Tabular data

Gradient boosted decision tree model
Classification
Or
Regression
85% is tabular data
Features is data or parameter
Model parameters is hyperparameter
Datapoint is rows of record
Gini is a loss function
What is gini ?
Sum(k2) here is k is the sample of data chunks in page 9 /27 first lecture notes
We have to minimize the gini, means need to minimize the loss at every child node the gini should reduce.
The average of gini, Is called gini coefficient or Entropy
Cart Training algorithm page 11/27
Gini calculation is on page 15/27
Entropy is used for classification problem
Regression problem the loss is Mean square error -> MSE, regression is a continues number.
Confusion Matrix in Page 17/27
Recall = TP/(TP+FN), "Of all the actual positives, how many did the model correctly predict?", Use **Recall** when **False Negatives** are costly.
Precision = TP/(TP+FP), "Of all the positive predictions made by the model, how many were actually correct?" , Use **Precision** when **False Positives** are costly.

F1 Score = Harmonic mean of Recall and Precision, It is particularly useful when the dataset is imbalanced (i.e., when one class is much more frequent than the other)., Use **F1 Score** when you want a balance between Precision and Recall, especially for imbalanced datasets.

Always go with F1 score, if the F1 score is high than the model is good

False Positive is Type1 Error,
False Negative is Type2 Error
Trick to remember is P is drawn with 1 line, N is drawn with 2 lines
For Regression to find if the mode is good, use MAPE or R2
Mape: Mean absolute percentage error and R2 is 1- Mse/variance
If r2 is < 0 than the mode is bad / worst

Regression model is trained with MSE
Classification mode is trained with gini / entropy
Regression model is evaluated with R2 or MAPE
Classification mode is evaluated with confusion matrix

.fit
.predict

K-fold
LabelEncoder, OrdinalEncoder
GridSearchCv
Stratify

The key hyper-parameters for DecisionTreeClassifier include:

1. **max_depth**: Limits the maximum depth of the tree to prevent overfitting. A shallower tree may generalize better.
2. **min_samples_split**: Specifies the minimum number of samples required to split an internal node. It controls the size of the leaf nodes.
3. **min_samples_leaf**: Sets the minimum number of samples that must be present in a leaf node. This helps smooth the model and prevent overfitting.
4. **max_features**: Determines the number of features to consider when looking for the best split. It can improve model generalization and reduce computation time.
5. **criterion**: Specifies the function to measure the quality of a split (e.g., `gini` or `entropy`). This influences how splits are determined.
6. **max_leaf_nodes**: Limits the number of leaf nodes in the tree, helping to simplify the model.

sweet spots
    data split = 80:20
    K-fold = 5

Most Important Hyperparameters for decision tree

Tree-specific hyperparameters control the construction and complexity of the decision trees:max_depth: maximum depth of a tree. Deeper trees can capture more complex patterns in the data, but may also lead to overfitting.

min_child_weight: minimum sum of instance weight (hessian) needed in a child. This can be used to control the complexity of the decision tree by preventing the creation of too small leaves.

subsample: percentage of rows used for each tree construction. Lowering this value can prevent overfitting by training on a smaller subset of the data.

colsample_bytree: percentage of columns used for each tree construction. Lowering this value can prevent overfitting by training on a subset of the features.

Learning task-specific hyperparameters control the overall behavior of the model and the learning process:eta(also known as learning rate): step size shrinkage used in updates to prevent overfitting. Lower values make the model more robust by taking smaller steps.
gamma: minimum loss reduction required to make a further partition on a leaf node of the tree. Higher values increase the regularization.
lambda: L2 regularization term on weights. Higher values increase the regularization.
alpha: L1 regularization term on weights. Higher values increase the regularization.

Here’s a **table format** for the key hyperparameters of XGBoost to fine-tune:

Hyperparameter	Description	Typical Range	Effect of Tuning
`learning_rate` **(eta)**	Step size for gradient updates. Lower values slow training but improve precision.	`0.01 - 0.3`	Too high → Overshooting; Too low → Slow training.
`n_estimators`	Number of boosting rounds (trees).	`100 - 1000+`	Too high → Overfitting; Too low → Underfitting.
`max_depth`	Maximum depth of each tree, controls model complexity.	`3 - 10`	Too high → Overfitting; Too low → Underfitting.
`subsample`	Fraction of training samples used per tree.	`0.5 - 1.0`	Low → Reduces overfitting; High → Risks overfitting, especially with noisy data.
`colsample_bytree`	Fraction of features used for each tree.	`0.5 - 1.0`	Low → Adds randomness, reduces overfitting; High → Uses more features.

Hyperparameter	Description	Typical Range	Effect of Tuning
`colsample_bylevel`	Fraction of features used at each tree level.	`0.5 - 1.0`	Similar effect as `colsample_bytree` but applies at the level of splits.
`colsample_bynode`	Fraction of features used for each split in a tree.	`0.5 - 1.0`	Adds fine-grained control over feature sampling.
`lambda`	L2 regularization term on weights. Helps reduce overfitting.	`0 - 10`	Higher values → Stronger regularization, prevents large weights.
`alpha`	L1 regularization term on weights. Helps reduce overfitting and sparse models.	`0 - 10`	Higher values → Encourages sparsity, reduces feature impact.
`gamma`	Minimum loss reduction required to make a split.	`0 - 5`	High → Fewer splits, simpler trees; Low → Allows more splits, complex trees.
`min_child_weight`	Minimum sum of weights for child nodes. Prevents overfitting.	`1 - 10`	High → Prunes small splits; Low → Allows small splits.
`scale_pos_weight`	Balances positive and negative classes in imbalanced datasets.	`1 - 10` (depends on imbalance)	Adjust based on the ratio of negative to positive classes.
`objective`	Defines the learning task and loss function.	Problem-specific (`binary:logistic`, `reg:squarederror`, etc.)	Selecting the wrong objective impacts model performance.

This table gives an overview of XGBoost hyperparameters and their typical values, effects, and impact on model performance. You can customize ranges and choices based on your dataset and problem.
4o

Neural Network, Week3 tabular data

Neural network is a data processing pipeline

What is a neuron?

Is computation unit, data processing unit, It does two activities, Takes a weighted liner combination of input, add a bias and applies an activation function on the result.

28x28

224x224

Layer2 = 1 =(# Laye1 neuron + 1 bias) * # neuron in layer2

(784+1) * 100

Sum of all the above is the number of parameters

Stochastic gradient descent (variant called ADAM) is used for the training / back propagation, this is done only when training, and not when prediction or inference is done.

Training error should always keeping going down, unless you have a bug in your code
We are interested in the validation error, when the validation error staring increasing that is the time we stop.

Entire data is used for Gradient Descent

Comparison between gradient vs stochastic

Comparison Table

Feature	Gradient Descent	Stochastic Gradient Descent (SGD)
Data Used Per Update	Entire dataset	One data point or small batch
Speed	Slower for large datasets	Faster due to fewer computations
Stability	Stable, smooth convergence	Unstable, may oscillate
Memory Usage	High (entire dataset in memory)	Low (only one or a few samples needed)
Suitability	Small to medium datasets	Large datasets
Risk of Local Minima	More prone to get stuck	Less prone due to randomness

What is local minma
What is global minma

Imagine you're playing a game to find the deepest hole in a garden:

- **Local Minima:** You find a small hole and think, "This must be the deepest!" but another, much deeper hole exists somewhere else.

- **Global Minima:** You keep searching and finally find the deepest hole, which wins you the game!

Takeaway:

- **Local minima:** Good, but not the best solution.
- **Global minima:** The best possible solution.

Difference between Iteration, batch, epoch

Batch is a small group of data points taken from the dataset, **Batch size** affects memory usage and the speed of training.

An iteration is one step of gradient descent, where the model processes one bath and updates its parameters (weights and biases), **Number of iterations** controls how often the model updates its parameters.

If the dataset is divided into N batches, there be will N iterations in one epoch.

Epoch: An epoch is one complete pass through the entire dataset, You might train the model for multiple epochs (e.g., 50 epochs), meaning the model will see the entire dataset 50 times., Reading the entire book once is an epoch

Here's a clear explanation of **iteration**, **batch**, **epoch**, and **mini-batch**, along with their differences in a table format:

Term	Definition	Example
Iteration	A single step where the model processes a single batch of data and updates its weights once.	If you have 1,000 data samples, batch size = 100, then **10 iterations** complete 1 epoch.
Batch	A subset of the training dataset used in a single iteration.	If dataset = 1,000 samples, batch size = 100 → Each batch contains **100 samples**.
Epoch	One complete pass through the entire training dataset.	If dataset = 1,000 samples, batch size = 100 → **10 iterations = 1 epoch**.
Mini-Batch	Smaller subsets of the training data that are processed in each iteration (when batch size < dataset).	If dataset = 1,000 samples, batch size = 10 → Each **mini-batch** contains **10 samples**.

Key Differences

Aspect	Iteration	Batch	Epoch	Mini-Batch
Scope	One step of processing.	Subset of data in an iteration.	Full pass through the entire dataset.	Subset of data (smaller than dataset).
Timeframe	Shortest (1 step).	Medium (1 iteration).	Longest (all batches in the dataset).	Same as batch (used interchangeably).
Purpose	Weight update.	Allows for efficient computation.	Ensures all data contributes to training.	Reduces memory usage, increases speed.
Size Relation	Depends on batch size.	Equal to the batch size.	Covers the entire dataset once.	Smaller than the dataset.
Usage Scenario	Defines a single computation step.	Refers to the data processed per step.	Indicates full data coverage in training.	Optimizes for large datasets.

Analogy for Understanding

Imagine training a dog:

- **Iteration:** Teaching the dog one trick (single step).
- **Batch:** A small group of tricks taught together (one session).
- **Epoch:** Repeating all tricks the dog knows to ensure mastery (entire dataset).
- **Mini-Batch:** Breaking the tricks into even smaller groups to make it manageable.

This breakdown should help clarify the concepts!

4o

Convolutional Neural network layers (CNN)
Deep learning in a data processing pipeline

How to calculate the parameters in a cnn layer
3x3 kernel
9 (per kernel) * 3 (previous layer channel) * 2 *(current layer channels) +2 (current layer channel) = 56 parameters

$$\boxed{\text{Calculating Parameters} = \left(f^{l} \times f^{l} \times n_c^{l-1} + 1\right) \times n_f^{l}}$$

f^{l} = size of filter in layer 'L' : (3), n_c^{l-1} = number of channel in layer 'L-1' : (3), n_f^{l} = number of filter in layer 'L' : (2)

To calculate the number of units in the **Flatten** layer, we need to compute the output dimensions of the image after each convolutional layer in the given architecture. The calculations depend on the formula for the output size of a convolutional layer:

Formula for Convolutional Output:

$$\text{Output Size} = \frac{\text{Input Size} - \text{Filter Size}}{\text{Stride}} + 1$$

$$Output\ shape\ after\ Conv = \frac{n+2p-f}{s} + 1$$

n= size of input (6), p=padding (0), f=size of filter(3), s=stride(1)

Note: The input is an RGB image, so it has 3 channels initially, but for the calculations of the Flatten layer size, we only need the spatial dimensions (height × width) and the number of filters after each layer.

Overfitting : when training error is very low and validation error is very high

Underfitting: when training error and validation error difference is high

Resnet, Inverted Residual bottleneck, depthwise separable convolutions, xception, batch normalization, week 4

Yolo & segmentation week 5

Loss Function

Summary Table for Cross-Entropy

Type	Use Case	Handles Imbalance	Label Encoding
Binary Cross-Entropy	Binary classification	No	Integer
Categorical Cross-Entropy	Multi-class classification	No	One-Hot
Sparse Categorical Cross-Entropy	Multi-class classification	No	Integer
Weighted Cross-Entropy	Imbalanced classification	Yes	Integer/One-Hot
Focal Cross-Entropy	Imbalanced datasets	Yes	Integer/One-Hot

Type	Use Case	Handles Imbalance	Label Encoding
Label Smoothing Cross-Entropy	Reducing overconfidence in predictions	No	One-Hot

---

Tips for Selecting Cross-Entropy Type

- Use **binary cross-entropy** for binary problems.
- Use **categorical cross-entropy** for multi-class problems if you have one-hot labels.
- Use **sparse categorical cross-entropy** for multi-class problems with integer labels.
- Use **weighted or focal cross-entropy** for imbalanced datasets.
- Apply **label smoothing** to reduce overconfidence and improve generalization.

Summary Table for MSE

Type	Focus	Use Case
Standard MSE	Average squared error	General regression tasks
Weighted MSE	Error weighted by sample importance	When some samples are more important
Mean Squared Logarithmic Error (MSLE)	Relative errors	Large-scale differences in target values
Root Mean Squared Error (RMSE)	Error in original units	Interpretability in original scale
Normalized MSE (NMSE)	Variance explained by model	Comparison across datasets
MSE with Regularization	Combines MSE with complexity penalty	Prevents overfitting
Huber Loss	Hybrid of MSE and MAE	Outlier-resistant regression
Generalized MSE	Customizable error sensitivity	Specialized regression tasks

---

Tips for Choosing the Right MSE Variant

1. **Standard MSE:** For general regression problems.
2. **MSLE:** If relative errors matter more than absolute errors.
3. **Weighted MSE:** If you want to prioritize certain samples.
4. **Huber Loss:** For datasets with outliers.
5. **Normalized MSE:** When comparing models across different datasets.

4o

Formula in finding the dense layer parameters

To calculate the number of trainable parameters in a dense (fully connected) layer of a neural network, use the following formula:

Parameters=(Number of inputs×Number of neurons)+Number of biases\text{Parameters} = (\text{Number of inputs} \times \text{Number of neurons}) + \text{Number of biases}Parameters=(Number of inputs×Number of neurons)+Number of biases

Where:

- **Number of inputs**: The number of features coming into the layer.
- **Number of neurons**: The number of neurons in the dense layer.
- **Number of biases**: Each neuron has one bias, so this equals the number of neurons.

Sigmoid & softmax **activation functions, is used in the output layer**

When to Use:

- **Sigmoid**: Use for **binary classification**, where you need a single probability score.
- **Softmax**: Use for **multi-class classification**, where you need probabilities distributed over multiple classes.

Relu is used in the dense layer, Activation functions like ReLU and variants are crucial at every layer of cnn

Optimisation

- For **Gradient Boosting**, focus on **hyperparameter tuning** for tree and loss function regularization.
- For **CNNs**, start with **Adam** for faster convergence, but explore **SGD with Momentum** for potentially better generalization.

Regularization Techniques in CNN Cheat Sheet

Technique	How It Works	Why It Helps	Where to Apply	Example Code
Dropout	Randomly sets a fraction of neurons to zero during training.	Prevents reliance on specific neurons, reduces overfitting.	After dense or convolutional layers.	`model.add(Dropout(0.5))`
L2 Regularization (Ridge)	Adds squared weights to the loss function.	Penalizes large weights to avoid overfitting.	Weights of convolutional /dense layers.	`model.add(Dense(128, kernel_regularizer=l2(0.01)))`

Technique	How It Works	Why It Helps	Where to Apply	Example Code
L1 Regularization (Lasso)	Adds absolute values of weights to the loss function.	Encourages sparsity in weights.	Weights of convolutional/dense layers.	`model.add(Dense(128, kernel_regularizer=l1(0.01)))`
Batch Normalization	Normalizes layer inputs to zero mean and unit variance.	Stabilizes training and acts as mild regularizer.	After convolutional layers or before activation.	`model.add(BatchNormalization())`
Data Augmentation	Applies random transformations (e.g., flipping, rotation) to input data.	Increases dataset diversity, reducing overfitting.	During data preprocessing.	`datagen = ImageDataGenerator(rotation_range=20, horizontal_flip=True)`
Early Stopping	Stops training when validation loss stops improving.	Avoids overfitting by stopping at the optimal point.	Training phase.	`EarlyStopping(monitor='val_loss', patience=5)`
Gradient Clipping	Limits the magnitude of gradients during backpropagation.	Prevents exploding gradients and stabilizes training.	Optimizer.	`optimizer = tf.keras.optimizers.Adam(clipnorm=1.0)`
Learning Rate Scheduling	Dynamically adjusts the learning rate during training.	Fine-tunes weight updates, avoids overfitting.	Training phase.	`ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5)`
Max/Global Pooling	Reduces spatial dimensions of feature maps using	Reduces overfitting and computational	After convolutional layers.	`model.add(MaxPooling2D(pool_size=(2, 2)))`

Technique	How It Works	Why It Helps	Where to Apply	Example Code
	pooling operations.	complexity.		
Reduce Network Complexity	Reduces the number of layers, filters, or neurons in the model.	Simplifies the model to prevent overfitting.	Model architecture design.	Reduce number of filters or layers directly in model design.
Transfer Learning	Leverages pre-trained models and trains only the top layers for the task.	Uses generalized features learned on larger datasets.	For smaller datasets.	`base_model = tf.keras.applications.ResNet50(weights='imagenet', include_top=False); base_model.trainable = False`

Summary Table for Residual and Batch Normalization

Feature	Residual Connection	Batch Normalization
Purpose	Helps gradients flow in deep networks.	Stabilizes and accelerates training by normalizing inputs.
Formula	Output=F(x)+x\text{Output} = F(x) + xOutput=F(x)+x	y=γx−meanvar+ϵ+βy = \gamma \frac{x - \text{mean}}{\sqrt{\text{var}+\epsilon}} + \betay=γvar+ϵx−mean+β
Position	Across layers (skip connection).	After linear transformations, before activations.
Main Benefit	Easier optimization, enables very deep networks.	Faster convergence, reduced sensitivity to initialization.
Common Usage	In ResNet and similar architectures.	In most modern CNN architectures.

Residual Connection

A **residual connection**, introduced in ResNet (Residual Networks), helps address the vanishing/exploding gradient problem by allowing the gradient to flow through the network more easily.

How It Works:

- A residual connection skips one or more layers and directly adds the input of those layers to their output.
- The layer output becomes:
  Output = F(x) + x
  Where:
  - `F(x)` is the learned transformation by the layers (e.g., convolution, activation).
  - `x` is the input that is directly added (the skip connection).

Benefits:

1. **Eases Optimization:** Enables the network to learn the residual function F(x)=H(x)−xF(x) = H(x) - xF(x)=H(x)−x, where H(x)H(x)H(x) is the desired mapping. This is easier than learning H(x)H(x)H(x) directly.
2. **Mitigates Vanishing Gradient:** The gradient can flow directly through the skip connection, avoiding excessive diminishing during backpropagation.
3. **Deeper Networks:** Residual connections make it feasible to train very deep networks (e.g., ResNet-152).

Batch Normalization

Batch Normalization (BatchNorm) is a technique to normalize the inputs to a layer during training, which helps stabilize and accelerate the training process.

How It Works:

1. Computes the mean and variance of each feature in the batch.
2. Normalizes the input by subtracting the batch mean and dividing by the batch standard deviation: x^=x−meanvariance+ϵ\hat{x} = \frac{x - \text{mean}}{\sqrt{\text{variance} + \epsilon}}x^=variance+ϵx−mean
3. Scales and shifts the normalized values using learnable parameters γ\gammaγ (scale) and β\betaβ (shift): y=γx^+βy = \gamma \hat{x} + \betay=γx^+β

Benefits:

1. **Improved Convergence:** Reduces internal covariate shift, stabilizing the training process.
2. **Higher Learning Rates:** Makes training more robust to large learning rates.
3. **Regularization:** Acts as a mild form of regularization, reducing overfitting.

Where to Apply:

- Typically applied after a convolution or dense layer and before the activation function.

Choosing the Right Neural Network

The type of neural network depends on your **data** (e.g., image, text, sequential data) and **task** (e.g., classification, generation, segmentation).

Summary Table

Type	Purpose	Use Cases
Feedforward Networks	Simple input-output mapping	Classification, regression
Convolutional Networks	Grid data processing	Image recognition, video
Recurrent Networks	Sequential data processing	NLP, speech recognition
GANs	Data generation	Image synthesis, style transfer
Autoencoders	Feature compression and generation	Dimensionality reduction
Transformer Networks	Attention-based sequence modeling	Translation, text generation
Spiking Networks	Event-driven processing	Neuromorphic computing
Capsule Networks	Hierarchical feature learning	Image classification
Boltzmann Machines	Unsupervised learning	Collaborative filtering