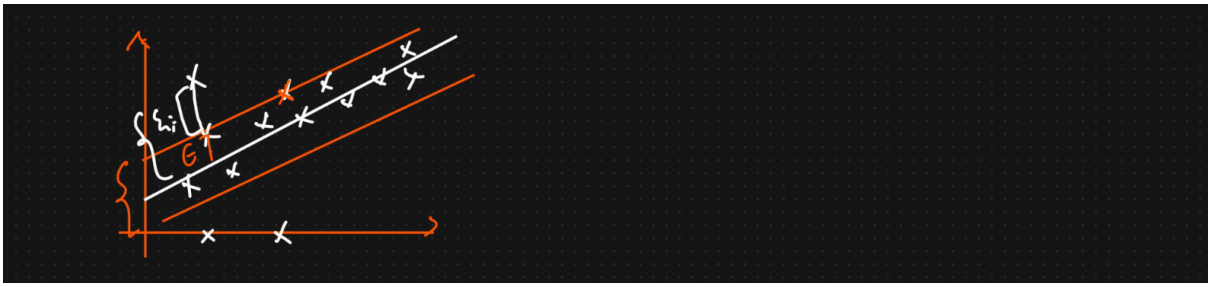
w

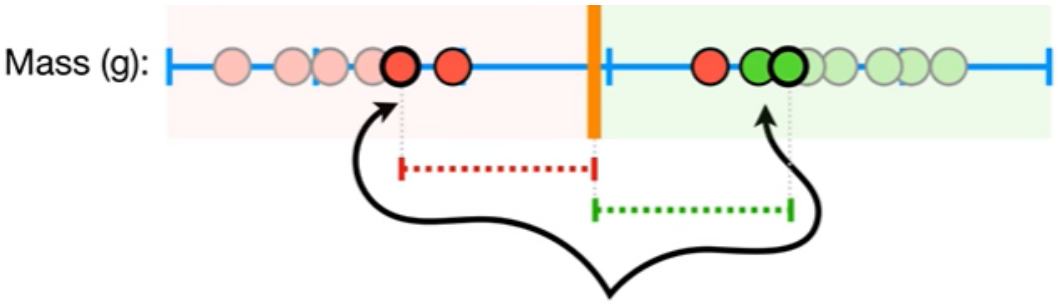
ξ_i {Summation of the distance of misclassification points from marginal plane}

{How many points we can}

$C = 8$


avoid misclassification
SVC
marginal plane
Cost func : Minimize ||w|| / 2 + C Σ ξi
w,b
Hyperparameter
Support Vector Regression
wTx+ε
wTx
wTx-ε
marginal plane
Price
Size
SVR
Cost func
Minimize ||w|| / 2
(w,b)
Constraint (MSE)
|yi - wTxi| ≤ ε
It is very good
Hinge Loss
Hyperparameter
Cost func : Minimize ||w|| / 2 + C Σ |ξi|
(w,b)
Constraint
|yi - wixi| ≤ ε + |ξi|





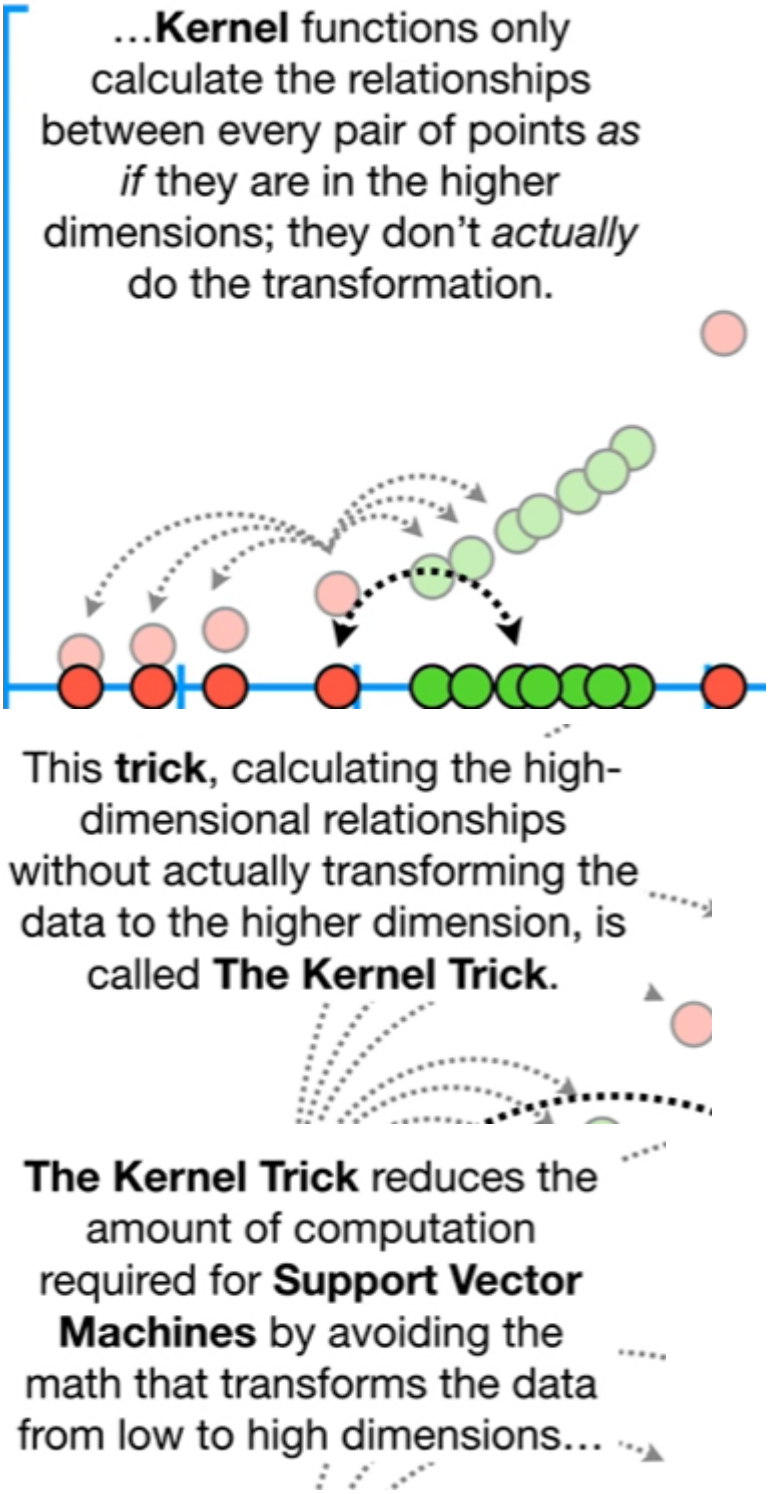
The name **Support Vector Classifier** comes from the fact that the observations on the edge *and within* the **Soft Margin** are called **Support Vectors**.

And when the data are in **4 or more Dimensions**, the **Support Vector Classifier** is a *hyperplane*.

psst! In mathematical jargon, a hyperplane is a "flat affine subspace".

NOTE: Technically speaking, all flat affine subspaces are called **hyperplanes**.

In order to make the mathematics possible, **Support Vector Machines** use something called **Kernel Functions** to *systematically* find **Support Vector Classifiers** in higher dimensions.



Support Vector Regression (SVR) is an extension of the Support Vector Machine (SVM) algorithm to handle regression problems. SVR applies the principles of SVM to predict a continuous variable, unlike SVM which is used for classification tasks. The goal of SVR is to find a function that has at most ε deviation from the actual target values of the training data and at the same time is as flat as possible.

How SVR Works:

- Fitting the Best Line/Plane/Curve: In SVR, the idea is to fit the best line (in two dimensions), plane (in three dimensions), or hyperplane (in more than three dimensions) within a threshold value, ε. This line (or hyperplane) is called the regression line and is the line that will be used to predict continuous values.
- ε-insensitive Tube: SVR creates an ε-insensitive tube (or band) around the regression line. Any points that fall within this tube are considered acceptable predictions and do not contribute to the loss used in the model training. This tube helps to ignore errors as long as they are smaller than ε.
- Support Vectors: Points that are outside of this tube are used to adjust the regression line. These points are called support vectors because they are supporting (or defining) the margin of the tube.
- Regularization Parameter (C): SVR has a regularization parameter, C, which controls the trade-off between the flatness of the regression function and the amount up to which deviations larger than ε are tolerated.
- Kernel Trick: SVR uses the kernel trick to handle non-linear data. The kernel trick transforms the data into a higher-dimensional space where it is possible to fit a linear regression line. Common kernels include linear, polynomial, and radial basis function (RBF).

Evaluating regression model performance & their selection

R Squared, also known as the coefficient of determination, is a statistical measure used in regression analysis to assess the goodness of fit of a model.

How to Use It: After fitting a regression model, calculate R Squared to determine how well the model explains the variability of the response data around its mean.

When to Use It: It's typically used in the context of linear regression to evaluate the model's explanatory power.

What Does It Signify?: A higher R Squared value (close to 1) indicates that the model explains a large portion of the variance in the response variable. Conversely, a lower value indicates poor explanatory power.

R Square Intuition: It represents the proportion of variance in the dependent variable that can be explained by the independent variable(s) in the model. For example, an R Squared value of 0.8 suggests that 80% of the variance in the response variable can be explained by the model.

R Squared helps in understanding the effectiveness of the model in explaining the variation in the data, but it should be interpreted carefully, especially in cases of overfitting or when the model has a large number of predictors.

R square intuition

- $R^2 = 0$ (fitted model is equally good as average line)
- $R^2 = 1$ (SS_{res} =0 -> ideal scenario)
- $R^2 < 0$ (if your simple linear model fits the data worse than the average line)
- R^2--> goodness of fit (greater is better)

Problems with R^2 include its tendency to increase as more variables are added, regardless of their relevance, which can lead to overfitting.

The formula for R^2 is:

$$R^2 = 1 - \frac{\text{Sum of Squared Residuals (SSR)}}{\text{Total Sum of Squares (SST)}}$$

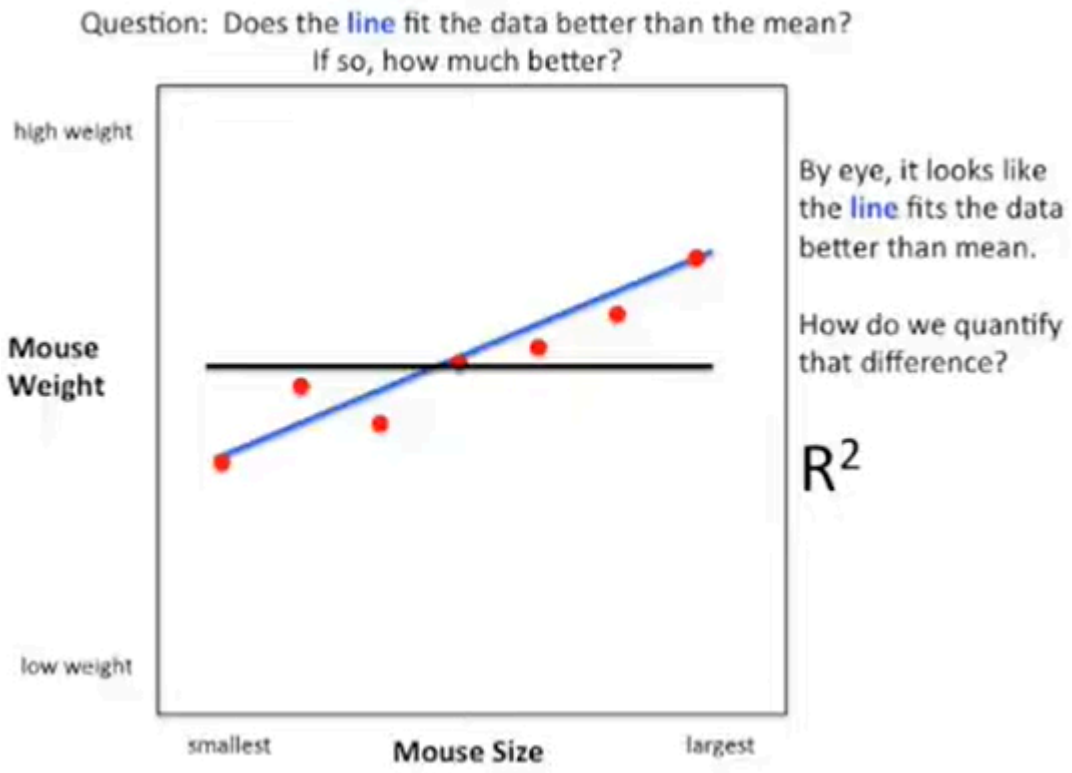
where SSR is the sum of squares due to error, and SST is the total sum of squares.

Adjusted R^2 accounts for the number of predictors in the model. Its formula is:

$$\text{Adjusted } R^2 = 1 - \frac{(1-R^2)(n-1)}{n-p-1}$$

where n is the number of observations and p is the number of predictors. Unlike R^2, Adjusted R^2 can decrease if irrelevant predictors are added, providing a more accurate measure of model performance.

Regression Model	Pros	Cons	When to Use It
Linear Regression	Works on any dataset size, relevant feature insight	Must meet linear assumptions	Ideal for simple, quick predictions with linearity
Polynomial Regression	Handles non-linear data well	Must select correct polynomial degree	Useful for non-linear relationships with more curve
SVR	Adapts well, not outlier-biased, handles non-linearity	Requires feature scaling, complex	Best for datasets with a clear margin of separation
Decision Tree Regression	Interpretable, no scaling needed, flexible	Prone to overfitting on small datasets	Good for complex datasets with clear rules
Random Forest Regression	Accurate, handles non-linearity well	Can overfit, needs careful selection of tree numbers	Great for ensemble methods, handling various types of data



$$R^2 = \frac{\text{Var(mean)} - \text{Var(line)}}{\text{Var(mean)}}$$

Var(**mean**) = 32

Var(**line**) = 6

$$R^2 = \frac{\text{Var(mean)} - \text{Var(line)}}{\text{Var(mean)}}$$

$$R^2 = \frac{32 - 6}{32}$$

$$R^2 = \frac{26}{32} = 0.81 = 81\%$$

There is 81% less variation around the line than the mean.

...or...

The size/weight relationship accounts for 81% of the variation.

Now, when someone says...

"The statistically significant R^2 was 0.9..."

You can think to yourself..

"Very good! The relationship between the two variables explains 90% of the variation in the data!"

And when someone else says...

"The statistically significant R^2 was 0.01..."

You can think to yourself...

"Dag! Who cares if that relationship is significant, it only accounts for 1% of the variation in the data.

Something else must explain the remaining 99%."

I like R^2 more than just plain old R because it is easier to interpret.

How much better is R = 0.7 than R = 0.5?

Well, if we convert those numbers to R^2, we see that:

$R^2 = 0.7^2 = 0.5$ 50% of the original variation is explained

$R^2 = 0.5^2 = 0.25$ 25% of the original variation is explained

With R^2, it is easy to see that the first correlation is twice as good as the second.

That said, R^2 does not indicate the direction of the correlation because squared numbers are never negative.

If the direction of the correlation isn't obvious, you can say, "the two variables were positively (or negatively) correlated with R^2 = ...

Decision Tree Classification

Working

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes
- **Step-4:** Generate the decision tree node, which contains the best attribute
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node

In **Decision Tree Splitting**, **the reduction in variance** is a technique used for continuous target variables (regression problems). This method splits the nodes on the decision tree by choosing the split that results in the largest decrease in variance of the target variable. The idea is to create child nodes that are more homogenous than the parent node, which means the values of the target variable in each node are close to each other. This helps the tree make more accurate predictions. The variance is calculated for each potential split, and the split that yields the lowest variance is chosen.

Entropy is a measure used in information theory to quantify the amount of uncertainty or randomness in data. In the context of a decision tree, it's used to measure the impurity or disorder in a group of examples.

Information Gain is a metric that measures the reduction in entropy after the dataset is split on an attribute. It's the difference between the entropy before the split and the weighted entropy after the split.

The relationship between both is that Information Gain is used to determine how much entropy is reduced by splitting on a particular attribute or feature. Decision tree algorithms use Information Gain to decide the optimal place to split the data to obtain the most homogeneous branches.

The **Gini Index, or Gini Impurity**, is a measure used in decision trees to determine the impurity of a node.

Properties of Gini Impurity:

- It ranges from 0 to 0.5, with 0 being pure (all elements belong to a single class) and 0.5 being maximum impurity.
- It is used because it can be computed faster than entropy, making it a good choice for large datasets.

Steps to Calculate Gini Impurity:

- For a set with multiple classes, count the occurrences of each class in the set.
- Calculate the proportion of each class, p_i , by dividing the count of a class by the total count of all classes in the set.
- Sum the squared proportion of each class.
- Subtract this sum from 1.

$$\text{Gini Impurity} = 1 - \sum(p_i)^2$$

This calculation is performed for each potential split in the dataset, and the one with the lowest Gini Impurity is chosen for the split.

Chi-Square in decision tree splitting is a statistical test used to measure the significance of the split. It compares the distribution of outcomes in the child nodes after a split to the distribution in the parent node. If the chi-square test yields a high value, it means the split made a significant impact.

Pruning a decision tree involves trimming off branches that have little to no value, to prevent overfitting and to get an optimal decision tree.

Advantages of Decision Trees:
- Easy to understand and interpret.
- Can handle both numerical and categorical data.
- Requires little data preprocessing.

Disadvantages of Decision Trees:
- Prone to overfitting.
- Can become unstable with slight variations in data.
- Biased with imbalanced datasets.

The choice of tree splitting technique in a decision tree often depends on the type of problem (classification or regression) and the nature of the data:

Gini Impurity: Typically used when the outcome is a classification problem. It's favored when you want a balance between speed and accuracy.

Entropy and Information Gain: Used for classification problems, particularly when you want to ensure the most informative features are used for splitting.

Chi-Square: It's used for classification tasks to determine if the difference between child and parent nodes is statistically significant.

Reduction in Variance: This is used for regression problems where the target variable is continuous. It chooses the split that results in the most homogenous child nodes.

The choice also depends on computational efficiency, the presence of outliers, and the distribution of the classes in the dataset.

The Decision Tree Classification algorithm involves a series of steps to split a dataset into smaller subsets while at the same time, an associated decision tree is incrementally developed. The main steps are as follows:

Select the Best Attribute: Use a measure like Gini impurity, Information Gain, or Gain Ratio to select the attribute that best splits the data into subsets.
Split Data: Based on the best attribute, split the data into subsets that contain possible values for this attribute.
Create Decision Node: The best attribute becomes a decision node, and branches are created for each possible value of the attribute.
Recursion on Subsets: Perform steps 1 to 3 recursively on each subset until one of the stopping conditions is met: all tuples belong to the same attribute value, there are no more attributes, or the subset is too small.

Leaf Node: Once a stopping condition is met, create a leaf node in the decision tree that predicts the outcome.

This process constructs a tree where decision nodes represent the data features, branches represent the decision rules, and each leaf node represents an outcome or class label.

Problem Statement:

Given a dataset with the following entries:

Income (in $)	Credit Score	Loan Approved?
5000	600	No
6000	700	Yes
7000	800	Yes
8000	900	Yes
4000	500	No

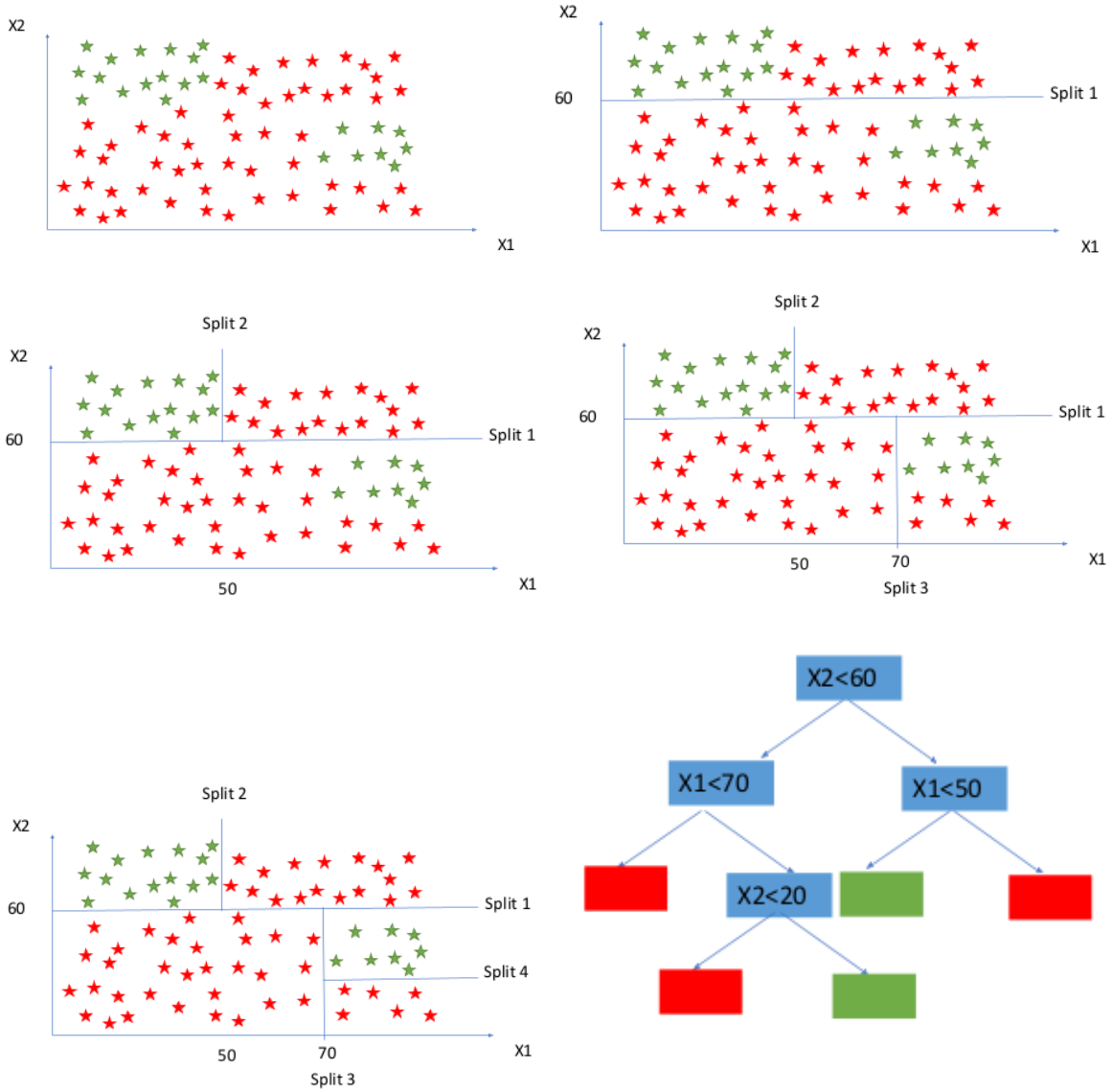
We need to build a decision tree to predict loan approval.

Steps to Solve:

1. **Select the Best Attribute:**
   - Calculate Information Gain for each attribute.
   - Suppose Credit Score has the highest Information Gain.
2. **Split Data on Credit Score:**
   - Split the data into subsets based on Credit Score values.
3. **Create Decision Nodes:**
   - Create a decision node for the Credit Score attribute.
4. **Recursion on Subsets:**
   - Apply steps 1-3 recursively if needed. In this simple example, further splits might not be necessary as the Credit Score alone could be decisive.
5. **Leaf Node:**
   - For Credit Score >= 700, Loan Approved = Yes.
   - For Credit Score < 700, Loan Approved = No.

Explanation:

- **Credit Score** is chosen as the splitting attribute because it provides the most significant information gain, indicating a strong correlation with loan approval outcomes.
- By splitting the dataset based on the Credit Score, we can clearly see a pattern that higher credit scores are associated with loan approval.
- The decision tree, in this case, would have a root node based on the Credit Score, with branches leading to "Yes" for scores >= 700 and "No" for scores < 700.



Attribute selection measure

In case of Continuous Target Variable

1. Reduction in Variance

In case of Categorical Target Variable

2. Gini Impurity
3. Information Gain
4. Chi-Square

Alright, let's break these down simply:

Reduction in Variance: This is used when your target variable (what you're trying to predict) is a number, like house prices. It's like trying to group similar numbers together and is calculated by seeing which split leads to the most similar numbers in each group.
Gini Impurity: This is for when your target is a category, like 'yes' or 'no'. Gini is like a measure of mix-up. Imagine you have a bag of red and blue balls; if it's all mixed up, Gini is high. You want to sort them so each bag has balls of mostly the same color, which means Gini is low.
Information Gain: This one also deals with categories. It's about surprise - how much less surprised you are about the outcome after the split. High information gain means the split gave you good insights and reduced your uncertainty a lot.
Chi-Square: It's a bit like detective work for categories, where you're looking at how much the observations differ from what you would expect randomly. If the difference is big, it means the attribute you're looking at is a good clue for predicting the outcome.

In simple terms, all these measures are ways to find the best questions that split your data into groups that are as pure as possible - meaning the items in each group are as similar to each other as they can be.

Decision Tree Classification.pptx

https://chat.openai.com/c/ec90613d-1cac-4361-bc1c-269f0b5b4839

WATCH A VIDEO ON DECISION TREE CLASSIFICATION

Bayesian Classifiers

Bayesian classifiers are a group of classifiers that rely on the probability theory's Bayes Theorem. They predict class membership probabilities, such as the probability that a given sample belongs to a particular class.

The Naïve Bayes Classifier works under the "naive" assumption that the attributes are conditionally independent given the class. Despite this simplification, Naïve Bayes Classifiers have worked quite well in many real-world situations, famously document classification and spam filtering.

The formula for Naïve Bayes is:

$$P(C_k|x) = \frac{P(C_k)\prod_{i=1}^{n} P(x_i|C_k)}{P(x)}$$

Where:

- $P(C_k|x)$ is the posterior probability of class C_k given predictor x.
- $P(C_k)$ is the prior probability of class C_k.
- $P(x_i|C_k)$ is the likelihood which is the probability of predictor x given class C_k.
- $P(x)$ is the prior probability of predictor x.

Algorithm for Naïve Bayes Classifier:

- Calculate the prior probability for given class labels
- Find the likelihood probability with each attribute for each class
- Put these value in Bayes Formula and calculate posterior probability.
- See which class has a higher probability, given the input belongs to the higher probability class.

Naïve Bayes is particularly advantageous due to its simplicity, efficiency, and effectiveness on large datasets. However, its assumption of feature independence rarely holds true in real-world scenarios, which can limit its performance.

Bayes Minimum Error Classification (BMEC) focuses on minimizing the probability of error in classification tasks by choosing the class with the highest posterior probability.

Bayes Minimum Error Classification, also known as Bayes Optimal Classifier, is a fundamental concept in the field of statistical pattern recognition and machine learning. Let's break it down into layman's terms and then delve into the formal definition, including an example for each.

In Layman's Terms

Imagine you're trying to guess whether it will rain today based on the clouds you see in the sky. You have a book that records every day's weather and the type of clouds that were observed. Over time, you notice patterns: certain clouds appear more often on rainy days, while others appear on dry days. You decide to use this book to make your daily guess more accurate.

Each morning, you look at the clouds and compare them with your book. If the clouds you see are more commonly associated with rain, you'll carry an umbrella, but if they're usually seen on dry days, you won't. You're trying to make the best guess to minimize your chance of getting wet or carrying an umbrella unnecessarily. This process of making the best guess based on historical patterns is similar to what Bayes Minimum Error Classification does, but with mathematical precision.

In Formal Terms

Bayes Minimum Error Classification is based on Bayes' Theorem, which is a way to update the probability of a hypothesis as more evidence or information becomes available. In the context of classification, the goal is to assign a new observation (data point) to the most likely class, given its features, to minimize the probability of misclassification. The "minimum error" part means we're trying to minimize the overall rate of wrong classifications.

Formally, given a set of classes $C_1, C_2, ..., C_n$ and an observation with features x, Bayes' classifier assigns x to the class C_k for which the posterior probability $P(C_k|x)$ is the highest. The posterior probability is calculated using Bayes' Theorem:

$$P(C_k|x) = \frac{P(x|C_k)P(C_k)}{P(x)}$$

where:

- $P(C_k|x)$ is the posterior probability of class C_k given observation x,
- $P(x|C_k)$ is the likelihood of observing x given it belongs to class C_k,
- $P(C_k)$ is the prior probability of class C_k,
- $P(x)$ is the evidence or the overall probability of observing x.

The denominator $P(x)$ is the same for all classes and can be ignored for the purpose of classification, simplifying the decision rule to selecting the class C_k that maximizes $P(x|C_k)P(C_k)$.

Example

Suppose we have two classes of fruits: apples and oranges. We want to classify a new fruit based on its sweetness (x) and texture (y). We have historical data indicating:

- 70% of the fruits are apples, and 30% are oranges (prior probabilities).
- Given an apple, there's a 90% chance it's sweet and has a smooth texture (likelihood).
- Given an orange, there's a 60% chance it's sweet but with a rough texture (likelihood).

When a new fruit is observed to be sweet with a smooth texture, we calculate the posterior probability for both classes and classify the fruit as an apple or orange based on which class has the higher posterior probability. This process minimizes the chance of misclassifying the new fruit, aiming for the minimum error.

In this way, Bayes Minimum Error Classification provides a powerful framework for making predictions or decisions based on probabilistic models of the world, leveraging prior knowledge and observed data to minimize mistakes.

Bayes Minimum Risk Classifier (BMRC) extends this concept by incorporating the costs or risks associated with making incorrect decisions. It minimizes the overall risk, not just the error rate.

Bayes Minimum Risk Classification is an extension of the Bayes Minimum Error approach that not only considers the likelihood of making a classification error but also incorporates the consequences (or costs) associated with those errors. Let's explore this concept in both layman's terms and formal terms, including examples for clarity.

In Layman's Terms

Let's use a simple analogy: imagine you're planning a picnic, and you need to decide whether to hold it outdoors or move it inside. The decision is risky because if it rains and you're outdoors, the picnic is ruined. If it doesn't rain and you're inside, you miss out on a beautiful day. However, the "cost" of moving indoors (missing out on a good day) is less than the cost of staying outdoors and having the picnic ruined by rain. You want to make a decision that minimizes your "regret" or risk of making the wrong choice, considering the different outcomes and their associated costs.

In this scenario, you'd use all the information available (weather forecasts, your own observations) to make the best decision. Bayes Minimum Risk Classification works similarly but applies mathematical models to calculate the decision that minimizes the overall risk, taking into account the probability of each outcome and the cost associated with being wrong.

In Formal Terms

Bayes Minimum Risk Classification formalizes the decision-making process by calculating the expected risk (or cost) of classifying an observation into each possible class and then choosing the classification that minimizes this expected risk.

The expected risk for classifying an observation x into class C_j is given by:

$$R(C_j|x) = \sum_i L(C_j, C_i)P(C_i|x)$$

where:

- $R(C_j|x)$ is the expected risk of classifying x as belonging to class C_j,
- $L(C_j, C_i)$ is the loss (or cost) associated with classifying an observation that truly belongs to class C_i as class C_j,
- $P(C_i|x)$ is the posterior probability of class C_i given observation x,
- The sum is taken over all possible true classes C_i.

The decision rule in Bayes Minimum Risk Classification is to choose the class C_j that minimizes $R(C_j|x)$ for each observation x.

Example

Imagine a medical diagnosis scenario where you're testing for a serious disease. There are two classes: positive (disease present) and negative (disease absent). The costs associated with each type of classification error are different:

- The cost of a false positive (diagnosing a healthy person as sick) might be the anxiety and unnecessary treatment costs.
- The cost of a false negative (failing to diagnose a sick person) could be much higher, potentially leading to serious health consequences or even death.

Given a patient's test results, you calculate the expected risk of classifying them as having the disease or not, incorporating the different costs of false positives and false negatives. Your goal is to make a decision that minimizes the total expected risk, which might mean erring on the side of caution (favoring false positives) because the cost of a false negative is much higher.

In this way, Bayes Minimum Risk Classification provides a more nuanced decision-making framework that accounts for the varying consequences of errors, enabling more informed and context-sensitive decisions.

The relationship between BMRC and BMEC lies in their objective: BMEC is a special case of BMRC when the cost of misclassification is the same for all classes. When specific risks or costs are associated with different types of misclassification, BMRC provides a more nuanced approach to decision-making.

Feature	Bayes Minimum Error Classification	Bayes Minimum Risk Classification
Objective	Minimize the overall error (misclassification) rate.	Minimize the expected cost (risk) of decisions.
Decision Criteria	Classify an observation into the class with the highest posterior probability.	Classify an observation into the class that minimizes the expected risk, considering both the probabilities of outcomes and their associated costs.
Consideration of Costs	Assumes equal cost for all types of misclassification errors.	Explicitly incorporates different costs associated with different types of classification errors.
Application Scenarios	Suitable for scenarios where all errors are considered equally serious.	Ideal for scenarios where the consequences of different types of errors vary significantly.
Use in Decision Making	Used when the goal is purely about being as accurate as possible without regard to the varying impacts of different types of errors.	Used when decisions have varying consequences, and there is a need to weigh the benefits and drawbacks of potential errors.
Example	Deciding whether an email is spam based solely on its content's characteristics.	Medical diagnosis where the cost of false negatives (missing a disease) and false positives (unnecessary worry or treatment) are significantly different.

Applications of Bayesian classifiers, including the Bayes Minimum Error and Risk Classifiers, span various fields:

Medical Diagnosis: Used to evaluate the risk of a patient having a particular disease based on observed symptoms.
Spam Filtering: Classifying emails as spam or not spam.
Financial Analysis: For credit scoring and evaluating the risk of loans.

Pros:

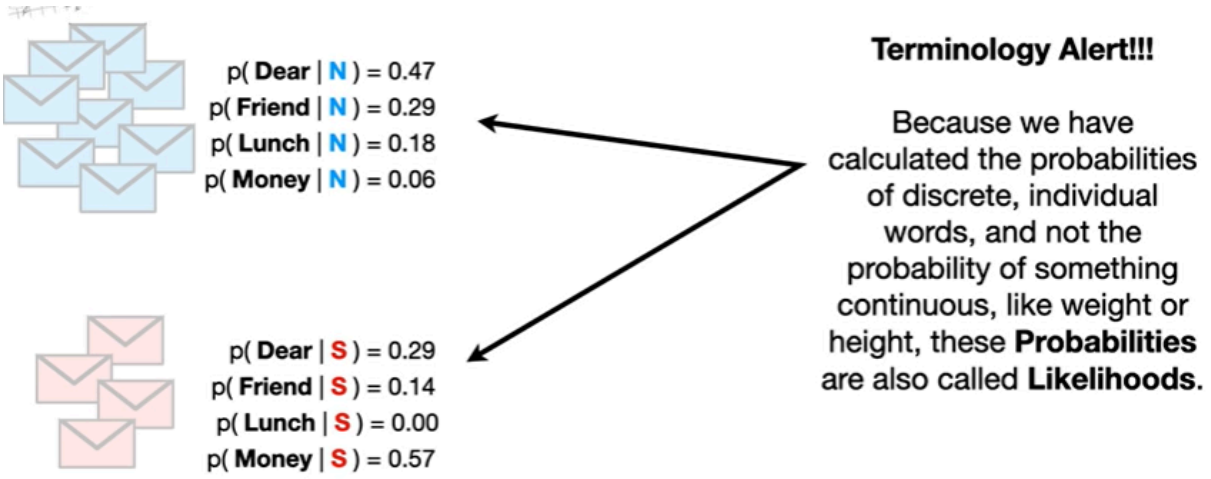
- Efficient with small datasets.
- Handles missing data well.

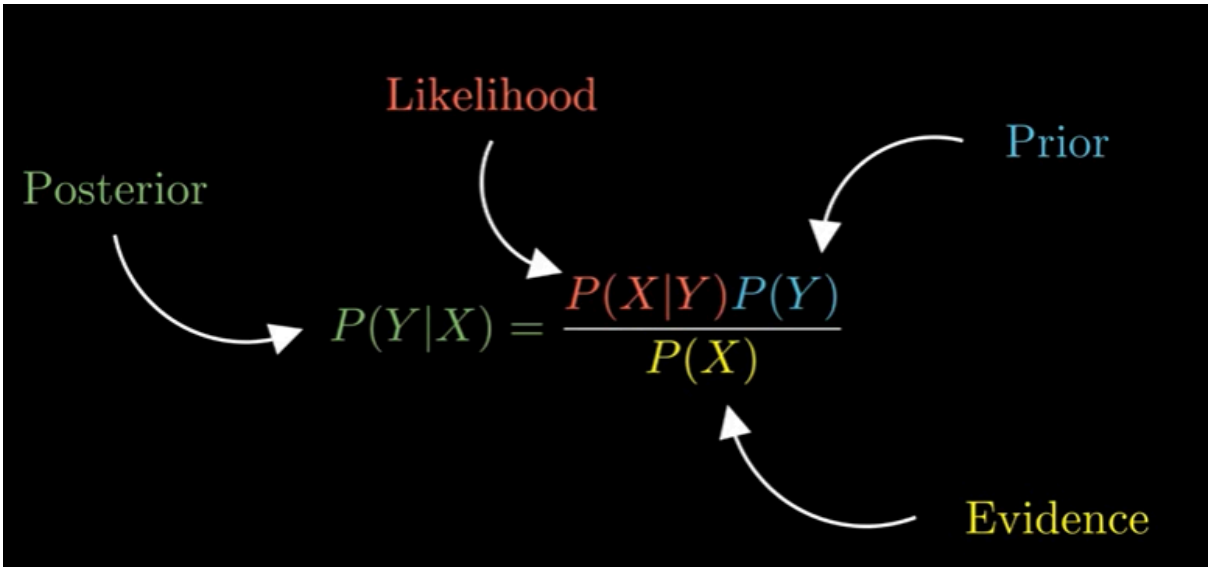
- Provides probability scores for predictions.

Cons:

- Assumes feature independence, which might not hold true.
- Sensitive to biased datasets.
- Performance can degrade with many irrelevant features.

These classifiers are invaluable tools in fields where probabilistic reasoning is key, despite their limitations.





Box Plots: A box plot is a standardized way of displaying the distribution of data based on a five-number summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum. It can reveal outliers, the spread of the data, and its skewness.

Histogram: A histogram is a graphical representation of the distribution of numerical data. It groups data into bins and displays the frequency of data points in each bin, offering a sense of the density of the data.

Difference Between Histogram and Box Plot:
- A histogram shows the distribution of a dataset and helps in understanding its central tendency and variability.
- A box plot summarizes data numerically and visually, highlighting the median, quartiles, and outliers, but does not depict data distribution in detail.

Each has its use, with histograms being more informative about the data distribution and box plots providing a quick summary of the data.

A **Scatter Plot** is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data. It shows how much one variable is affected by another and helps in understanding the correlation between them.

A **Contingency Table**, also known as a cross-tabulation or crosstab, is a table showing the frequency distribution of the variables. It helps in analyzing the relationship between two or more categorical variables by showing the number of observations for each combination of variables.

Handling outliers in a dataset is crucial as they can significantly impact the results of a data analysis. Various methods to handle outliers include:

Removing Outliers: Simply exclude them from the data. This is effective for clear, erroneous data points but can introduce bias if not done carefully.

Transforming Data: Use transformations like logarithms to reduce the effect of outliers.

Imputing: Replace outliers with values estimated from other data, like the mean or median.

Using Robust Statistical Measures: Employ techniques like median, interquartile range, or robust regression that are less sensitive to outliers.

Binning: Group data into bins which can minimize the effect of minor outliers.

The approach chosen depends on the context, the nature of the data, and the specific analysis or modeling task at hand.

High-dimensional data, commonly found in fields like genomics and image processing, presents several challenges, collectively known as the **"Curse of Dimensionality."** These challenges include:

- **Sparsity of Data:** In high dimensions, data becomes sparse, making it difficult to find patterns or make reliable statistical inferences.
- **Increased Computational Complexity:** More dimensions can significantly increase the computational burden.
- **Risk of Overfitting:** High-dimensional models can easily overfit, capturing noise instead of the underlying data pattern.
- **Distance Metrics Lose Meaning:** In high dimensions, traditional distance metrics used in algorithms like KNN become less effective, as all points tend to be equidistant from each other.
- **Feature Selection Complexity:** Identifying the most relevant features becomes more challenging as the dimensionality increases.

Handling these issues often involves techniques like dimensionality reduction, feature selection, or specialized algorithms designed for high-dimensional spaces.

Handling high-dimensional data effectively involves various strategies:

Dimensionality Reduction: Techniques like Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) reduce the number of variables under consideration.

Feature Selection: Identifying and keeping only the most relevant features can reduce dimensionality without much loss of information.

Regularization: Techniques like Lasso or Ridge regression penalize the number of features in the model.

Sampling: Using representative samples from the dataset can reduce complexity.

Ensemble Methods: Techniques like Random Forests can handle high-dimensional data by building multiple models and averaging their results.

Using Algorithms Designed for High-Dimensional Data: Some algorithms perform better with high-dimensional data.

These methods can help mitigate the issues of overfitting, computational complexity, and data sparsity that come with high-dimensional data.

5Naive Bayes.pptx

HANDLING OUTLIERS

- **Remove outliers:** If the number of records which are outliers is not many, a simple approach may be to remove them.
- **Imputation:**
  - Imputation is a method to assign a value to the data elements having outliers.
  - One other way is to impute the value with mean or median or mode.
  - The value of the most similar data element may also be used for imputation.
- **Capping:** For values that lie outside the 1.5|×| IQR limits, we can cap them by replacing those observations below the lower limit with the value of 5th percentile and those that lie above the upper limit, with the value of 95th percentile.

HANDLING MISSING VALUES

- Eliminate records having a missing value of data elements
- Imputing missing values
  - assign a value to the data elements having missing values.
  - Mean/mode/median is most frequently assigned value.
    - For quantitative attributes
      - all missing values are imputed with the mean, median, or mode of the remaining values under the same attribute.
    - For qualitative attributes
      - all missing values are imputed by the mode of all remaining values of the same attribute.
  - However, another strategy may be identify the similar types of observations whose values are known and use the mean/median/mode of those known values.
- *Estimate missing values*

- *Estimate missing values*
  - If there are data points similar to the ones with missing attribute values, then the attribute values from those similar data points can be planted in place of the missing value.
  - For finding similar data points or observations, distance function can be used.
  - For example, let's assume that the weight of a Russian student having age 12 years and height 5 ft. is missing.
    - Then the weight of any other Russian student having age close to 12 years and height close to 5 ft. can be assigned.

Box plots, histograms, and scatter plots are graphical representations used in statistics to visualize data, but they serve different purposes and provide different insights into the data's structure.

Box Plots

- Purpose: To show the distribution of a dataset and its central tendency, variability, and outliers.
- Components: A box plot displays the median of the data as a line inside a box, which represents the interquartile range (IQR, the middle 50% of the data). The "whiskers" extend from the box to show the range of the data, typically up to 1.5 * IQR from the quartiles, and points outside this range are plotted as outliers.
- Useful For: Identifying outliers, comparing distributions across groups, and understanding the spread and skewness of the data.

Histograms

- Purpose: To show the frequency distribution of a dataset.
- Components: A histogram is a type of bar chart that represents the distribution of numerical data by dividing the data into "bins" or intervals and counting the number of observations in each bin. The height of each bar shows the frequency of data points within that range.
- Useful For: Understanding the shape (e.g., bell-shaped, skewed, uniform) of the distribution, central tendency, and variability.

Scatter Plots

- Purpose: To show the relationship between two (or more, with extensions) quantitative variables.
- Components: A scatter plot displays points in a two-dimensional graph, with one variable represented along the x-axis and the other along the y-axis. Each point represents an observation.
- Useful For: Identifying trends, patterns, and correlations between variables, as well as potential outliers.

Summary of Differences

- Data Representation: Box plots summarize a single variable's distribution with five-number summary (minimum, first quartile, median, third quartile, maximum) and outliers. Histograms show the frequency distribution of a single variable. Scatter plots illustrate the relationship between two variables.
- Usage Context: Use box plots for comparing distributions and identifying outliers, histograms for visualizing the shape and spread of a single variable's distribution, and scatter plots for exploring relationships and correlations between variables.
- Visual Elements: Box plots use boxes and lines, histograms use bars, and scatter plots use points.

Each of these plots has its unique strengths and is chosen based on the specific aspects of the data one wishes to explore or present.

Linear Classifier

2 class problem

A Linear Classifier in a 2-class problem is a machine learning model used for binary classification. It works by finding a linear boundary that separates two classes. The algorithm assigns a new data point to one of the two classes based on which side of the boundary it falls. In mathematical terms, this involves finding weights and a bias such that a linear combination of the input features separates the classes as well as possible. This method is effective for problems where the two classes can be separated by a linear boundary. Common examples include Logistic Regression and Support Vector Machines with a linear kernel.

1. **What is a Linear Classifier?**
   A Linear Classifier is a type of classification model that makes predictions based on a linear predictor function. It uses a combination of weights and feature values.
2. **How is the Linear Classifier Represented Mathematically?**
   Mathematically, it's represented as $Y = aX + b$, where Y is the output, X is the input feature vector, and a and b are the model parameters.
3. **What is the Role of the Feature Vector?**
   The feature vector X represents the input data, and each element in X corresponds to a feature.
4. **How are Predictions Made in a Linear Classifier?**
   Predictions are made based on the sign of the function atY. If $atY > 0$, the instance is classified into one class, and if $atY < 0$, it's classified into another.
5. **What is the Error Term in Linear Classification?**
   The error term is computed as $-atY$ for misclassified instances.
6. **How is the Total Error Calculated?**
   The total error J is the sum of $-atY$ for all misclassified instances.

7. **What is the Objective of the Linear Classifier?**
   The objective is to minimize the total error J.
8. **What is Gradient Descent and How is it Used in Linear Classification?**
   Gradient Descent is an optimization algorithm used to minimize the error by updating the parameters a iteratively.
9. **How is the Feature Vector Modified During Training?**
   During training, the feature vector may be negated to ensure correct classification.
10. **What is the Process of Iteratively Updating the Solution Vector?**
    The solution vector a is updated iteratively by adding $\cap \sum y$ to a, where $\cap$ is a learning rate and y is the sum of misclassified vectors.

A linear classifier is a method used in machine learning and statistics to separate two classes of objects based on a linear combination of input features. The goal of a linear classifier is to find a decision boundary that can classify all input samples into one of two classes. The decision boundary in a 2-class problem is typically a line, plane, or hyperplane, depending on the dimensionality of the input space. Mathematically, a linear classifier can be understood through the following components:

1. Feature Vector

Let $X = [x_1, x_2, ..., x_n]$ represent the feature vector of an input sample, where n is the number of features. Each x_i corresponds to a particular characteristic or attribute of the sample.

2. Weight Vector

Correspondingly, there is a weight vector $W = [w_1, w_2, ..., w_n]$ and a bias term b. Each weight w_i indicates the importance of the corresponding feature x_i in determining the classification.

3. Linear Combination

The classifier makes its prediction by calculating a linear combination of the input features and their corresponding weights, plus a bias term. This can be represented as:

$$f(X) = W \cdot X + b = \sum_{i=1}^{n} w_i x_i + b$$

4. Decision Rule

To classify the input sample, the model applies a decision rule to the result of the linear combination. The most common decision rule is to compare $f(X)$ to a threshold, typically 0. If $f(X) > 0$, the sample is classified into one class (e.g., class 1), and if $f(X) \leq 0$, it is classified into the other class (e.g., class 2).

Continuing from the mathematical representation, once the optimal weights W and bias b are found, the linear classifier can make predictions on new, unseen data. The process of training a linear classifier and making predictions can be further detailed as follows:

Training the Classifier

1. **Initialization**: Begin with an initial guess for the weights W and bias b. This could be random values or zeros.
2. **Optimization**: Use an optimization algorithm (such as gradient descent) to iteratively adjust W and b to minimize the loss function. The choice of loss function depends on the specific type of linear classifier:
   - For **SVMs**, the objective might include maximizing the margin between classes while penalizing points that fall on the wrong side of the margin.
   - For **logistic regression**, the logistic loss function is used, which models the probability that a given sample belongs to a particular class.

3. **Update Rule:** In each iteration, W and b are updated based on the gradient of the loss function with respect to W and b. For example, in gradient descent, the update rule might look like:
   $W := W - \alpha \nabla_W L(W, b)$
   $b := b - \alpha \nabla_b L(W, b)$
   where α is the learning rate, and $\nabla_W L(W, b)$ and $\nabla_b L(W, b)$ are the gradients of the loss function with respect to W and b, respectively.

Making Predictions

Once the model is trained, making predictions on new data involves calculating the linear combination of the input features and the learned weights, adding the bias, and applying the decision rule:

- If $f(X) = W \cdot X + b > 0$, predict class 1.
- If $f(X) = W \cdot X + b \leq 0$, predict class 2.

This decision rule can also be expressed in terms of probability, especially in the context of logistic regression, where the sigmoid function is applied to $f(X)$ to estimate the probability that X belongs to class 1.

Mathematical Challenges

- **Linear Separability:** Linear classifiers work best when the two classes can be separated by a linear boundary. If the data is not linearly separable, linear classifiers may not perform well.
- **Feature Space Transformation:** Sometimes, the original input features are transformed or augmented (e.g., by polynomial expansion) to make the data linearly separable in the transformed feature space.
- **Regularization:** To prevent overfitting, especially in high-dimensional spaces, regularization techniques (e.g., L1, L2 regularization) are applied. This involves adding a penalty term to the loss function that constrains the magnitude of the weights.

Example: Logistic Regression

In logistic regression, the decision function $f(X)$ is passed through a logistic (sigmoid) function $\sigma(f(X))$ to predict the probability that X belongs to class 1:
$P(y=1|X) = \sigma(f(X)) = \frac{1}{1+e^{-(W \cdot X + b)}}$

The model is trained to maximize the likelihood of the observed data, which is equivalent to minimizing the negative log-likelihood (or cross-entropy loss) with respect to the parameters W and b.

Summary

Linear classifiers provide a powerful and interpretable method for binary classification by finding a hyperplane that separates the two classes in the feature space. The specific characteristics of the classifier, such as the decision boundary, loss function, and optimization method, depend on the type of linear model being used (e.g., SVM, logistic regression). The success of a linear classifier largely depends on the nature of the data and the appropriateness of the model assumptions to the task at hand.

K-Nearest Neighbor

The K-Nearest Neighbors (KNN) Classifier is a simple yet effective algorithm used in machine learning for classification tasks.

Algorithm:

- Choose the number of neighbors (K).
- Calculate the distance between the new data point and all other points in the training set.
- Sort these distances and take the K shortest.
- The class of the new data point is determined by the majority class among these K neighbors.

Pros:

- Simple and easy to implement.
- No assumption about data distribution.
- Effective if the training set is large.

Cons:

- Computationally expensive, especially with large datasets.
- Sensitive to irrelevant features and the scale of the data.
- Performance degrades with high-dimensional data (curse of dimensionality).

The k-Nearest Neighbors (kNN) algorithm is a simple, yet powerful machine learning technique used for classification (and regression). The basic idea behind kNN is to classify a new data point based on the majority class among its k nearest neighbors. Here's a step-by-step algorithm for kNN classification:

Step 1: Choose the Number of Neighbors (k)

1. **Select `k`:** The first step is to choose the number of `k` neighbors. The choice of `k` affects the classification accuracy, and there's no one-size-fits-all answer. A small value of `k` can make the algorithm sensitive to noise, while a large `k` makes it computationally expensive and may include points from other classes.

Step 2: Calculate Distance

1. **Calculate Distance:** For a new data point, calculate the distance between this point and all other points in the dataset. The distance can be Euclidean, Manhattan, Minkowski, or any other distance metric. The Euclidean distance for two points `x` and `y` in an `n`-dimensional space is commonly used:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Step 3: Find Nearest Neighbors

1. **Identify Nearest Neighbors:** Sort the distances in ascending order and select the top `k` points from the sorted list. These points are the `k` nearest neighbors to the new data point.

Step 4: Vote for Classes (continued)

For classification tasks, after identifying the `k` nearest neighbors, we determine the class of the new data point based on these neighbors. Here are a few strategies to refine this step:

- **Simple Majority Vote**: The most straightforward approach is a simple majority vote: the class with the majority among the `k` neighbors is assigned to the new data point. If `k = 5` and three neighbors are of class A and two are of class B, the new data point is assigned to class A.
- **Weighted Voting**: Instead of giving each neighbor an equal vote, you can assign weights based on their distance to the new data point. Closer neighbors exert more influence on the classification. For example, the weight can be inversely proportional to the distance. This approach can help reduce the impact of outliers or irrelevant points.

- **Handling Ties**: Ties can occur, especially with even values of `k` or when multiple classes have the same number of neighbors. Several strategies can resolve ties:
  - **Decrease `k`**: Reduce `k` by 1 and recalculate until the tie breaks.
  - **Random Choice**: Randomly select among the tied classes.
  - **Nearest Neighbor Rule**: Choose the class of the closest neighbor among the tied classes.
  - **Weighted Decision**: In the case of weighted voting, a tie is less likely but can still occur. You might consider the sum of weights for the tied classes and choose the one with the higher sum.

Step 5: Assign Class to the New Point

Once the voting is complete and any ties are resolved, the class with the highest vote among the `k` nearest neighbors is assigned to the new data point. This step finalizes the classification process for the new point.

Beyond Basic kNN

- Feature Scaling: kNN is sensitive to the scale of the data features because distance calculations are affected by the magnitude of the features. Normalizing or standardizing data can be crucial for good performance.

- Choosing the Right Distance Metric: While Euclidean distance is common, the choice of distance metric (Manhattan, Minkowski, Hamming for categorical data, etc.) should be based on the nature of the data and the problem context.
- Optimizing kNN: Large datasets can make kNN computationally expensive due to the need to calculate distances to all training samples. Techniques like kd-trees, ball trees, or approximate nearest neighbor methods can help speed up the search for the nearest neighbors.
- Cross-Validation for Hyperparameter Tuning: Use cross-validation to find the optimal `k` and other parameters (like the distance metric). This process involves dividing the dataset into training and validation sets, testing different parameter values, and selecting the ones that achieve the best validation performance.

kNN's simplicity and ease of understanding make it a useful algorithm for classification tasks, especially as a baseline model. However, its performance and efficiency can significantly depend on the data's nature and how well the hyperparameters are tuned.

Support Vector Machine

Binary Classifier

A Support Vector Machine (SVM) is a powerful supervised machine learning model used for binary classification. It works by finding a hyperplane that best separates two classes in the feature space. SVM aims to maximize the margin between the data points of the different classes, making the classifier robust.

For binary classification, SVM identifies the optimal separating hyperplane which maximizes the margin between two classes. Data points that lie closest to the hyperplane, called support vectors, are crucial in defining the boundary.

SVM is effective in high-dimensional spaces and works well when there is a clear margin of separation. However, it requires careful tuning of parameters and is not suitable for large datasets due to its computational complexity. Additionally, SVMs perform poorly when the data has a lot of noise and overlapping classes.

Support Vector Machine (SVM) is a powerful and versatile machine learning algorithm used for classification and regression tasks. At its core, SVM seeks to find the best hyperplane that separates the data points of different classes in the feature space. Here is a simplified algorithm for SVM classification focusing on the linear case. For non-linear separations, kernel tricks are applied, which allow the algorithm to work in higher-dimensional spaces without explicitly computing the coordinates in that space.

Algorithm for Linear SVM Classification

1. **Input Data:** Let X be the input features with dimension $n \times m$, where n is the number of samples and m is the number of features. Let y be the target vector containing the class labels, where $y_i \in \{-1, 1\}$ for binary classification.
2. **Choose a Kernel:** For linear SVM, the kernel is simply the dot product of two vectors. For non-linear separations, common kernels include polynomial, radial basis function (RBF), and sigmoid.
   - Linear Kernel: $K(x_i, x_j) = x_i^T x_j$
3. **Objective Function:** The SVM algorithm aims to maximize the margin between the two classes. The optimization problem can be formulated as minimizing the following objective function subject to certain constraints:
   $\min_{w,b} \frac{1}{2}\|w\|^2$
   Subject to $y_i(w^T x_i + b) \geq 1$ for $i = 1, ..., n$, where w is the weight vector, and b is the bias.
4. **Lagrange Multipliers:** To solve the optimization problem, introduce Lagrange multipliers $\alpha_i \geq 0$ for each constraint. The problem then becomes finding the saddle point of the Lagrangian:
   $L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w^T x_i + b) - 1]$
5. **Optimization:** Solve the dual problem by maximizing $L(w, b, \alpha)$ with respect to α while minimizing with respect to w and b, under the constraint that $\alpha_i \geq 0$.
6. **Compute the Weight Vector and Bias:** From the solution of the dual problem, compute the optimal weight vector w as a linear combination of the support vectors:
   $w = \sum_{i=1}^{n} \alpha_i y_i x_i$
   The bias b can be computed using the support vectors by ensuring that the distance from them to the hyperplane is exactly 1.
7. **Decision Function:** The decision function that classifies new samples x is given by:
   $f(x) = \text{sgn}(w^T x + b)$
   A positive value of $f(x)$ classifies x as belonging to the positive class, and a negative value classifies it as belonging to the negative class.

8. **Support Vectors:** The data points for which $\alpha_i > 0$ are called support vectors. These are the points that lie closest to the decision boundary (hyperplane) and are critical in defining the position and orientation of the hyperplane.

This algorithm focuses on the essence of SVM for linear classification. In practice, solving the optimization problem often involves using specialized libraries like LIBSVM or tools available in machine learning libraries such as scikit-learn, which provide efficient implementations for both linear and non-linear SVM classification.

K - Means Clustering

Clustering in Machine Learning (ML) refers to the process of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups. It is a type of unsupervised learning method because it doesn't rely on labeled input data to learn. Instead, it identifies patterns directly from the data itself. The goal of clustering is to discover inherent groupings in data.

Key Concepts in Clustering:

- Clusters: Collections of data points aggregated together because of certain similarities.
- Centroids: The center point of a cluster. In some algorithms, such as K-means, each cluster is represented by a centroid.
- Distance Measures: Metrics used to determine the similarity or dissimilarity between data points. Common measures include Euclidean distance, Manhattan distance, and cosine similarity.

Applications of Clustering:

- Market Segmentation: Identifying distinct groups in customer data to tailor marketing strategies.
- Social Network Analysis: Detecting communities within large networks of social interactions.
- Image Segmentation: Grouping together pixels in an image based on similarity to segment the image into meaningful parts.
- Anomaly Detection: Isolating unusual data points that do not fit into any cluster could indicate fraudulent activity or outliers.
- Bioinformatics: Classifying genes with similar expression patterns or grouping proteins with similar functions.

Challenges in Clustering:

- Choosing the Right Number of Clusters: For many algorithms, such as K-means, the number of clusters needs to be specified beforehand, which is not always straightforward.
- Scalability: Some clustering algorithms do not scale well with large datasets.
- High Dimensionality: Data with a large number of features can make clustering more challenging due to the curse of dimensionality.
- Noise and Outlier Sensitivity: Some algorithms are sensitive to outliers, which can significantly affect the clustering outcome.

The K-means clustering algorithm is one of the simplest and most commonly used clustering methods in machine learning. It aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

Algorithm Overview:

Initialization: Start by selecting k initial centroids, where k is a user-specified number of clusters. The initial centroids can be selected randomly from the data points or using more sophisticated methods to facilitate convergence.
Assignment Step: Assign each data point to the nearest centroid. The "nearest" is usually defined using Euclidean distance, although other distance measures can also be used. After this step, each point is assigned to exactly one cluster.
Update Step: Calculate the new centroids as the mean of all points assigned to each cluster. The centroid is updated to be the center (mean) of all data points that are assigned to the cluster.
Repeat: Repeat the Assignment and Update steps until the centroids no longer change significantly, indicating that the algorithm has converged, or until a specified number of iterations is reached.

The Algorithm Steps in Detail:

Step 1: Initialize Centroids

- Select k distinct data points randomly as the initial centroids. Alternatively, use methods like K-means++ for smarter initialization to improve convergence.

Step 2: Assign Points to the Nearest Centroid

- For each point in the dataset, calculate the distance to every centroid and assign the point to the closest centroid's cluster.

Step 3: Recalculate Centroids

- Update each centroid to the mean (center) of all points assigned to its cluster.

Step 4: Convergence Check

- If the centroids do not change or the change is below a certain threshold, or a predetermined number of iterations is reached, the algorithm stops. Otherwise, it returns to Step 2.

Considerations:

- Choosing k: The number of clusters k is a user-defined parameter and choosing the right value is crucial for getting meaningful clusters. Methods like the Elbow Method, the Silhouette Method, or the Gap Statistic can help determine a suitable k.
- Convergence: K-means will always converge but may converge to a local minimum. The quality of the final solution depends on the initial set of centroids. Therefore, it's common practice to run K-means multiple times with different initial configurations and choose the best result.
- Distance Measure: While Euclidean distance is standard, different contexts may require different metrics. The choice of distance measure can significantly affect the results.
- Scalability and Efficiency: For very large datasets, the standard K-means algorithm can be computationally expensive. Variants like Mini-Batch K-means can be more efficient.

K-means is widely used for its simplicity and speed, but its effectiveness heavily depends on the shape and scale of the data distribution, the quality of the initial centroids, and the chosen value of k

Distance Metrics in Machine Learning

To deal with continuous or numerical variables

1. Euclidean Distance
2. Manhattan Distance
3. Minkowski Distance

To deal with categorical variables

4. Hamming Distance

Formula for Euclidean Distance

$$d = ((p_1 - q_1)^2 + (p_2 - q_2)^2)^{1/2}$$

$$D_e = \left(\sum_{i=1}^{n} (p_i - q_i)^2\right)^{1/2}$$

Where,
- n = number of dimensions
- pi, qi = data points

Manhattan Distance

- is the sum of absolute differences between points across all the dimensions

$$d = |p_1 - q_1| + |p_2 - q_2|$$

$$D_m = \sum_{i=1}^{n} |p_i - q_i|$$

Where,
- n = number of dimensions
- pi, qi = data points

Minkowski Distance

- Minkowski Distance is the generalized form of Euclidean and Manhattan Distance.

$$D = \left(\sum_{i=1}^{n} |p_i - q_i|^p\right)^{1/p}$$

Hamming Distance

- measures the similarity between two strings of the same length
- is the number of positions at which the corresponding characters are different.

NUMERICAL IN PPT

P 1K-means Clustering.pptx

K-Means ++

K-means++ is an algorithm for choosing the initial values (or "seeds") for the K-means clustering algorithm. The goal of K-means clustering is to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. However, the standard K-means algorithm is highly sensitive to the initial randomly selected cluster centers

and can result in poor convergence and suboptimal solutions. K-means++ addresses this issue by using a smarter initialization technique that tends to select seeds that are far apart from each other, leading to better clustering results and faster convergence.

The K-means++ Initialization Algorithm

1. Choose the first cluster center randomly from the data points.
2. For each data point, calculate the distance $D(x)$ from the point to its nearest already chosen cluster center.
3. Select the next cluster center from the data points with a probability proportional to $D(x)$^2. This step biases the selection towards points that are far away from the existing cluster centers.
4. Repeat Steps 2 and 3 until k cluster centers have been chosen.
5. Proceed with standard K-means clustering using the k seeds selected by the above process.

Advantages of K-means++

- Improved Initialization: By spreading out the initial cluster centers, K-means++ reduces the chance of poor clusterings and improves the quality of the final solution.
- Faster Convergence: The algorithm often requires fewer iterations to converge to a solution, since the initial seeds are chosen more wisely.
- Easy to Implement: The modifications to the K-means algorithm are straightforward and can be easily added to any existing K-means implementation.

Performance and Complexity

K-means++ improves the initialization step of K-means, which can significantly enhance the algorithm's overall performance in terms of both the quality of the clustering results and the time to convergence. However, the initialization process itself is more computationally intensive than random initialization, as it involves computing distances from each point to existing cluster centers multiple times. Despite this, the benefits of using K-means++ are generally considered to outweigh the additional computational costs, especially for applications where the quality of clustering is critical.

In practice, K-means++ is widely used and is often the default choice for initializing K-means clustering in many machine learning libraries due to its balance of efficiency and effectiveness.

2K-means++.pptx
Numerical Above

Hierarchical Clustering

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. It is a popular method for statistical data analysis used in various fields such as machine learning, pattern recognition, image analysis, bioinformatics, and others. The goal of hierarchical clustering is to group similar objects into clusters, where objects in the same cluster are more similar to each other than to those in other clusters.

There are two main types of hierarchical clustering:

Agglomerative (Bottom-up approach): This is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. It starts with each object in its own cluster. Then, it merges these atomic clusters into larger and larger clusters, until all objects are in a single cluster or until a desired structure is achieved. The process can be visualized using a dendrogram, which is a tree-like diagram that records the sequences of merges or splits.
Divisive (Top-down approach): This approach starts with all the objects in a single cluster. Then, this cluster is divided into smaller clusters, which in turn are recursively split into even smaller clusters until each object ends up in its own cluster, or a desired level of clustering is achieved.

The choice of similarity measure is a critical step in hierarchical clustering and significantly affects the outcomes. Common measures include the Euclidean distance, Manhattan distance, maximum distance, and the Mahalanobis distance. Additionally, when performing agglomerative clustering, the method of calculating the distance between clusters (linkage criteria) needs to be chosen. The most common linkage criteria include:

- Single linkage: distance between the closest members of two clusters.
- Complete linkage: distance between the farthest members of two clusters.
- Average linkage: average distance between all members of two clusters.
- Centroid linkage: distance between the centroids of two clusters.

- Ward's method: minimizes the variance within the clusters.

Hierarchical clustering does not require a pre-specified number of clusters, which is a significant advantage over k-means clustering. However, it is computationally more expensive and not suitable for large datasets. The results of hierarchical clustering are not deterministic; different methods of measuring distance and linkage can result in different hierarchies being produced.

Agglomerative HC

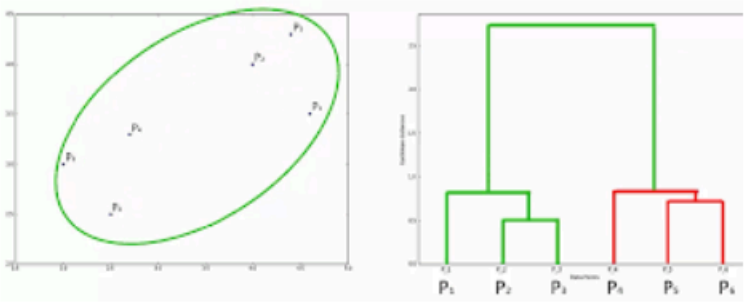
- Step 1: Make each data point a single-point cluster □ that forms N clusters
- Step 2: Take the two closest data points and make them one cluster □ that forms N-1 clusters
- Step 3: Take the two closest clusters and make them one cluster □ that forms N-2 clusters
- Step 4: Repeat Step 3 until there is only one cluster

Dendrogram

A Dendrogram is a tree-like diagram that records the sequences of merges or splits.

How do dendrograms work?

- 6 points on the left chart.
- Points are having dissimilarity
- We can measure this dissimilarity by finding distance between them.
- Using these we will create dendrograms.
- Initially every single point is an individual cluster.



Dendrograms and silhouette analysis are two methods used to interpret and evaluate the results of hierarchical clustering, each serving distinct purposes in the analysis of cluster structures.

Dendrograms

A dendrogram is a tree-like diagram that displays the arrangements of the clusters produced by hierarchical clustering. It illustrates how each cluster is composed by drawing a U-shaped link between a non-singleton cluster and its children. The height of the U-link indicates the distance (dissimilarity) at which the clusters merged, suggesting the level of similarity between clusters. Dendrograms are particularly useful for:

- Visualizing the clustering process: They show how individual elements are merged into clusters and how these clusters are merged with each other.
- Determining the number of clusters: By examining the heights of the merges, one can decide on a cutoff distance that determines the number of clusters to be used for further analysis. A large jump in distance might suggest a natural division within the data.

Silhouette Analysis

Silhouette analysis measures how close each point in one cluster is to points in the neighboring clusters. This metric provides a succinct graphical representation of how well each object has been classified. The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette value ranges from -1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. Silhouette analysis is useful for:

- Assessing the clustering quality: It provides insight into the distance between the resulting clusters. Clusters with a high average silhouette width are considered to be more distinct.
- Determining the optimal number of clusters: By comparing the average silhouette scores for different numbers of clusters, one can determine the best number of clusters to use.

Both dendrograms and silhouette analysis are valuable tools in the exploratory data analysis phase, helping to uncover the inherent structure within the data and to validate the consistency of the clusters formed. They complement each other well: dendrograms provide a visual representation of the cluster formation process, while silhouette analysis offers a quantitative measure to assess the quality of the clustering. Together, they can guide the decision-making process regarding the number of clusters to choose and provide insights into the dataset's clustering structure.

Aspect	Divisive Clustering	Agglomerative Clustering
Starting Point	Begins with all data points in one cluster.	Begins with each data point in its own cluster.
Process	Divides clusters iteratively until singletons.	Merges clusters iteratively until one cluster.
Approach	Top-down.	Bottom-up.
Cluster Formation	Begins with one large cluster and splits.	Begins with many small clusters and merges.
Complexity	More computationally intensive.	Less computationally intensive.
Number of Clusters	Can specify a desired number of clusters or stop when each data point forms its own cluster.	Typically, the number of clusters is determined by the algorithm based on the structure of the data.
Resulting Structure	Produces a dendrogram showing the hierarchy of clusters.	Also produces a dendrogram showing the hierarchy of clusters.
Computational Efficiency	Less efficient, especially for large datasets.	More efficient, particularly for large datasets.
Common Algorithms	DIANA (Divisive Analysis) is a notable example.	Single Linkage, Complete Linkage, Average Linkage, Ward's Method, etc.

Fuzzy C-Means Clustering

Fuzzy C-Means (FCM) clustering is a method used in data mining and machine learning for partitioning a dataset into clusters. It is an extension of the well-known K-Means clustering algorithm, but with the added flexibility of allowing data points to belong to multiple clusters with varying degrees of membership (or "fuzziness").

Here's a brief overview of how FCM works:

Initialization: Begin by randomly initializing cluster centroids and setting a fuzziness parameter (usually denoted by 'm').
Membership Assignment: Calculate the degree of membership of each data point to each cluster. This is done using a membership function, typically based on the Euclidean distance between the data point and the cluster centroid. Unlike in K-Means, where a point belongs only to the nearest cluster,

in FCM, a point can belong to multiple clusters with varying degrees of membership.
Centroid Update: Recalculate the cluster centroids using the updated membership values. Each centroid is computed as a weighted average of all data points, where the weights are determined by the membership values.
Iteration: Steps 2 and 3 are repeated iteratively until convergence, where either the membership values or the centroids (or both) stop changing significantly.
Termination: The algorithm stops when a stopping criterion is met, such as reaching a maximum number of iterations or when the membership values/centroids have converged.

FCM is sensitive to the choice of the fuzziness parameter 'm'. A higher value of 'm' allows for more overlap between clusters, resulting in softer boundaries, whereas a lower value of 'm' leads to harder, more distinct cluster boundaries. Selecting an appropriate value for 'm' is often done empirically or through techniques like cross-validation.

FCM has various applications, including image segmentation, pattern recognition, and customer segmentation in marketing. However, it can be computationally expensive, especially for large datasets, due to its iterative nature.

Overall, FCM provides a flexible approach to clustering that accommodates uncertainty and overlapping clusters, making it suitable for situations where data points may belong to multiple groups simultaneously.

4Fuzzy C-means.pptx

Gaussian Mixtures as Soft K-means Clustering

- It's a probabilistic version of KMeans
- KMeans uses a distance-based approach
- GMM uses a probabilistic approach
- KMeans have below limitations
  - It's a hard clustering method. Meaning each data point is assigned to a single cluster.
- There is one primary assumption in GMM
    - the dataset consists of multiple Gaussians, in other words, a mixture of the gaussian.

Gaussian Mixture Model (GMMs)

- A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters.
- In a Gaussian mixture model, each cluster is associated with a multivariate Gaussian distribution, and the mixture model is a weighted sum of these distributions.
- The weights indicate the probability that a data point belongs to a particular cluster, and the Gaussian distributions describe the distribution of the data within each cluster.

- The parameters of a Gaussian mixture model can be estimated using the expectation-maximization (EM) algorithm.
- This involves alternating between estimating the parameters of the Gaussian distributions and the weights of the mixture model until convergence is reached.
- Hence, for a dataset with *d* features, we would have a mixture of *k* Gaussian distributions (where *k* is equivalent to the number of clusters), each having a certain mean vector and variance matrix.

Gaussian Mixture Models (GMM):

- It is a probabilistic version of K-Means clustering. Unlike KMeans which assigns data points to clusters based on distance, GMM takes a probabilistic approach.
- A key assumption of GMM is that the dataset consists of a mixture of multiple Gaussian distributions. In other words, the data has been generated from a mixture of several different Gaussian distributions.
- In a GMM, each cluster is associated with a Gaussian distribution with its own mean, covariance and weight (mixing coefficient). The entire dataset is modeled as a weighted sum of these Gaussian distributions.

How it works:

- The parameters (mean, covariance, weights) of the GMM are estimated using an iterative process called the Expectation Maximization (EM) algorithm.
- E-step: For each data point, calculate the probability (responsibility) that it came from each of the k Gaussian distributions using Bayes theorem and the current estimates of parameters.
- M-step: Update the parameters (mean, covariance, weights) of each Gaussian distribution in order to maximize the overall likelihood of the data under the current responsibilities estimated in E-step.
- These steps are repeated alternately until model has converged and log-likelihood is maximized.
- So GMM considers both mean and variance of data to model it as coming from a mixture of Gaussians, as opposed to KMeans which only uses mean to define clusters.

In summary, GMM is a soft clustering probabilistic technique that models data as being generated from a mixture of Gaussian distributions. It iteratively estimates the parameters of these distributions to fit the data using the EM algorithm

Linear SVM Multi Class Classifier

A Linear Support Vector Machine (SVM) is a supervised learning algorithm used for classification tasks. In the case of a multiclass classification problem, where there are more than two classes, one common approach is to use a one-vs-all (also known as one-vs-rest) strategy. This approach trains a binary classifier for each class, which distinguishes that class from all other classes.

Here's a basic overview of the algorithm:

1. **Data Preparation:** Given a dataset with features X and corresponding labels y, where X is an $m \times n$ matrix (m samples, n features), and y is a vector of labels.
2. **Training Phase:**
   - For each class i, create a binary target variable y_i, where $y_i = 1$ if the sample belongs to class i and $y_i = -1$ otherwise.
   - Train a linear SVM classifier h_i on the data (X, y_i), where the objective function is to find the optimal separating hyperplane that maximizes the margin between the classes.
3. **Prediction Phase:**
   - For a new sample x, compute the decision function $f_i(x)$ for each classifier h_i.
   - Predict the class with the highest decision function value, i.e., $\hat{y} = \text{argmax}_i(f_i(x))$.

The decision function $f_i(x)$ for a linear SVM can be expressed as:

$$f(x) = \text{sign}(\sum_{j=1}^{n} w_{ij}x_j + b_i)$$

Where:

- w_{ij} are the coefficients of the hyperplane for class i obtained during training.
- b_i is the bias term for class i.

The training phase typically involves optimizing the following objective function:

$$\min_{w_i, b_i} \frac{1}{2}||w_i||^2$$

Subject to the constraints:

$$y_i(w_i^T x^{(j)} + b_i) \geq 1 \text{ for } j = 1, 2, ..., m$$

This can be solved using optimization techniques such as gradient descent or quadratic programming.

The prediction phase involves calculating the decision function $f_i(x)$ for each class and assigning the sample to the class with the highest decision function value.

This is the basic idea behind implementing a linear SVM for multiclass classification. There are variations and optimizations to this algorithm, such as using different loss functions or regularization techniques, but the core principles remain the same.

Linear Multi-Class Classifier:

- For a multi-class classification problem, a score function is defined which gives a score value for an instance x belonging to each possible class y.
- The goal is to learn parameters of the score function such that the score sy for the true class y is maximum among all class scores.
- This provides confidence in the classification instead of just predicting the class with maximum score.

Score Function:

- Parameters: w1, w2,...,wc (weight vectors for each class)
- For an instance xi, the score for class yj is given by the dot product wj · xi
- Want to learn the parameters wj such that the score syi for the true class yi is maximum.

Loss Function:

- A hinge loss function is used that penalizes cases where the score for the true class syi is not greater than other class scores by some margin.
- Regularization loss is added to prevent overfitting.
- Total loss function to minimize is: Data Loss + Regularization Loss
- Two hyperparameters:
    - C controls the tradeoff between data loss and regularization
    - Margin size controls the gap between class scores

Optimization:

- Gradient descent algorithm is used to iteratively update parameters wj to minimize the loss function over training data.

In summary, a linear multi-class classifier with parameters for score functions of each class is trained by optimizing a regularized hinge loss function using gradient descent.

Difference Between

1. **SVM Linear Multi-class Classification:**
   - Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification tasks.
   - SVM with a linear kernel is suitable for linearly separable datasets.
   - In multi-class classification, SVM can be extended to handle multiple classes using strategies like One-vs-One or One-vs-All.
   - It aims to find the hyperplane that best separates different classes in feature space.
2. **SVM Linear Classification:**
   - This refers to a binary classification scenario using SVM with a linear kernel.
   - The algorithm finds the best hyperplane that separates two classes with a maximum margin.
3. **KNN Classification:**
   - K-Nearest Neighbors (KNN) is a non-parametric and instance-based learning algorithm.
   - It makes predictions based on the majority class of its k-nearest neighbors in feature space.
   - KNN can be used for both binary and multi-class classification tasks.
   - It doesn't learn explicit models but rather memorizes the training data.
4. **Linear Classification:**
   - This is a general term referring to classification models that assume a linear relationship between features and class labels.
   - Algorithms like Logistic Regression and SVM with linear kernels fall under this category.
   - Linear classifiers are suitable for linearly separable data.

5. **Naive Bayes Classification:**
   - Naive Bayes is a probabilistic classifier based on Bayes' theorem.
   - It assumes independence between features, which is why it's called "naive."
   - Despite its simplicity, Naive Bayes often performs well in text classification and other tasks with high-dimensional data.
   - It's computationally efficient and easy to implement.
6. **Decision Tree Classification:**
   - Decision Trees are a type of supervised learning algorithm used for classification and regression tasks.
   - They partition the feature space into regions based on feature values.
   - Each internal node represents a decision based on a feature, leading to splits in the data.
   - Decision Trees can handle both categorical and numerical data and are capable of capturing complex relationships.

1. **Gaussian Mixture as Soft K-means Clustering:**
   - **Gaussian Mixture Models (GMM)** represent data as a mixture of several Gaussian distributions. It's a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions.
   - GMM can be interpreted as a generalization of K-means clustering where instead of strict assignments of points to clusters, there are probabilities or "soft assignments" indicating the likelihood of a point belonging to each cluster.
2. **Fuzzy C-Means (FCM):**
   - FCM is a clustering method where each data point belongs to every cluster with a certain degree of membership ranging from 0 to 1.
   - Unlike K-means where a point is strictly assigned to one cluster, in FCM, a point can belong to multiple clusters simultaneously with different degrees of membership.

3. **Hierarchical Clustering:**
   - Hierarchical clustering builds a hierarchy of clusters either top-down (divisive) or bottom-up (agglomerative) by merging or splitting clusters based on their similarities.
   - It does not require a predefined number of clusters and produces a dendrogram which can be cut at different levels to obtain different numbers of clusters.
4. **K-means++:**
   - K-means++ is an improvement over the standard K-means algorithm for initializing cluster centroids.
   - It chooses initial centroids in a way that the probability of selecting a point as the centroid is proportional to its distance from the existing centroids, helping to mitigate the problem of convergence to suboptimal solutions.
5. **K-means Clustering:**
   - K-means is a popular clustering algorithm that partitions data into K clusters where each data point belongs to the cluster with the nearest mean.
   - It aims to minimize the within-cluster sum of squares, often called inertia or distortion.

1. **Simple Linear Regression:**
   - Involves finding the best-fitting straight line through the data points.
   - Suitable when there is a linear relationship between the independent and dependent variables.
2. **Multiple Linear Regression:**
   - Extends simple linear regression to multiple independent variables.
   - It models the relationship between the dependent variable and two or more independent variables.
3. **Polynomial Regression:**
   - It fits a curve to the data points by using a polynomial equation.
   - Can capture more complex relationships between variables compared to linear regression.
4. **Piecewise Regression:**
   - Involves fitting multiple regression models to different segments of the data.
   - Useful when the relationship between variables changes at certain points or intervals.

5. **KNN for Regression** (K-Nearest Neighbors Regression):
   - Predicts the value of a new data point by averaging the values of its k nearest neighbors.
   - It's a non-parametric method, meaning it doesn't assume anything about the underlying data distribution.
6. **Decision Tree Regression**:
   - Uses a decision tree to predict the value of a target variable.
   - The tree splits the data into subsets based on the value of the independent variables.
7. **SVR (Support Vector Regression)**:
   - A type of support vector machine (SVM) used for regression tasks.
   - It finds the hyperplane that best fits the data, with a margin of tolerance for errors.
   - Particularly useful when dealing with non-linear data and high-dimensional spaces.