

Machine Learning

Video 106:

Gradient Boosting Explained | How Gradient Boosting Works?

Gradient Boosting is a machine learning technique that builds predictive models by sequentially adding weak learners (typically decision trees), each correcting the errors of the previous ones. It minimizes loss using gradient descent, improving accuracy. Popular implementations include XGBoost, LightGBM, and CatBoost, widely used for classification and regression tasks.

Gradient Boosting works by building an ensemble of weak learners (usually decision trees) sequentially, where each new model corrects the errors of the previous ones. Here's a step-by-step breakdown:

1. **Initialize the Model**
 - Start with a simple model, often a weak predictor like a small decision tree. This serves as the initial guess.
2. **Compute Residuals (Errors)**
 - Calculate the difference (residuals) between the actual values and the predictions from the previous model.
3. **Train a New Model on Residuals**
 - Fit a new weak learner to predict these residual errors instead of the actual target values.
4. **Update the Model**
 - Adjust predictions by adding the new model's weighted output to the previous predictions.
 - The weight (learning rate) controls how much influence each new model has.
5. **Repeat the Process**
 - Continue adding new models iteratively, refining predictions at each step, until a stopping criterion (e.g., number of iterations or minimal error improvement) is met.
6. **Final Prediction**
 - The final model is the weighted sum of all weak learners, forming a strong ensemble predictor.

Example:

Code link:




<https://colab.research.google.com/drive/17Fdl2m05wVdkc240tcDlfsS1a9ZPz8rp?usp=sharing>

Video 107:

Gradient Boosting Regression Part 2 | Mathematics of Gradient Boosting

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iter

1. Initialize $f_0(x) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$.
2. For $m = 1$ to M :
 - (a) For $i = 1, 2, \dots, N$ compute
$$r_{im} = - \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}.$$
 - (b) Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \dots, J_m$.
 - (c) For $j = 1, 2, \dots, J_m$ compute
$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$
 - (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.

8 / 56:41 • Algorithm >   

What is Additive modelling?

Additive modelling in machine learning is a technique where the final prediction is the sum of multiple simpler models or functions. Each component captures different patterns in the data, improving flexibility and interpretability. Examples include Generalized Additive Models (GAMs) and boosting methods like Gradient Boosting Machines (GBMs).

Video 108:

Gradient Boosting for Classification | Geometric Intuition | CampusX

Gradient boosting for classification is an ensemble learning technique that builds multiple decision trees sequentially. Each tree corrects the errors of the previous one by minimizing a loss function using gradient descent. This improves model accuracy and reduces bias. Popular implementations include XGBoost, LightGBM, and CatBoost. It's widely used in machine learning.

Example:

Code link:

<https://colab.research.google.com/drive/13p46lFhg3h6BlDjxUcfXPco13jI0CV6l?usp=sharing>

<https://colab.research.google.com/drive/15G44KBuSgHs7hdSluhwh-LR7Qur4xRyD?usp=sharing>

Video 109:

Stacking and Blending Ensembles

Stacking and blending are ensemble learning techniques that combine multiple models to improve predictive performance. Stacking trains a meta-model on base model predictions, learning optimal weightings. Blending uses a holdout set for base models and combines predictions using a simple method (e.g., weighted average). Both reduce overfitting and enhance accuracy.

Bagging, boosting, stacking, and blending are all ensemble learning techniques, but they differ in approach:

1. Bagging (Bootstrap Aggregating)

- Trains multiple models independently on bootstrapped subsets of data.
- Combines predictions (e.g., averaging for regression, voting for classification).
- Reduces variance (e.g., Random Forest).

2. Boosting

- Trains models sequentially, where each model corrects the errors of the previous one.
- Models are weighted based on performance (e.g., AdaBoost, XGBoost).
- Reduces bias and improves weak learners.

3. Stacking

- Trains multiple models and combines their predictions using a meta-model.
- Learns optimal weightings for different models.
- More complex but powerful.

4. Blending

- Similar to stacking but uses a validation set for base models.

- Combines predictions using a simple method (e.g., weighted average).
- Less prone to overfitting than stacking.

Key Differences:

- **Bagging** and **boosting** focus on improving individual weak learners (boosting sequentially, bagging in parallel).
- **Stacking** and **blending** focus on intelligently combining multiple models for better generalization.

Problems in Stacking and Blending:

1. **Overfitting** – The meta-model in stacking may overfit if it learns noise from base models instead of generalizable patterns.
2. **Computational Complexity** – Training multiple models and a meta-model requires significant computational resources and time.
3. **Data Leakage** – Improper validation set splitting in blending can lead to data leakage, resulting in overly optimistic performance estimates.
4. **Difficult Optimization** – Choosing the right base models, meta-model, and blending strategy requires careful tuning and experimentation.
5. **Interpretability** – Stacking and blending create complex models that are harder to interpret compared to simpler ensembles like bagging and boosting.

Solutions for Problems in Stacking and Blending:

1. **Overfitting**
 - ✓ Use cross-validation to train the meta-model on out-of-sample predictions.
 - ✓ Regularize the meta-model (L1/L2 regularization, dropout, etc.).
 - ✓ Use diverse base models to avoid learning similar patterns.
2. **Computational Complexity**
 - ✓ Reduce the number of base models or use simpler models.
 - ✓ Use efficient algorithms (e.g., LightGBM, XGBoost).
 - ✓ Optimize code with parallel processing and GPU acceleration.
3. **Data Leakage**
 - ✓ Ensure proper train-validation split for blending.
 - ✓ Use strict separation between training and validation data.
 - ✓ Validate the final ensemble on an unseen test set.
4. **Difficult Optimization**
 - ✓ Experiment with different base models and meta-models.
 - ✓ Use automated model selection techniques (e.g., Grid Search, Bayesian Optimization).
 - ✓ Start with simple averaging before applying complex stacking.
5. **Interpretability**
 - ✓ Use feature importance analysis in base models.
 - ✓ Choose simpler meta-models (e.g., linear regression instead of deep learning).
 - ✓ Visualize model contributions to understand decision-making.

Example:

Code link:

<https://github.com/campusx-official/100-days-of-machine-learning/blob/main/day68-stacking-and-blending/Untitled.ipynb>

Video 110:

Agglomerative Hierarchical Clustering | Python Code Example

Problems with K-Means Clustering & Solutions

1. Choosing the Optimal Number of Clusters (K)

Problem: K-Means requires predefining the number of clusters, which can be difficult.

Solution: Use techniques like the Elbow Method, Silhouette Score, or Gap Statistics to determine the best K.

2. Sensitivity to Initial Centroid Selection

Problem: Poor initial centroids can lead to suboptimal clustering.

Solution: Use K-Means++ for better centroid initialization instead of random selection.

3. Sensitivity to Outliers

Problem: K-Means minimizes squared distances, so outliers can heavily influence centroids.

Solution: Remove or scale outliers, use K-Medoids (which uses actual data points as centroids) instead of K-Means.

4. Assumes Spherical Clusters

Problem: K-Means assumes clusters are convex and evenly sized, failing with non-spherical clusters.

Solution: Use DBSCAN or Gaussian Mixture Models (GMM) for non-spherical data.

5. Unequal Cluster Sizes & Density

Problem: K-Means struggles with clusters of varying sizes and densities.

Solution: Use hierarchical clustering or GMM, which can model different cluster shapes.

6. Computational Complexity

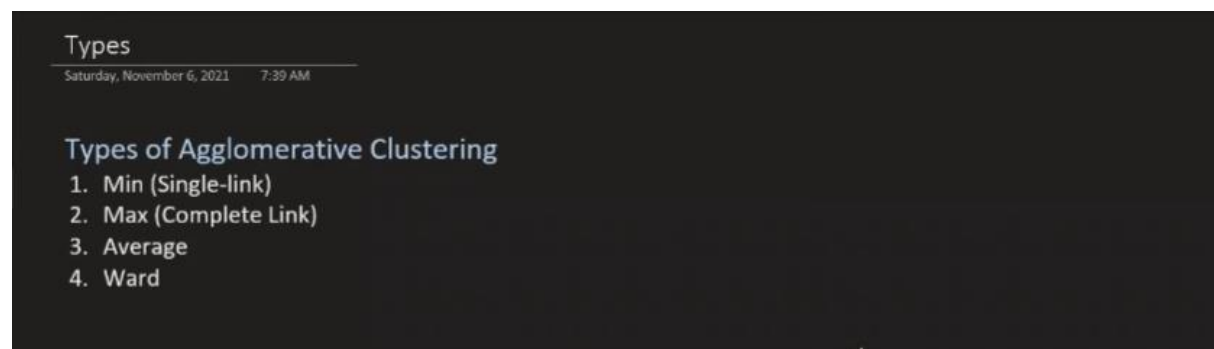
Problem: K-Means can be slow on large datasets, especially with high-dimensional data.

Solution: Use Mini-Batch K-Means for faster training on big data.

Hierarchical clustering is an unsupervised learning method that builds a hierarchy of clusters. It can be **agglomerative** (bottom-up, merging clusters) or **divisive** (top-down, splitting clusters). The results are visualized using a **dendrogram**, allowing flexible cluster selection. Unlike K-Means, it doesn't require specifying the number of clusters beforehand.

Types:

1. Agglomerative clustering
2. Divisive clustering



Example:

Code Link:

<https://github.com/campusx-official/agglomerative-hierarchical-clustering-demo/blob/main/agglomerative-clustering.ipynb>