---

Slide 1: Ensemble Learning

This slide is the title slide, introducing the topic: **Ensemble Learning**. It also credits the presenter as DR. BHARGAVI, PROFESSOR, VIT CHENNAI.

**Explanation and Additional Knowledge:**

Ensemble learning is a powerful machine learning technique where, instead of training a single model, we train multiple models and then combine their predictions to achieve a better overall performance. Think of it like a group of diverse experts working together on a problem. Each expert (individual model) might have their own strengths and weaknesses, but by pooling their knowledge and decisions, the group's collective judgment is often superior to any single expert's opinion.

The main goals of ensemble learning are:

- **Improved Accuracy:** Ensembles typically achieve higher predictive accuracy than any single base model.
- **Increased Robustness:** They are less sensitive to specific data characteristics or the weaknesses of individual models. If one model makes a mistake, others might correct it.
- **Reduced Overfitting:** While individual complex models can overfit, well-designed ensembles often reduce overfitting by averaging out the noise or errors from individual models.

This field has had a massive impact on machine learning, consistently winning competitions and being widely adopted in various industries due to its effectiveness.

---

Slide 2: Simple Learners

This slide introduces the concept of "Simple Learners," also known as "weak learners."

- They are fast to learn.
- They have low variance.
- However, they suffer from high bias.
- Examples shown are Logistic Regression with few features, Shallow Decision Tree, and Decision Stump.

**Explanation and Additional Knowledge:**

A "simple learner" or "weak learner" is a model that is relatively straightforward and performs only slightly better than random guessing. On its own, it's not highly accurate, but it's computationally efficient.

- **Fast to learn:** These models don't require extensive computational resources or long training times. They can quickly grasp basic patterns in the data.
- **Low variance:** Variance refers to how much the model's predictions change when it's trained on different subsets of the data. A model with low variance is consistent in its predictions. However, this consistency often means it might not capture complex relationships.
- **High Bias:** Bias is the error introduced by approximating a real-world problem with a simplified model. A model with high bias makes strong assumptions about the data, leading it to "underfit" – meaning it fails to capture important patterns and relationships,

resulting in systematic errors. It's like trying to fit a straight line to data that clearly follows a curve; the line will consistently miss the true pattern.
The examples provided illustrate this:

- **Logistic Regression with few features:** If you use only a handful of features, a logistic regression model can't capture much complexity, leading to high bias.
- **Shallow Decision Tree:** A decision tree with only a few levels or "splits" makes very simple rules. It can't delve deep into the data to find intricate patterns.
- **Decision Stump:** This is the simplest form of a decision tree, consisting of just one split. It's essentially asking one true/false question to make a classification.

The brilliant idea behind ensemble learning is to combine many of these high-bias, low-variance weak learners in a smart way to create a single "strong learner" that achieves both low bias and low variance.

---

Slide 3: Simple Learners (cont...)

This slide continues the discussion on simple learners and poses a question: "How to model a learner that is just right?" It then offers two options to address the limitations (specifically high bias) of simple learners:

- **Option 1: Add more features (Logistic) or increase depth (Decision Tree)**
- **Option 2: Boosting with Ensemble learning**

**Explanation and Additional Knowledge:**

This slide brings up a fundamental concept in machine learning: the **bias-variance trade-off**. We want a model that is "just right" – meaning it's complex enough to capture the true patterns in the data (low bias) but not so complex that it starts memorizing noise and performing poorly on new data (low variance).

- **Option 1: Making a single learner more complex:**
  - **Add more features (Logistic Regression):** By giving a logistic regression model more relevant features, it has more information to work with, allowing it to potentially learn more complex relationships and reduce its bias. However, if you add too many irrelevant features or if the model becomes too complex, it can start to suffer from higher variance (overfitting).
  - **Increase depth (Decision Tree):** Making a decision tree deeper allows it to create more intricate decision boundaries and capture finer details in the data. A very deep tree can fit the training data almost perfectly, significantly reducing bias. However, deep decision trees are highly prone to **high variance** (overfitting). They become too specific to the training data and generalize poorly to new, unseen examples. The next slide, "Decision Tree - Pros and Cons," specifically mentions that single decision trees suffer from high variance.
- **Option 2: Boosting with Ensemble learning:** This is the path the presentation will focus on. Instead of trying to perfect *one* complex model, ensemble methods combine *many* simple, imperfect models. Boosting is a specific type of ensemble method designed to primarily reduce bias. It achieves this by sequentially building models, where each new model learns to correct the errors made by the models that came before it. This iterative error correction leads to a powerful model that is both accurate and robust.

The presentation is setting the stage to show how ensemble learning provides a more robust and often more effective solution than simply trying to make a single model more complex, especially for algorithms like decision trees.

---

Slide 4: Ensemble Models

This slide defines ensemble models and lists the main types of ensemble methods:

- An ensemble method combines many simple "building block" models in order to obtain a single and potentially very powerful model.
- Ensemble Methods:
  - Bagging
  - Boosting
  - Stacking
- Have an amazing impact. Wins most Kaggle competitions. Makes them widely used in Industry.

**Explanation and Additional Knowledge:**

This slide gives a concise overview of what ensemble models are and emphasizes their practical importance.

- **Definition:** At its core, an ensemble model is a "team" of models working together. Each "building block" model (often a weak learner like a shallow decision tree) contributes its prediction, and then these individual predictions are combined (e.g., averaged or voted upon) to produce a final, more accurate, and more reliable prediction. The idea is that the collective wisdom of the group is better than any single member's judgment.
- **Benefits of Ensemble Methods (Beyond what's on the slide):**
  - **Improved Accuracy:** This is the most significant advantage. Ensembles consistently outperform individual models on a wide range of tasks.
  - **Increased Robustness:** They are less sensitive to specific characteristics of the training data or to the individual weaknesses of their base models. If one model makes an error, others in the ensemble can compensate.
  - **Reduced Overfitting:** By combining multiple models, ensembles tend to average out the idiosyncratic errors and noise that individual models might pick up, leading to better generalization to unseen data.
- **Types of Ensemble Methods:** The three main categories are introduced here:
  - **Bagging (Bootstrap Aggregating):** Primarily focuses on reducing **variance**. It trains multiple models independently on different random subsets (bootstrapped samples) of the training data and then averages their predictions. **Random Forests** are a very popular and successful extension of bagging, specifically using decision trees as base learners.
  - **Boosting:** Primarily focuses on reducing **bias**. It trains models sequentially, where each new model tries to correct the errors made by the previous ones. **AdaBoost** and **Gradient Boosting** are prominent examples.
  - **Stacking (Stacked Generalization):** This is a more advanced technique that combines diverse models using *another model* (a "meta-learner") to learn how to best combine their predictions. It's like having a manager who learns how to best utilize the skills of different team members.
- **Practical Impact:** The mention of **Kaggle competitions** is very significant. Kaggle is a platform where data scientists compete to build the best predictive models. Ensemble methods, particularly Gradient Boosting (like XGBoost, LightGBM, CatBoost) and Random Forests, consistently form the basis of winning solutions due to their superior performance. This success directly translates to widespread use in industry for tasks like

fraud detection, medical diagnosis, recommendation systems, and more, where high predictive accuracy is crucial.

This slide sets the stage for diving into the specifics of Bagging, Boosting, and Stacking.

---

Slide 5: Decision Tree - Pros and Cons (Pros)

This slide outlines the advantages (Pros) of Decision Trees:

- **Interpretability:** Trees are very easy to explain to people. (In fact, easier compared to linear regression).
- **Graphical Display:** Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- **Human-like Decision Making:** Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches.
- **Handles Qualitative Predictors:** Trees can easily handle qualitative predictors without the need to create dummy variables.

**Explanation and Additional Knowledge:**

Decision trees are fundamental building blocks in machine learning, and understanding their strengths helps explain why they are often chosen as the "weak learners" in powerful ensemble methods.

- **Interpretability (White Box Model):** This is arguably the biggest advantage of decision trees. You can literally trace the path from the root node to a leaf node and understand the exact set of rules that led to a particular prediction. For example, "If a customer's age is greater than 30 AND their income is greater than $50,000, THEN approve their loan." This transparency is invaluable in fields like finance, healthcare, or legal contexts where understanding *why* a decision was made is as important as the decision itself. In contrast, many complex "black box" models (like deep neural networks) are hard to interpret.
- **Graphical Representation:** The visual, flowchart-like structure of a decision tree makes it very intuitive and easy to explain to stakeholders who might not have a technical background. A small, well-pruned tree can be an excellent communication tool.
- **Human-like Thought Process:** Our own decision-making often involves a series of sequential questions and conditions. Decision trees mimic this hierarchical, conditional reasoning, making them feel very natural and easy for humans to relate to.
- **Handles Mixed Data Types (Qualitative/Categorical Predictors):** Unlike some models (e.g., linear regression, which requires numerical inputs), decision trees can directly work with both numerical features (like age, income) and categorical features (like gender, city, product type) without requiring extensive preprocessing like one-hot encoding for categorical variables. The tree simply creates splits based on the categories.

While decision trees offer these compelling advantages, they also have significant drawbacks when used as single, standalone models, which the next slide will cover. These drawbacks are precisely what ensemble methods aim to address.

---

Slide 6: Decision Tree - Pros and Cons (Cons)

This slide outlines the disadvantages (Cons) of Decision Trees and how to overcome them:

- Trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.

- Trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.
- Decision Trees suffer from high variance.
- **So, How to overcome the disadvantages?**
    - By aggregating many decision trees, using methods like bagging, random forests, and boosting, the predictive performance of trees can be substantially improved.

**Explanation and Additional Knowledge:**

This slide is extremely important as it directly explains *why* ensemble methods are needed, especially for decision trees. It highlights the inherent weaknesses that ensemble learning is designed to overcome.

- **Lower Predictive Accuracy (compared to other complex models):** While single decision trees are interpretable, they often struggle to achieve the highest predictive accuracy. Simple trees might oversimplify complex relationships (high bias), while very deep trees, although having low bias on the training data, tend to overfit and perform poorly on new data (high variance).
- **Non-Robustness / Instability:** This is a major issue. A tiny alteration in the training data—perhaps adding or removing a few data points, or a slight change in the value of a feature for one observation—can lead to a drastically different tree structure and prediction. This makes a single decision tree model quite unreliable and sensitive to noise in real-world scenarios. It lacks stability.
- **High Variance:** This is the most significant statistical drawback and the primary reason why bagging and random forests were developed. A single decision tree, especially one grown deep without pruning, is highly sensitive to the specific training data it sees. If you train two deep trees on slightly different subsets of the same data, they will very likely produce very different trees and different predictions. This "memorization" of the training data means they don't generalize well to unseen data. This is the essence of high variance – the model's performance fluctuates significantly with changes in the training data.
- **How to overcome these disadvantages? (The Solution):**

  The slide explicitly states that the solution lies in **aggregating many decision trees** using ensemble methods. This is the core message of the entire presentation:
    - **Bagging:** Specifically designed to reduce variance by training multiple trees on bootstrapped (resampled) versions of the data and then averaging their predictions. Each tree might be high variance, but averaging them cancels out individual errors.
    - **Random Forests:** An advanced form of bagging that further reduces variance (and improves accuracy) by not only bootstrapping the data but also randomly selecting a subset of features at each split point, making the individual trees less correlated.
    - **Boosting:** Primarily aims to reduce bias by building trees sequentially, with each new tree trying to correct the errors of the preceding ones.

By combining many "weak" (often high-variance or high-bias depending on the method) decision trees, these ensemble methods leverage the "wisdom of crowds" to create a "strong" model that is much more accurate, robust, and has lower variance and/or bias than any single decision tree.

---

Slide 7: Bootstrap Aggregation - Bagging

This slide introduces Bootstrap Aggregation, or Bagging, as a general-purpose procedure for reducing the variance of a machine learning method.

- Decision Trees suffer from High Variance – i.e if the training data is split into two parts at random, and fit a decision tree to both halves, the results obtained could be quite different.
- **Purpose:** Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a machine learning method.
- **Method:**
    - In this approach we generate B different bootstrapped training data sets from the given single training dataset.
    - Then train the model on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$.
    - Finally average all the predictions, to obtain $\hat{f}*{bag}(x)=*frac{1}{B}*\sum*{b=1}^{B}\hat{f}^{*b}(x)$.

**Explanation and Additional Knowledge:**

Bagging is the first ensemble method we'll delve into, and its primary goal is to address the problem of high variance, particularly common in models like deep decision trees.

- **Understanding "High Variance" in Decision Trees (Recap):** The first point reiterates the instability of decision trees. Imagine you have 100 data points. If you randomly select 50 of them and train a deep decision tree, and then randomly select another 50 and train a second deep decision tree, these two trees could look drastically different and produce different predictions. This sensitivity to the specific training data is high variance.
- **The "Bootstrap" Component:**
    - **Bootstrapping:** This is a fundamental statistical technique. From your original training dataset (let's say it has N observations), you create new datasets by *sampling with replacement*. "With replacement" means that after you pick a data point, you put it back into the pool, so it can potentially be picked again. Each bootstrapped dataset will also have N observations. Because of sampling with replacement, each bootstrapped dataset will be slightly different from the original and from each other. Some original observations will appear multiple times, while others won't appear at all (these are the "out-of-bag" observations we'll discuss later).
- **The "Aggregation" Component:**
    - You generate a large number, B (e.g., 100, 500, or even more), of these different bootstrapped training datasets.
    - For each of these B datasets, you train an *independent* base model ($\hat{f}^{*b}(x)$). For bagging, these base models are often deep, unpruned decision trees (which inherently have high variance and low bias).
    - **Combining Predictions:**
        - For **regression tasks** (predicting a continuous number), you simply average the predictions from all B individual models to get the final bagging prediction ($\hat{f}_{bag}(x)$).
        - For **classification tasks** (predicting a category), you typically use a majority vote. The class predicted by the most individual models becomes the final prediction.
- **Why does it reduce variance?** By averaging or taking a majority vote of many independent (or nearly independent) models, the errors and sensitivities of individual

models tend to cancel each other out. If one tree makes a random error in one direction, another tree might make an error in the opposite direction, and their average will be closer to the true value. This process significantly reduces the overall variance of the combined ensemble model without substantially increasing its bias. It's like having multiple slightly flawed instruments measuring the same thing; averaging their readings often provides a more accurate result than any single reading.

---

Slide 8: Bagging (cont...)

This slide continues to explain Bagging, specifically detailing its application to regression and classification, and its general effectiveness.

- Regression - Construct B regression trees using B bootstrapped training sets, and average the resulting predictions.
- These trees are grown deep, and are not pruned.
- Hence each individual tree has high variance, but low bias.
- Averaging these B trees reduces the variance.
- Classification – After the trees are modelled, for a given test observation, record the class predicted by each of the B trees, and take a majority vote for predicting the label of the test observation.
- Bagging can improve predictions for many regression methods, especially useful for decision trees.

**Explanation and Additional Knowledge:**

This slide solidifies our understanding of bagging's mechanics and why it's so effective, particularly with decision trees.

- **Deep, Unpruned Trees in Bagging:** This is a critical point. When using bagging with decision trees, you intentionally grow each individual tree *very deep* and **do not prune** them.
    - **High Variance, Low Bias (Individual Tree):** A deep, unpruned decision tree has the capacity to perfectly fit (or even overfit) its specific bootstrapped training data. This means it has *low bias* because it's not making overly simplistic assumptions about the data – it can capture complex relationships. However, because it's so perfectly tailored to that specific sample, it has *very high variance*; its exact structure and predictions would change dramatically if given a slightly different training set.
    - **The Power of Averaging:** The "magic" of bagging is that by combining many such high-variance, low-bias trees, the high variance component is drastically reduced. Each tree might have its own unique errors or sensitivities due to the random nature of its bootstrapped sample, but these errors tend to be somewhat random and cancel each other out when averaged or voted upon. The desirable low bias, however, is preserved in the ensemble. The result is a powerful ensemble model that has both low bias and low variance.
- **Regression vs. Classification (Combining Predictions):**
    - **Regression:** If your task is to predict a continuous numerical value (e.g., house price), you simply sum up the predictions from all B trees and divide by B to get the average.
    - **Classification:** If your task is to predict a categorical label (e.g., "spam" or "not spam"), you count how many trees predicted each class and choose the class with

the most votes (a majority vote). For instance, if 7 out of 10 trees predict "spam," the final ensemble prediction is "spam."

- **General Applicability:** While bagging is particularly effective for decision trees due to their inherent high variance, it's a general procedure. It can be applied to other types of base learners as well. However, the benefits might be less pronounced if the base learner itself doesn't suffer from high variance. Nevertheless, its most popular and impactful application is undoubtedly with decision trees, forming the foundation of Random Forests.

---

Slide 9: Out-of-Bag Error Estimation

This slide explains a clever way to estimate the test error for bagged models without needing cross-validation or a separate validation set.

- Estimation of Test error (without the need to perform cross-validation or the validation set approach).
- Each bagged tree makes use of around two-thirds of the observations.(Can be proved)
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the **out-of-bag (OOB) observations**.
- So, Predict the response for the ith observation using each of the trees in which that observation was OOB. (This will be around B/3 predictions for the ith observation).
- In order to obtain a single prediction for the ith observation, we can average these predicted responses (For regression) or can take a majority vote (For classification).

**Explanation and Additional Knowledge:**

Out-of-Bag (OOB) error estimation is a fantastic and efficient feature of bagging. It allows you to get a reliable estimate of your model's performance on unseen data without the computational cost of cross-validation or the need to set aside a separate validation set.

- **The "Two-Thirds" Rule (Mathematical Basis):** When you perform bootstrap sampling (sampling N observations with replacement from an original dataset of size N), it can be mathematically proven that, on average, each bootstrap sample will contain approximately 63.2% of the unique original observations. This means roughly **one-third (about 36.8%)** of the original observations will *not* be present in any given bootstrap sample.
- **What are OOB Observations?** For each individual decision tree in your bagging ensemble, the observations from the *original* training dataset that were *not* included in its specific bootstrapped training set are called **Out-of-Bag (OOB) observations**. Crucially, these OOB observations have never been "seen" by that particular tree during its training. They act like a fresh, independent validation set for that specific tree.
- **How OOB Error is Calculated (Step-by-Step):**
    1. **For each observation i in your *original* training data:**
        - Identify all the individual bagged trees for which observation i was an OOB observation (i.e., it was not used to train that specific tree).
    2. **Make Predictions:** Use these "OOB trees" to predict the response for observation i. Since each observation i will be OOB for approximately B/3 trees, you'll get roughly B/3 predictions for X_i.
    3. **Combine Predictions:**
        - For **regression**, average these B/3 predictions for X_i to get a single OOB prediction for X_i.
        - For **classification**, take a majority vote among these B/3 predictions to get a

single OOB prediction for X_i.

4. **Calculate Error:** Compare this combined OOB prediction for X_i with its actual true label y_i.
5. **Aggregate:** Repeat this process for all observations i in your original training data. The overall error rate (e.g., mean squared error for regression, misclassification rate for classification) calculated from these OOB predictions is the **OOB error estimate**.

- **Major Benefit:** The OOB error estimate is a highly reliable estimate of the model's test error (how it would perform on truly unseen data) because each prediction for an observation X_i is made by trees that *never saw X_i during their training*. This means you get a free and unbiased estimate of your model's generalization performance, saving you the effort and computational cost of explicit cross-validation or reserving a separate validation set. It's like having built-in cross-validation!

---

Slide 10: Random Forests

This slide introduces Random Forests, an enhancement over bagged trees.

- Random forests decorrelate the trees to provide an improvement over bagged tress.
- As in bagging, a number of decision trees are build on bootstrapped training samples.
- **But when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.**
- The split is allowed to use only one of those m predictors.
- A fresh sample of m predictors is taken at each split.
- Typically we choose $m \approx \sqrt{p}$.

**Explanation and Additional Knowledge:**

Random Forests are one of the most powerful and widely used ensemble methods, building directly on the concept of bagging to address a crucial limitation.

- **The Problem with Standard Bagging of Trees:** While bagging effectively reduces variance, it still has a subtle flaw. If there's one or a few very strong predictor variables in your dataset, many (or even most) of the individual bootstrapped trees will likely choose to split on that same strong predictor near the top of their structure. This makes the individual trees in the ensemble highly correlated. If the trees are making similar types of predictions and errors, then averaging them doesn't reduce the overall variance as effectively as averaging truly *uncorrelated* trees.
- **How Random Forests Decorrelate Trees (The Key Difference):** This is the ingenious modification that makes Random Forests so powerful.
    - **Feature Subsampling (Random Subspace Method):** Instead of considering *all* p available features at each split point in a tree, a random subset of m features is selected. The tree is then *forced* to choose the best split *only from these m randomly selected features*. This means the strongest predictor might not even be available for consideration at some splits.
    - **Fresh Sample at Each Split:** This random selection of m features is done *at every single split* in *every single tree*. So, even if a tree used feature X for an earlier split, it might not see feature X as an option for a later split, further ensuring diversity among the trees.
- **Why $m \approx \sqrt{p}$?** This is a common heuristic (a rule of thumb) for choosing

the number of features m to consider at each split:
    - For **classification tasks**, $m = \sqrt{p}$ (the square root of the total number of predictors) is a popular and often effective choice. It's small enough to introduce decorrelation but large enough to allow for good splits.
    - For **regression tasks**, $m = p/3$ (one-third of the total number of predictors) is often used.
    - The optimal m can be tuned as a hyperparameter for your specific dataset.
- **Benefits of Decorrelation:** By forcing the trees to consider only a random subset of features at each split, Random Forests make the individual trees much more diverse and less correlated with each other. This leads to a greater reduction in the overall variance of the ensemble and often a significant improvement in predictive accuracy compared to simple bagged trees.
- **Summary of Random Forests:** They introduce two levels of randomness:
    1. **Bootstrap Sampling:** Randomly sampling data points (like bagging).
    2. **Feature Subsampling:** Randomly sampling features at each split.

This dual randomness makes Random Forests highly robust, accurate, and resistant to overfitting, making them a very popular choice for many machine learning tasks.

---

Slide 11: Boosting

This slide introduces Boosting, another ensemble technique, contrasting it with bagging.

- In Boosting, trees are grown sequentially. i.e each tree is grown using information from previously grown trees.
- Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set.
- Given the current model, we fit a decision tree to the residuals from the model. i.e we fit a tree using the current residuals, rather than the outcome Y as the response.
- This new decision tree is then added into the fitted function in order to update the residuals.

**Explanation and Additional Knowledge:**

Boosting takes a fundamentally different approach compared to bagging and random forests. While bagging aims to reduce variance by combining many *independent* models, boosting aims to primarily reduce **bias** by building models *sequentially*, where each new model learns from and tries to correct the mistakes of the models that came before it.

- **Sequential Learning (Iterative Improvement):** This is the defining characteristic of boosting. Instead of training all models in parallel, boosting trains them one after another. The training of each new model is influenced by the performance of the previously trained models in the ensemble.
- **Learning from Mistakes (Focus on Residuals):**
    - Imagine you train a first, simple model (a weak learner). It will make some errors – differences between its predictions and the actual target values. These errors are called "residuals" (observed value - predicted value).
    - Instead of training the next weak learner on the *original target variable (Y)*, you train it to predict these *residuals* from the previous step. The idea is that this new model will learn patterns in the *errors* and effectively correct the mistakes of the earlier models.
    - Then, the (scaled-down) prediction of this new model is added to the existing

ensemble model, which updates the overall prediction.
    - This process continues: new residuals are calculated based on the updated ensemble's prediction, and the next tree is trained to predict *those* remaining residuals.
- **No Bootstrap Sampling (Typically):** Unlike bagging, boosting typically uses the *entire* original training dataset for each sequential model. However, the "focus" or "version" of the dataset changes. In some boosting algorithms like AdaBoost, data points are re-weighted, while in others like Gradient Boosting (which the algorithm on Slide 13 is closer to), the target variable for each new tree is the residuals from the current ensemble.
- **Goal: Reduce Bias:** By iteratively focusing on and correcting errors, boosting gradually builds a more complex and accurate overall model that can capture intricate patterns that individual simple models might miss. This systematic error correction is highly effective at reducing the bias of the ensemble model. While it primarily targets bias, it often results in models with surprisingly low variance as well, provided careful tuning and regularization.

Think of it like repeatedly pointing a spotlight at the areas where the model is currently performing poorly, and then training a new specialist model to address those specific areas.

---

Slide 12: Boosting (cont...)

This slide continues the explanation of Boosting, focusing on the characteristics of the individual trees and the role of the shrinkage parameter.

- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve f in areas where it does not perform well.
- The shrinkage parameter $\lambda$ slows the process down even further, allowing more and different shaped trees to attack the residuals.
- In general, machine learning approaches that learn slowly tend to perform well.

**Explanation and Additional Knowledge:**

This slide elaborates on the "slow and steady wins the race" philosophy that is central to boosting's effectiveness.

- **Small Trees (Weak Learners):** In boosting, the individual "base learners" are deliberately kept simple and shallow. They are typically short decision trees with only a few splits (e.g., d splits, leading to d+1 terminal nodes). These are purposely "weak" because the goal is not for one tree to solve the entire problem, but for many simple trees, each contributing a small, focused correction, to collectively arrive at a powerful solution.
    - d: This parameter directly controls the depth or complexity of each individual tree. A smaller d means simpler, weaker trees.
- **Iterative Improvement on Residuals:** By fitting these small trees to the residuals (the errors remaining from the previous iteration), the overall model gradually focuses its efforts on the data points that are currently mispredicted or poorly estimated. It's like finding the "hardest" examples for the current ensemble and then training a specialized, simple model specifically to address those remaining errors.
- **Shrinkage Parameter $\lambda$ (Learning Rate):** This is a critical hyperparameter in boosting, often called the learning rate.
    - When a new tree's predictions are added to the overall ensemble, its contribution is

scaled down by this $\lambda$ parameter. $\lambda$ is a small positive number (e.g., 0.1, 0.01, 0.001).
    - **Effect:** A small $\lambda$ means that each new tree makes only a very modest, cautious contribution to the overall model. This prevents the model from quickly overfitting to the training data. It forces the algorithm to learn slowly, taking many small, incremental steps rather than large, potentially erroneous leaps.
    - **Why "Slow Learning" is Good:** This "slow and steady" approach acts as a form of **regularization**. It makes boosting models very robust, allowing the ensemble to explore the error space more thoroughly and leading to better generalization performance on unseen data. It significantly reduces the risk of overfitting, even though boosting primarily targets bias.
- **Analogy:** Imagine trying to hit a target. Instead of one person taking a huge, powerful shot (a deep, single tree), boosting is like many people taking turns, each making a tiny, precise adjustment based on how far off the last attempt was. The $\lambda$ parameter dictates how tiny each adjustment can be, ensuring they eventually get very close to the target without wildly overshooting.

This gradual, regulated learning process is what makes boosting algorithms so incredibly effective and accurate.

---

Slide 13: Boosting (cont...)

This slide provides a more formal, step-by-step algorithm for Boosting.

1. Set $\hat{f}(x)=0$ and $r_{i}=y_{i}$ for all i in the training set.
2. For b=1,2,...,B, repeat:

   (a) Fit a tree $\hat{f}^{b}$ with d splits (d+1 terminal nodes) to the training data (X, r).

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree: $\hat{f}(x)\leftarrow\hat{f}(x)+\lambda\hat{f}^{b}(x).$

   (c) Update the residuals, $r_{i}\leftarrow r_{i}-\lambda\hat{f}^{b}(x_{i}) .$
3. Output the boosted model, $\hat{f}(x)=\sum_{b=1}^{B}\lambda\hat{f}^{b}(x).$

**Explanation and Additional Knowledge:**

This algorithm outlines the generalized boosting process, which is the foundation for algorithms like Gradient Boosting Machines (GBMs). It formalizes the iterative error-correction mechanism.

- **Step 1: Initialization:**
    - $\hat{f}(x)=0$: The ensemble model, which we are building, starts with an initial prediction of zero for all observations. It essentially has learned nothing yet.
    - $r_{i}=y_{i}$: The initial residuals ($r_i$) are set to the actual target values ($y_i$) for each observation. This is because the "error" of our initial zero model is simply the target itself ($y_i - \hat{f}(x_i) = y_i - 0 = y_i$).
- **Step 2: Iterative Learning (The Loop for B Trees):** This loop runs for B iterations (where B is the total number of trees in the ensemble). In each iteration b:
    - **(a) Fit a tree to residuals:** A new weak learner ($\hat{f}^{b}$) is trained. Crucially, it's not trained on the original target variable y, but on the *current residuals* (r). This tree learns to predict the remaining "error" from the previous iteration of the

ensemble. The d splits ensure it's a simple, shallow tree (a weak learner).

Sources:

- ensemble_learning.pdf