SNS COLLEGE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)

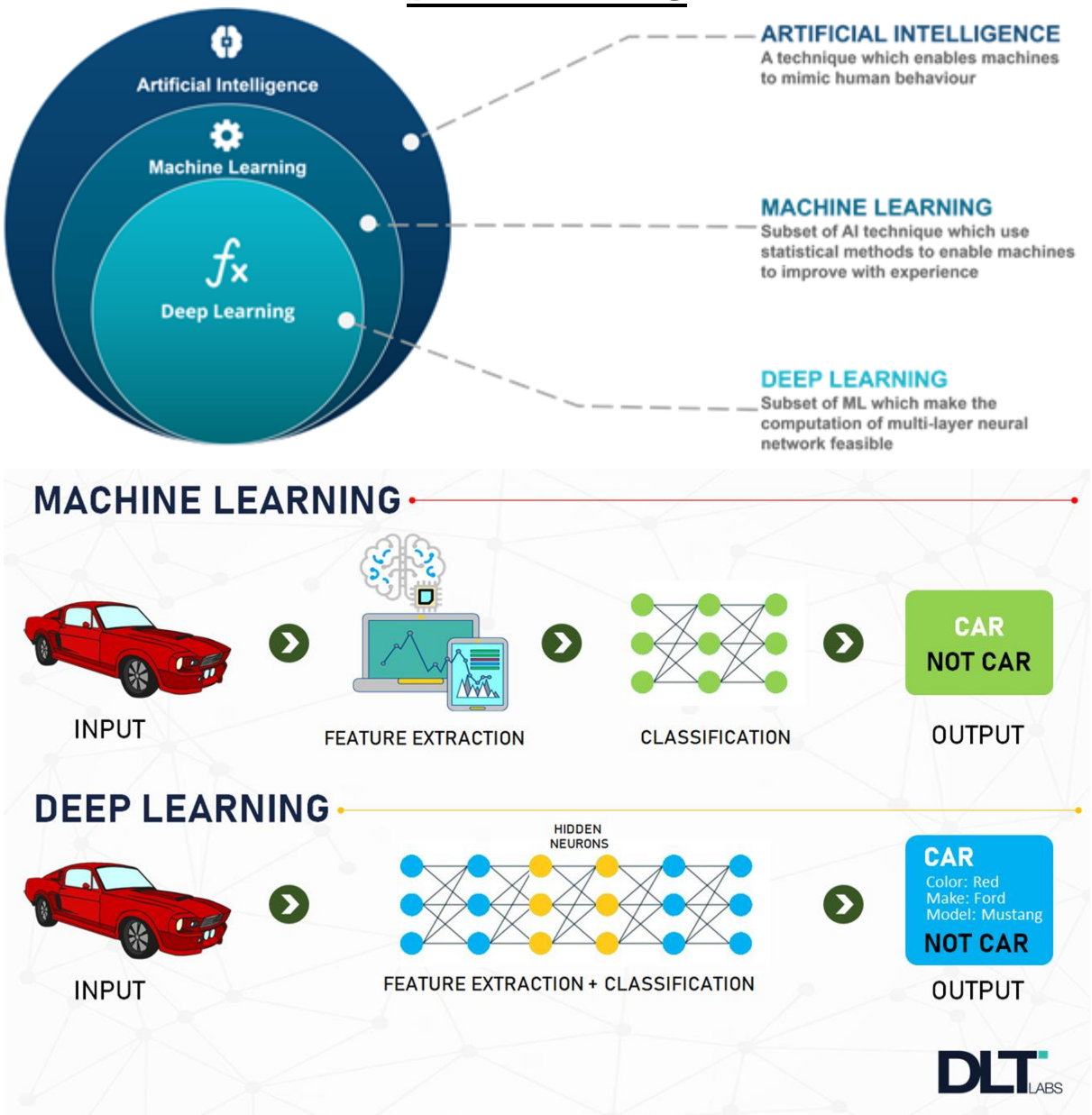
COIMBATORE – 35

DEPARTMENT OF COMPUTER SIENCE AND ENGINEERING

UNIT III DEEP LEARNING

Boosting and Additive Trees – Boosting Trees – Regularization – Interpretation – Illustrations
Neural Networks – Fitting Neural Network - Bayesian Neural Net
Neural Network Representation – Problems – Perceptron
Multilayer Networks
Back Propagation Algorithms
Case Study: Handwriting Recognition

DEEP LEARNING



Boosting and Additive Trees

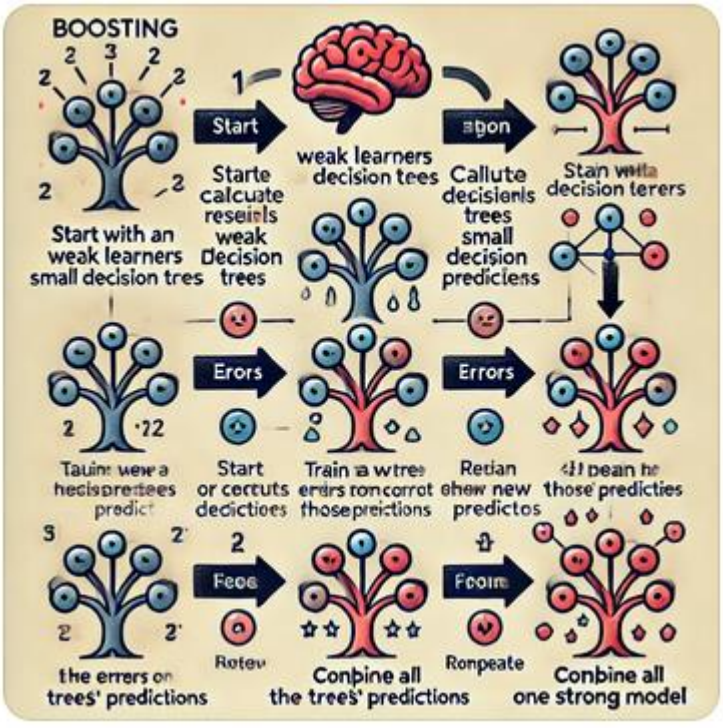
Boosting and Additive Trees are important concepts in ensemble learning, particularly in supervised machine learning. They are used to improve the performance of decision trees by combining multiple models.

Boosting

Boosting is an ensemble technique that combines several weak learners to create a strong predictive model. In boosting, models are trained sequentially, with each new model focusing on correcting the errors made by the previous models.

Additive Trees

Additive Trees are closely related to boosting, where the model is built as an additive series of trees, one at a time, with each new tree fitting the residuals (errors) of the previous trees.

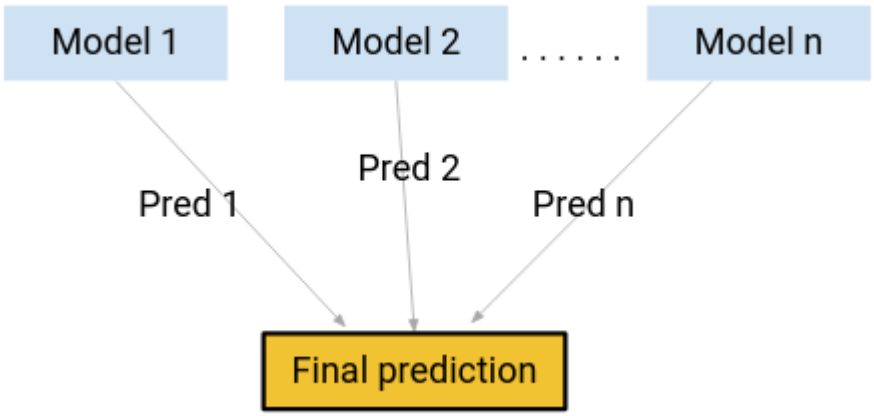


Introduction to Boosting (What is Boosting?)

Picture this scenario:

You've built a linear regression model that gives you a decent 77% accuracy on the validation dataset. Next, you decide to expand your portfolio by building a k-Nearest Neighbour (KNN) model and a decision tree model on the same dataset. These models gave you an accuracy of 62% and 89% on the validation set respectively.

It's obvious that all three models work in completely different ways. For instance, the linear regression model tries to capture linear relationships in the data while the decision tree model attempts to capture the non-linearity in the data.



How about, instead of using any one of these models for making the final predictions, we use a combination of all of these models?

I'm thinking of an average of the predictions from these models. By doing this, we would be able to capture more information from the data, right?

That's primarily the idea behind ensemble learning. And where does boosting come in?

Boosting is one of the techniques that uses the concept of ensemble learning. A boosting algorithm combines multiple simple models (also known as weak learners or base estimators) to generate the final output.

Boosting:

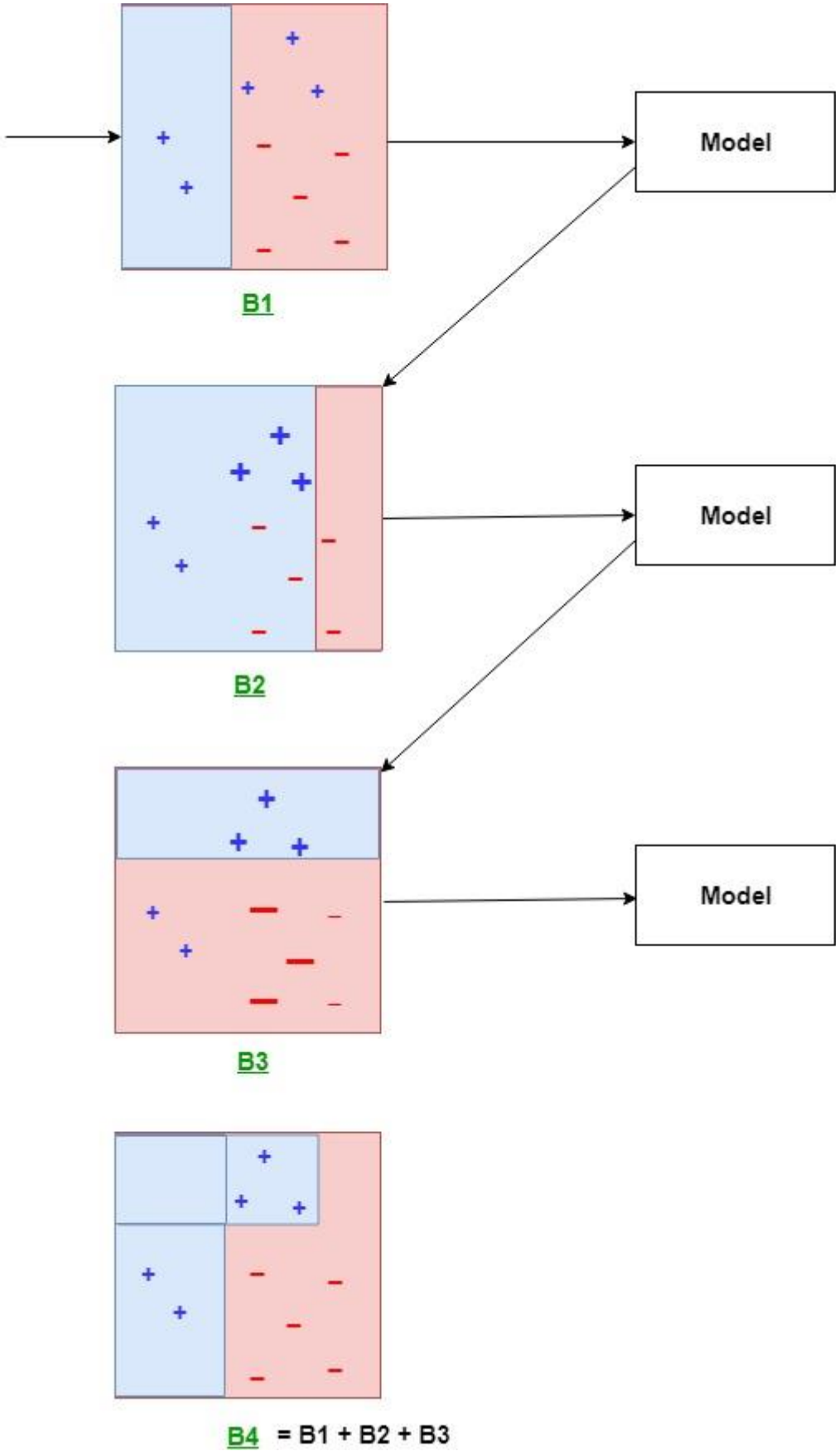
Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

AdaBoost was the first really successful boosting algorithm developed for the purpose of binary classification. *AdaBoost* is short for *Adaptive Boosting* and is a very popular boosting technique that combines multiple "weak classifiers" into a single "strong classifier". It was formulated by Yoav Freund and Robert Schapire. They also won the 2003 Gödel Prize for their work.

Algorithm:

1. Initialise the dataset and assign equal weight to each of the data point.
2. Provide this as input to the model and identify the wrongly classified data points.
3. Increase the weight of the wrongly classified data points.
4. if (got required results)
   Goto step 5
   else
   Goto step 2

5. End


Model
B1
Model
B2
Model
B3
B4 = B1 + B2 + B3

Explanation:
The above diagram explains the AdaBoost algorithm in a very simple way. Let's try to understand it in a stepwise process:

- **B1** consists of 10 data points which consist of two types namely plus(+) and minus(-) and 5 of which are plus(+) and the other 5 are minus(-) and each one has been assigned equal weight initially. The first model tries to classify the data points and generates a vertical separator line but it wrongly classifies 3 plus(+) as minus(-).
- **B2** consists of the 10 data points from the previous model in which the 3 wrongly classified plus(+) are weighted more so that the current model tries more to classify these pluses(+) correctly. This model generates a vertical separator line that correctly classifies the previously wrongly classified pluses(+) but in this attempt, it wrongly classifies three minuses(-).
- **B3** consists of the 10 data points from the previous model in which the 3 wrongly classified minus(-) are weighted more so that the current model tries more to classify these minuses(-) correctly. This model generates a horizontal separator line that correctly classifies the previously wrongly classified minuses(-).
- **B4** combines together B1, B2, and B3 in order to build a strong prediction model which is much better than any individual model used.

4 Boosting Algorithms in Machine Learning

1. Gradient Boosting Machine (GBM)
2. Extreme Gradient Boosting Machine (XGBM)
3. LightGBM
4. CatBoost

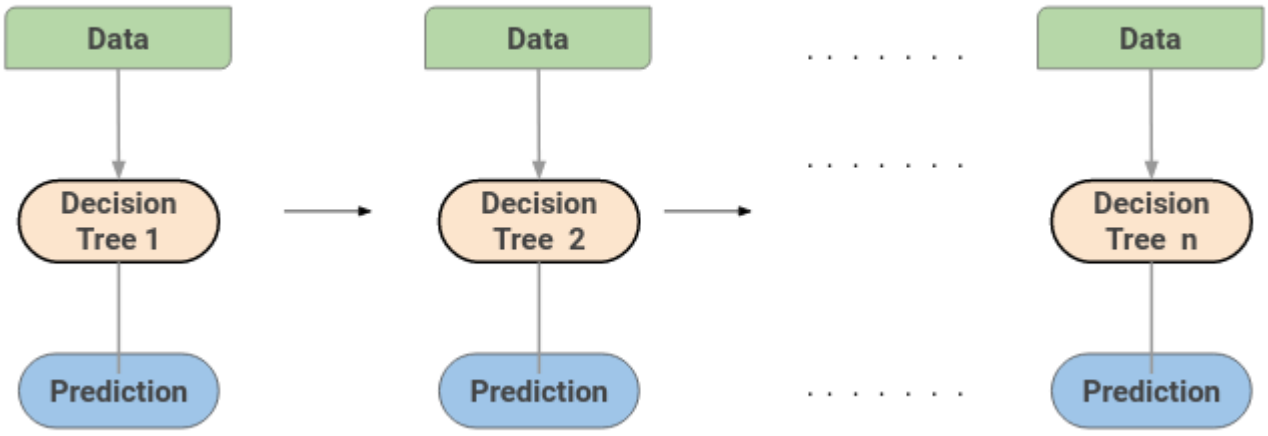
1. Gradient Boosting Machine (GBM)

A Gradient Boosting Machine or GBM combines **the predictions from multiple decision trees to generate the final predictions.** Keep in mind that all the weak learners in a gradient boosting machine are decision trees.

But if we are using the same algorithm, then how is using a hundred decision trees better than using a single decision tree? How do different decision trees capture different signals/information from the data?

Here is the trick – **the nodes in every decision tree take a different subset of features for selecting the best split.** This means that the individual trees aren't all the same and hence they are able to capture different signals from the data.

Additionally, each new tree takes into account the errors or mistakes made by the previous trees. So, every successive decision tree is built on the errors of the previous trees. This is how the trees in a gradient boosting machine algorithm are built sequentially.



Here is an article that explains the hyperparameter tuning process for the GBM algorithm:

- Guide to Parameter Tuning for a Gradient Boosting Machine (GBM) in Python

2. Extreme Gradient Boosting Machine (XGBM)

Extreme Gradient Boosting or XGBoost is another popular boosting algorithm. In fact, XGBoost is simply an improvised version of the GBM algorithm! The working procedure of XGBoost is the same as GBM.

The trees in XGBoost are built sequentially, trying to correct the errors of the previous trees.

Here is an article that intuitively explains the math behind XGBoost and also implements XGBoost in Python:

- An End-to-End Guide to Understand the Math behind XGBoost

But there are certain features that make XGBoost slightly better than GBM:

- One of the most important points is that XGBM implements **parallel preprocessing (at the node level) which makes it faster than GBM**
- XGBoost also includes a variety of regularization techniques that **reduce overfitting and improve overall performance.** You can select the regularization technique by setting the hyperparameters of the XGBoost algorithm

Learn about the different hyperparameters of XGBoost and how they play a role in the model training process here:

- Guide to Hyperparameter Tuning for XGBoost in Python

Additionally, if you are using the XGBM algorithm, you don't have to worry about imputing missing values in your dataset. **The XGBM model can handle the missing values on its own**. During the training process, the model learns whether missing values should be in the right or left node.

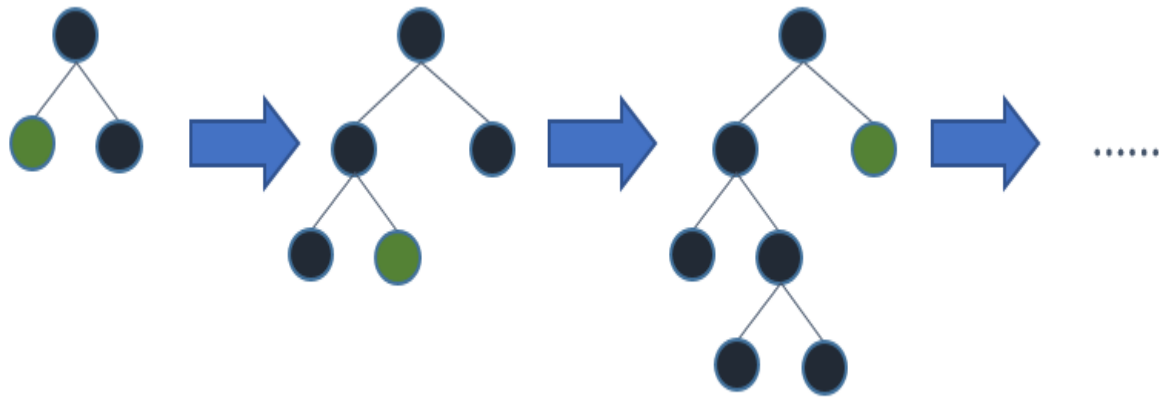
3. LightGBM

The LightGBM boosting algorithm is becoming more popular by the day due **to its speed and efficiency.** LightGBM is able to **handle huge amounts of data** with ease. But keep in mind that this algorithm **does not perform well with a small number of data points**.

Let's take a moment to understand why that's the case.

The trees in **LightGBM have a leaf-wise growth, rather than a level-wise growth**. After the first split, the next split is done only on the leaf node that has a higher delta loss.

Consider the example I've illustrated in the below image:



Leaf-wise tree growth

After the first split, the left node had a higher loss and is selected for the next split. Now, we have three leaf nodes, and the middle leaf node had the highest loss. The leaf-wise split of the LightGBM algorithm enables it to work with large datasets.

In order to speed up the training process, **LightGBM uses a histogram-based method for selecting the best split**. For any continuous variable, instead of using the individual values, these are divided into bins or buckets. This makes the training process faster and lowers memory usage.

Here's an excellent article that compares the LightGBM and XGBoost Algorithms:

- LightGBM vs XGBOOST: Which algorithm takes the crown?

4. CatBoost

As the name suggests, **CatBoost** is a boosting algorithm that **can handle categorical variables in the data.** Most machine learning algorithms **cannot work with strings or categories** in the data. Thus, converting categorical variables into numerical values is an essential preprocessing step.

CatBoost can internally handle categorical variables in the data. These variables are transformed to numerical ones using various statistics on combinations of features.

If you want to understand the math behind how these categories are converted into numbers, you can go through this article:

- Transforming categorical features to numerical features

Another reason why **CatBoost** is being widely used is that it works well with the default set of hyperparameters. Hence, as a user, we do not have to spend a lot of time tuning the hyperparameters.

Here is an article that implements CatBoost on a machine learning challenge:

- CatBoost: A Machine Learning Library to Handle Categorical Data Automatically

Additive-trees are used to represent objects as "leaves" on a tree, so that the distance on the tree between two leaves reflects the similarity between the objects. Formally, an observed similarity δ is represented by a tree-distance *d*. As such, additive-trees belong to the descriptive multivariate statistic tradition. Additive-tree representations are useful in a wide variety of domains.

Reference Links:

- https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/
- https://www.slideshare.net/afolaborn/boosting-approach-to-solving-machine-learning-problems
- https://www.analyticsvidhya.com/blog/2020/02/4-boosting-algorithms-machine-learning/
- https://towardsdatascience.com/introduction-to-boosted-trees-2692b6653b53

Additive Trees

Additive Trees refer to a method used in ensemble learning, where multiple decision trees are combined additively to form a stronger predictive model. The key concept is that each tree is added to improve upon the predictions made by the previous trees. This method is primarily used in **boosting** algorithms, where the trees are trained sequentially, with each tree attempting to correct the residual errors of the preceding model.

How Additive Trees Work**:**

1. **Initialize with a Base Model**: Start with a simple decision tree (or any other weak learner), which makes initial predictions.
2. **Compute Residuals (Errors)**: Calculate the difference between the predicted values and the true values, which gives you the residual errors.
3. **Fit the Next Tree**: Train a new decision tree to predict the residuals or errors from the previous tree.
4. **Add the New Tree to the Model**: The new tree's predictions are added to the existing model, improving the overall prediction by reducing the errors.
5. **Repeat**: This process is repeated multiple times, each time adding another tree that focuses on minimizing the remaining errors.

Final Model**:**

The final model is a sum of all the trees' predictions, which form an **additive** combination. Each tree contributes a part of the prediction, and the combined result is much more accurate than any individual tree.

Use of Additive Trees in Boosting**:**

- **Boosting** algorithms like **Gradient Boosting**, **AdaBoost**, **XGBoost**, etc., use this additive approach. In these algorithms, each tree is added sequentially to minimize a loss function (like Mean Squared Error for regression or Log Loss for classification).
- The model becomes progressively better with each added tree, reducing both bias and variance.

Advantages**:**

- **Improved Accuracy**: Each additional tree improves the model's performance by focusing on errors of the previous ones.
- **Adaptability**: The method allows for flexibility in both regression and classification tasks.

Example Algorithms**:**

- **Gradient Boosting Machine (GBM)**: Each new tree minimizes the residual errors by performing gradient descent on the loss function.

- **AdaBoost**: Weights are adjusted to focus on the hardest-to-predict samples, and each new tree corrects errors of previous trees.

In summary, **Additive Trees** form the backbone of boosting algorithms, sequentially improving models by combining weak learners (typically decision trees) to make a strong, accurate predictive model.

Boosting Trees

Boosting Trees refers to the process of applying boosting techniques to decision trees, creating an ensemble of weak learners (small decision trees) that work together to produce a more accurate and robust predictive model. In boosting, trees are trained sequentially, and each subsequent tree corrects the errors made by the previous ones.

How Boosting Trees Work:

1. **Initial Weak Learner**: Start with a small decision tree (often called a "stump" because it's shallow).
2. **Calculate Errors**: After training the first tree, calculate the residual errors or misclassified instances.
3. **Train the Next Tree on Errors**: A new decision tree is trained to focus on these errors, paying more attention to the instances where the previous tree made mistakes.
4. **Sequential Improvement**: This process is repeated, with each new tree improving the overall prediction by correcting the errors of the combined previous models.
5. **Final Prediction**: The final model is a weighted sum of the predictions from all the trees, where trees that performed better in correcting errors contribute more to the final result.

Types of Boosting Trees**:**

1. AdaBoost (Adaptive Boosting)**:**

- Assigns higher weights to misclassified instances and trains new trees to focus on these harder cases.
- Each subsequent tree is trained to correct errors made by the earlier ones.
- The final prediction is based on a weighted majority vote.

2. Gradient Boosting**:**

- Optimizes a loss function (such as Mean Squared Error or Log Loss) by adding new trees that predict the residuals (errors) of the previous model.
- More sophisticated than AdaBoost, Gradient Boosting allows trees to be added based on the gradient of the loss function.
- Advanced implementations include **XGBoost**, **LightGBM**, and **CatBoost**, which add regularization and performance optimizations.

Key Characteristics**:**

- **Sequential Learning**: Boosting Trees are trained one at a time, with each tree improving the accuracy by focusing on errors made by the previous models.
- **Weighted Predictions**: In boosting, the predictions from each tree are combined in a weighted manner, with higher-performing trees contributing more to the final prediction.
- **Correcting Bias**: Boosting works by reducing both bias and variance, making the final model more accurate.

Advantages of Boosting Trees**:**

- **High Accuracy**: Boosting trees generally outperform many other algorithms because they focus on correcting mistakes iteratively.
- **Flexibility**: Can handle regression, classification, and even ranking problems.
- **Less Overfitting (with Regularization)**: Regularization techniques can be applied to prevent overfitting, as seen in XGBoost and LightGBM.

Challenges**:**

- **Slower Training**: Because trees are trained sequentially, boosting is slower compared to parallelized algorithms like bagging.
- **Overfitting**: If not properly tuned, boosting can overfit the training data, especially if the model is too complex.

Boosting Trees are widely used in a variety of fields, such as finance, healthcare, and marketing, due to their ability to handle complex datasets with great accuracy.

Regularization

Interpretation
Illustrations

Regularization in deep learning refers to techniques used to prevent overfitting, which occurs when a model learns the training data too well, including its noise and outliers, and fails to generalize to unseen data.

Here are some common regularization methods:

1. L1 and L2 Regularization

- **L1 Regularization (Lasso)**: Adds a penalty equal to the absolute value of the weights. This can lead to sparsity, effectively reducing the number of features.
- **L2 Regularization (Ridge)**: Adds a penalty equal to the square of the weights. This helps to shrink the weights and can smooth the model.

2. Dropout

- A technique where randomly selected neurons are ignored during training. This prevents co-adaptation of hidden units and encourages the network to learn more robust features.

3. Early Stopping

- Monitoring the model's performance on a validation set during training and stopping the training process when performance starts to degrade, indicating overfitting.

4. Data Augmentation

- Increasing the size of the training dataset by applying random transformations (like rotations, translations, and flips) to the existing data. This helps the model generalize better.

5. Batch Normalization

- Normalizing the outputs of a layer for each mini-batch. This can lead to faster convergence and can have a slight regularization effect by introducing some noise into the training process.

6. Weight Noise

- Adding small random noise to the weights during training to make the model more robust and prevent overfitting.

7. Ensemble Methods

- Combining predictions from multiple models to improve generalization. Techniques like bagging and boosting fall under this category.

8. Transfer Learning

- Utilizing a pre-trained model on a similar task and fine-tuning it on the target dataset. This can help reduce overfitting, especially when the target dataset is small.

Conclusion

Regularization is crucial in deep learning, as it helps create models that generalize well to new data. The choice of regularization technique often depends on the specific problem, the architecture of the model, and the dataset at hand.

Interpretation

1. L1 Regularization (Lasso)

- **Interpretation:**
  - Adds a penalty to the loss function equal to the absolute values of the weights.
  - Encourages sparsity by pushing some weights to zero, effectively performing feature selection.
  - Ideal when you suspect that many features are irrelevant, as it can simplify the model.

2. L2 Regularization (Ridge)

- **Interpretation:**
  - Adds a penalty equal to the square of the weights to the loss function.
  - This discourages large weights, leading to a smoother model with smaller coefficients.
  - Helps to reduce the model complexity without eliminating features, making it useful when all features may contribute to the outcome.

3. Dropout

- **Interpretation:**
  - Randomly deactivates a subset of neurons during each training iteration.
  - Forces the network to learn robust features that are not reliant on any specific neuron.
  - Acts as a form of ensemble learning, as different subsets of the network are trained each time.

4. Early Stopping

- **Interpretation:**
  - Involves monitoring the model's performance on a validation set during training.
  - Stops training when performance on the validation set begins to degrade, which indicates potential overfitting.
  - This allows the model to retain the best weights before overfitting occurs.

5. Data Augmentation

- **Interpretation:**
  - Involves artificially increasing the size of the training dataset by applying random transformations to existing data.
  - Helps the model generalize better by exposing it to variations of the input data, thereby reducing overfitting.
  - Particularly useful in image classification tasks, where transformations like rotation or flipping can create a diverse dataset.

6. Ensemble Methods

- **Interpretation:**
  - Combine predictions from multiple models to improve overall performance.
  - By averaging or voting, these methods reduce variance and improve robustness, leading to better generalization.
  - Examples include bagging (e.g., Random Forests) and boosting (e.g., AdaBoost).

7. Transfer Learning

- **Interpretation:**
  - Utilizes knowledge gained from one task (usually with a larger dataset) to improve learning on a related task with a smaller dataset.
  - Fine-tuning pre-trained models allows for better performance without requiring extensive data, reducing the risk of overfitting.
  - Effective when the target domain has limited labeled data.

8. Weight Noise

- **Interpretation:**
  - Introduces small amounts of noise to the weights during training.
  - This randomness can prevent the model from becoming too reliant on specific weights, encouraging robustness.
  - Acts similarly to dropout but affects the entire weight distribution.

Regularization techniques in machine learning play a vital role in enhancing model performance and preventing overfitting. Each method has its unique approach to controlling complexity, ensuring that the model not only fits the training data well but also generalizes effectively to new, unseen data. Understanding these interpretations can help in selecting the right regularization techniques based on the specific characteristics of the dataset and the problem at hand.

1. L1 Regularization (Lasso)

- **Illustration:**
  - **Visual:** Show a scatter plot with a linear regression line. Highlight how L1 regularization leads to some weights being exactly zero, creating a simpler model.
  - **Example:** Display two models—one with L1 regularization showing fewer active features and another without it, with many non-zero weights.

2. L2 Regularization (Ridge)

- **Illustration:**
  - **Visual:** A weight distribution graph where L2 regularization compresses weights closer to zero, compared to a model without regularization where weights are more spread out.
  - **Example:** Use a comparison of model predictions with and without L2 regularization, showing smoother predictions with L2.

3. Dropout

- **Illustration:**
  - **Visual:** A neural network diagram with some neurons 'grayed out' or crossed out to represent dropout during training.
  - **Example:** Show two network architectures—one with dropout and one without—indicating that the dropout model has to learn robust features that don't rely on any single neuron.

4. Early Stopping

- **Illustration:**
  - **Visual:** A graph plotting training and validation loss over epochs. Highlight the point where validation loss starts to increase.
  - **Example:** Use annotations to indicate the optimal stopping point, demonstrating how training was halted before overfitting occurred.

5. Data Augmentation

- **Illustration:**
  - **Visual:** A grid of images showing the original image alongside its augmented versions (e.g., rotated, flipped, color-changed).
  - **Example:** Include a flowchart showing how augmented data feeds into the training process, emphasizing its role in enhancing generalization.

6. Ensemble Methods

- **Illustration:**
  - **Visual:** A diagram showing multiple models (e.g., decision trees) combined to produce a final prediction (like a voting system).
  - **Example:** Use a bar graph to show the performance metrics (accuracy, precision) of individual models versus the ensemble, illustrating the improvement.

7. Transfer Learning

- **Illustration:**
  - **Visual:** A flowchart showing the process of taking a pre-trained model (like a CNN trained on ImageNet) and fine-tuning it for a new task (like a specific image classification).
  - **Example:** Show before-and-after accuracy metrics to demonstrate how transfer learning improves performance on the new dataset.

8. Weight Noise

- **Illustration:**
  - **Visual:** A graph depicting weight distributions before and after adding noise, showing more variation in the noisy model.
  - **Example:** Compare the performance of a model with and without weight noise through a line graph showing validation accuracy over epochs.

Summary of Illustrations

- Each illustration should clearly communicate the concept and impact of the respective regularization technique.
- Visual aids such as graphs, flowcharts, and comparison images can help the audience grasp the abstract concepts more concretely.
- Be sure to include annotations or captions to explain each illustration clearly and enhance understanding.