BREAK THROUGH TECH

Machine Learning Foundations

Lab 6

Week of July 7

BREAK THROUGH TECH

Icebreaker: Two Truths and a Lie

Two Truths and a Lie

Rules:

- Think of two truths and one lie about yourself.
- Share with your group and let them guess the lie.
- Reveal the lie and explain your truths.

Two Truths and a Lie: Breakouts

Breakout Instructions:

- You will be in groups of about 10.
- Each person has about one minute to share.
- Listen and guess attentively.
- We'll come back and share fun/surprising facts you learned about your peers.

Two Truths and a Lie: Share Out

What’s something unexpected you learned about a peer?

BREAK
THROUGH
TECH

Week 6 Concept Overview + Q&A

Q+A on Unsupervised Learning

Do you have any questions about unsupervised learning or the weekly assignment?

Concept Overview

This week covered a number of topics. To refresh your memory, here is what you've completed:

- Explore the bias-variance tradeoff
- Improve model performance with ensemble methods
- Understand the mechanics of three ensemble methods: stacking, random forests and gradient boosted decision trees
- Explore unsupervised learning
- Implement unsupervised clustering

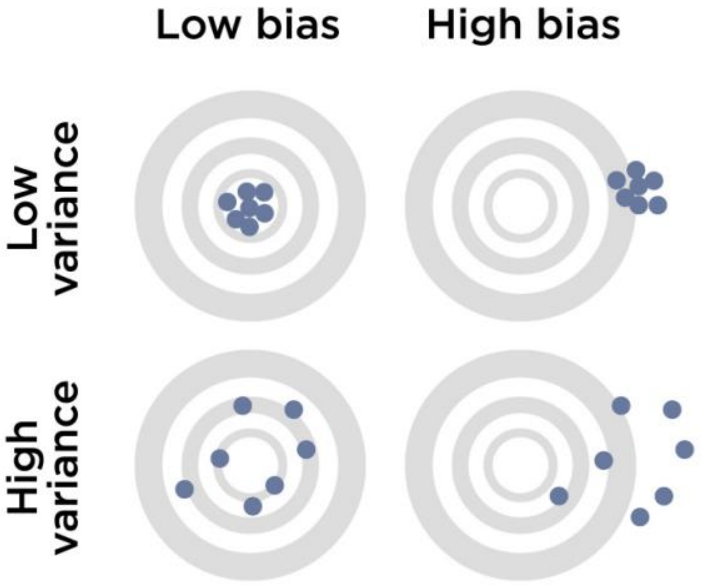
Improve Model Performance with Ensemble Methods: Bias and Variance

The graph below exemplifies data with high/low variance and high/low bias as "darts" thrown at a target. The bullseye at the center of the target is the location of the perfect classifier on the testing data. The blue dots illustrate the darts, which represent classifiers trained on different training data sets.

High variance/low bias models perform well when performance is averaged over large data sets. In expectation they are close to the bullseye but can perform very differently on any two particular data sets. We say that these models are overfit; that they have learned to predict meaningless noise patterns in the data.

High bias/low variance settings lead to models that are very similar across different training data sets (the blue dots are close together); however, they are systematically off-target (i.e., they make wrong assumptions).

The worst case is **high bias and high variance**. The goal is to achieve a model with low bias and low variance.



Ensemble Methods

- **Stacking**: taking a weighted combination of the predictions of a total K different models

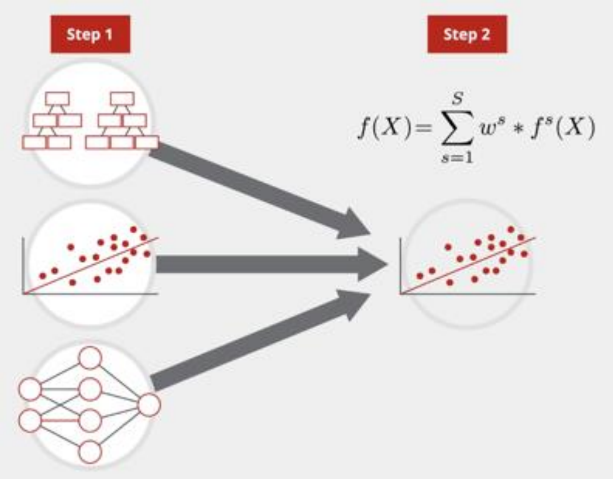
- **Bagging:** generating multiple models from the same data by taking bootstrapped samples and averaging the individual model predictions

- **Boosting:** iteratively building models by focusing on the cumulative error from prior iterations' predictions

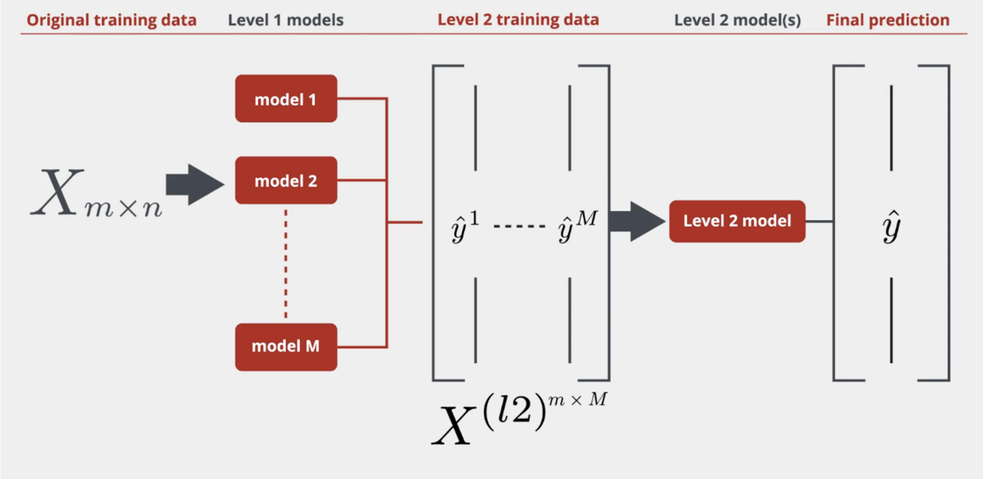
Stacking

Two approaches:

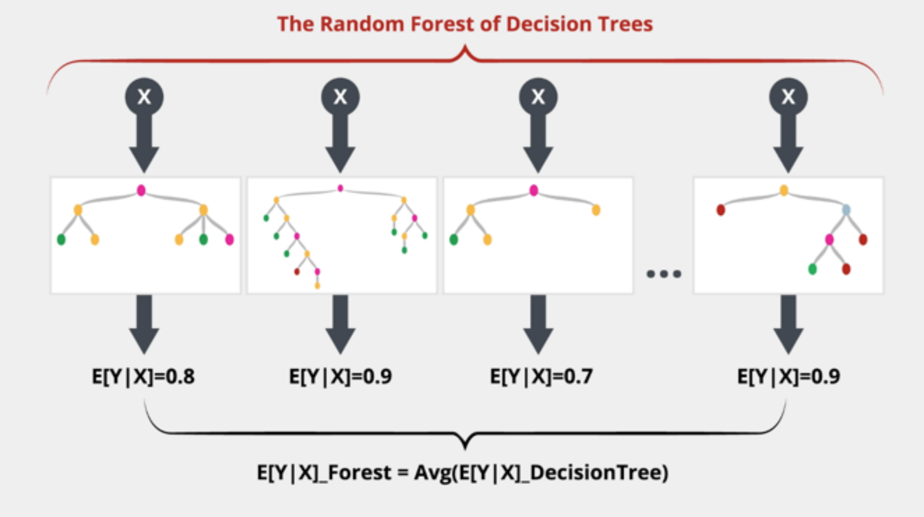
1.

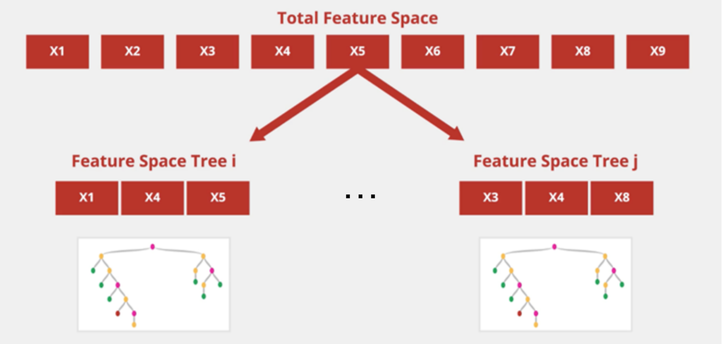


2.

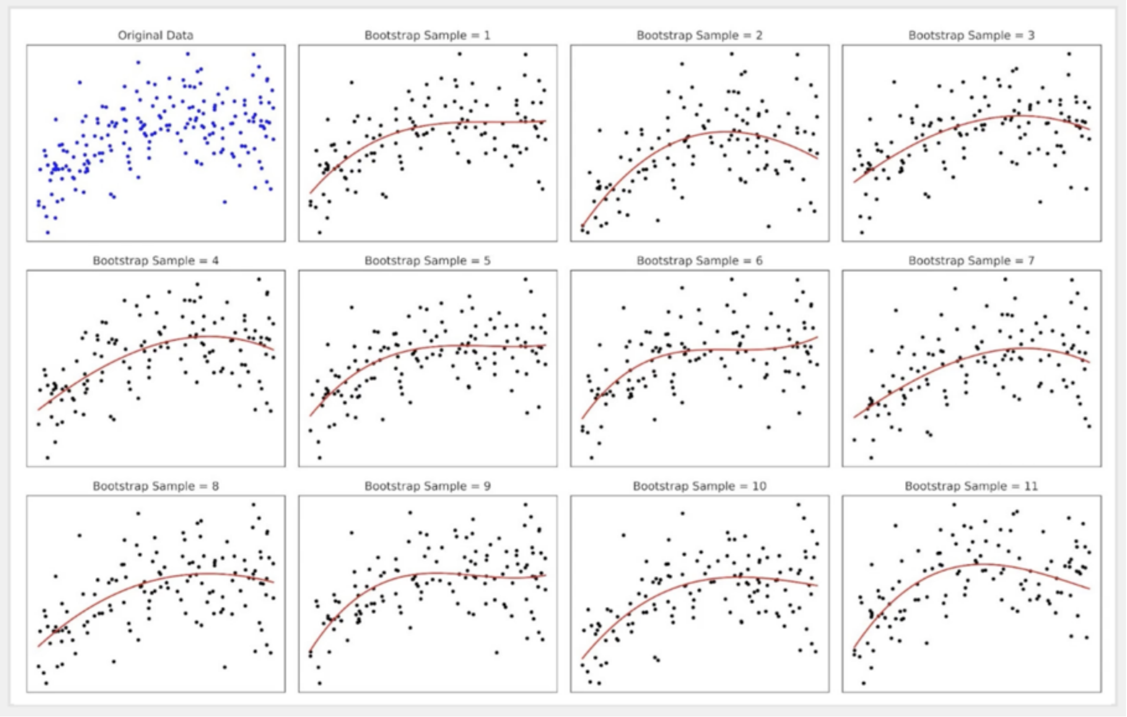


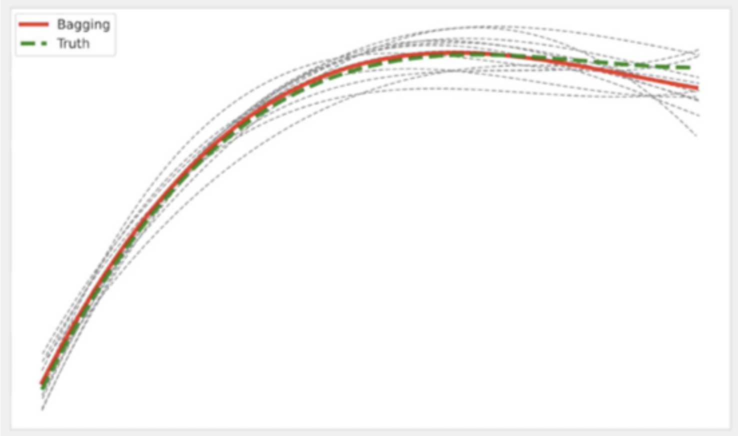
Random Forest


The Random Forest of Decision Trees
X
X
X
X
...
E[Y|X]=0.8
E[Y|X]=0.9
E[Y|X]=0.7
E[Y|X]=0.9
E[Y|X]_Forest = Avg(E[Y|X]_DecisionTree)

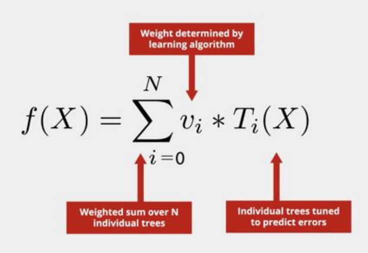


Random Forest: Bagging





Gradient Boosted Decision Trees



$$f(X) = \sum_{i=0}^{N} v_i * T_i(X)$$

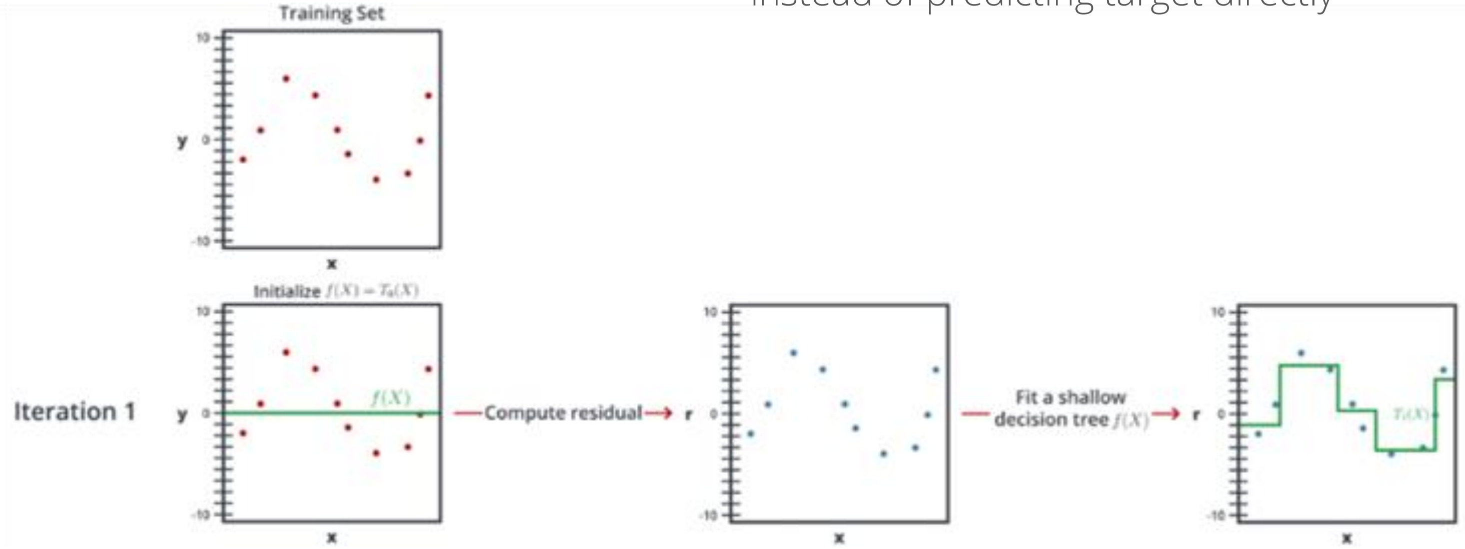
Weight determined by learning algorithm

Weighted sum over N individual trees

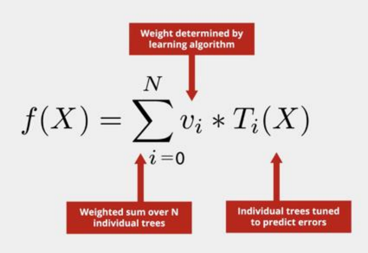
Individual trees tuned to predict errors

tune using residuals (error), not data points

new tree predicts error by previous model
instead of predicting target directly



Gradient Boosted Decision Trees



Weight determined by learning algorithm

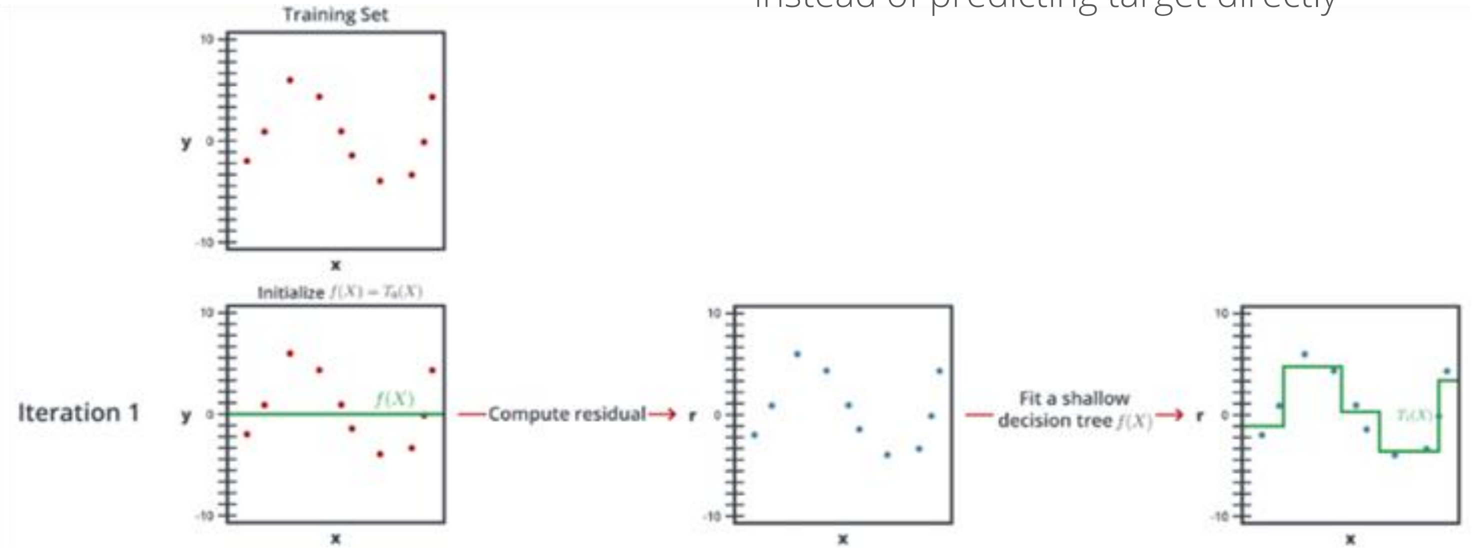
$$f(X) = \sum_{i=0}^{N} v_i * T_i(X)$$

Weighted sum over N individual trees

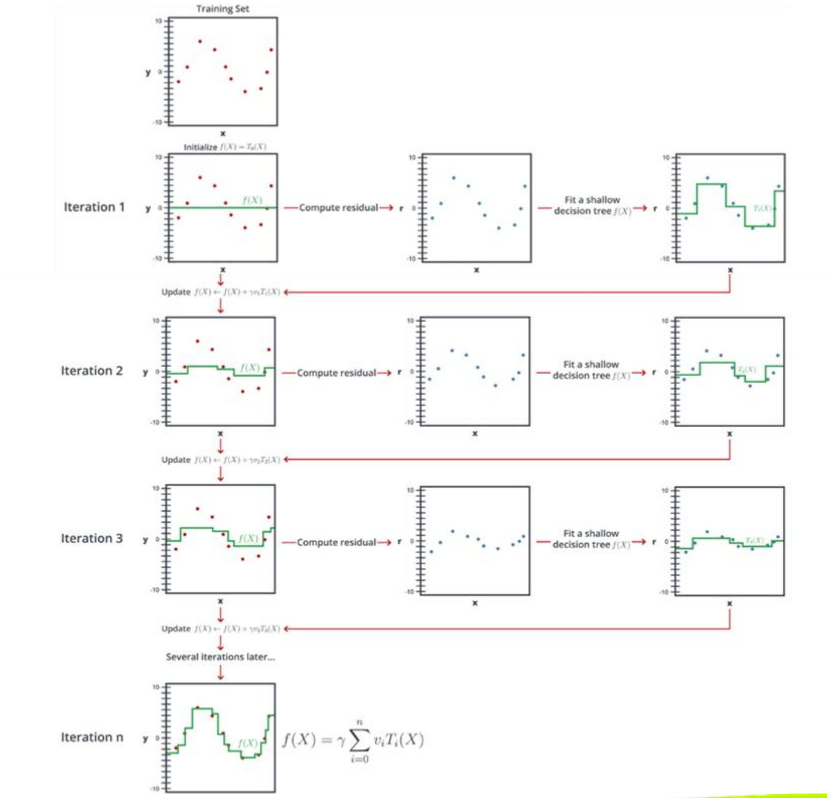
Individual trees tuned to predict errors

tune using residuals (error), not data points

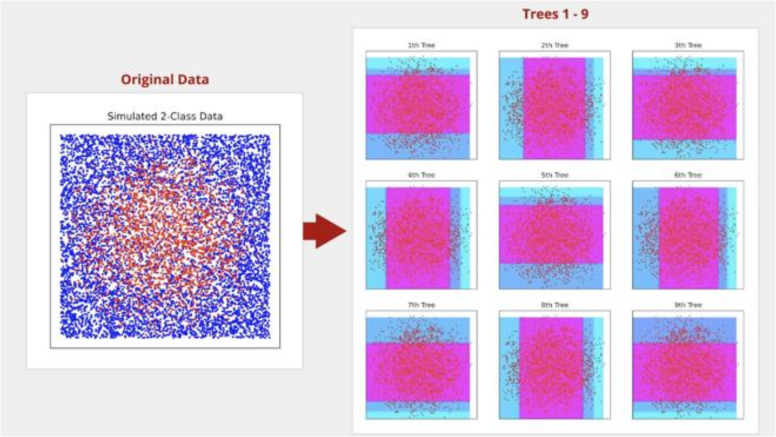
new tree predicts error by previous model
instead of predicting target directly

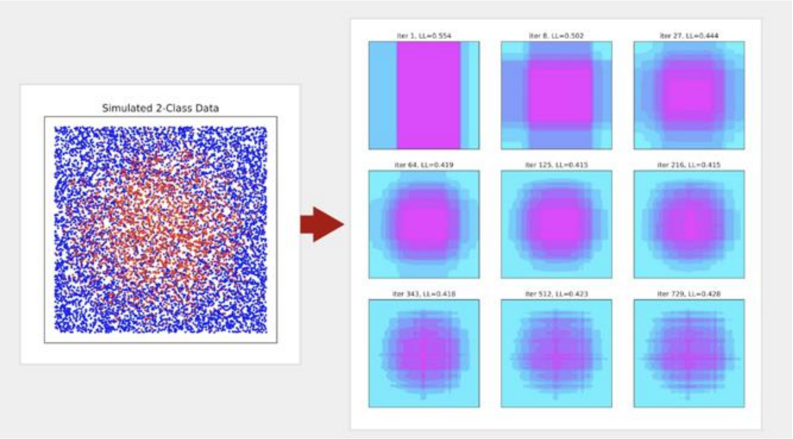


Gradient Boosted Decision Trees

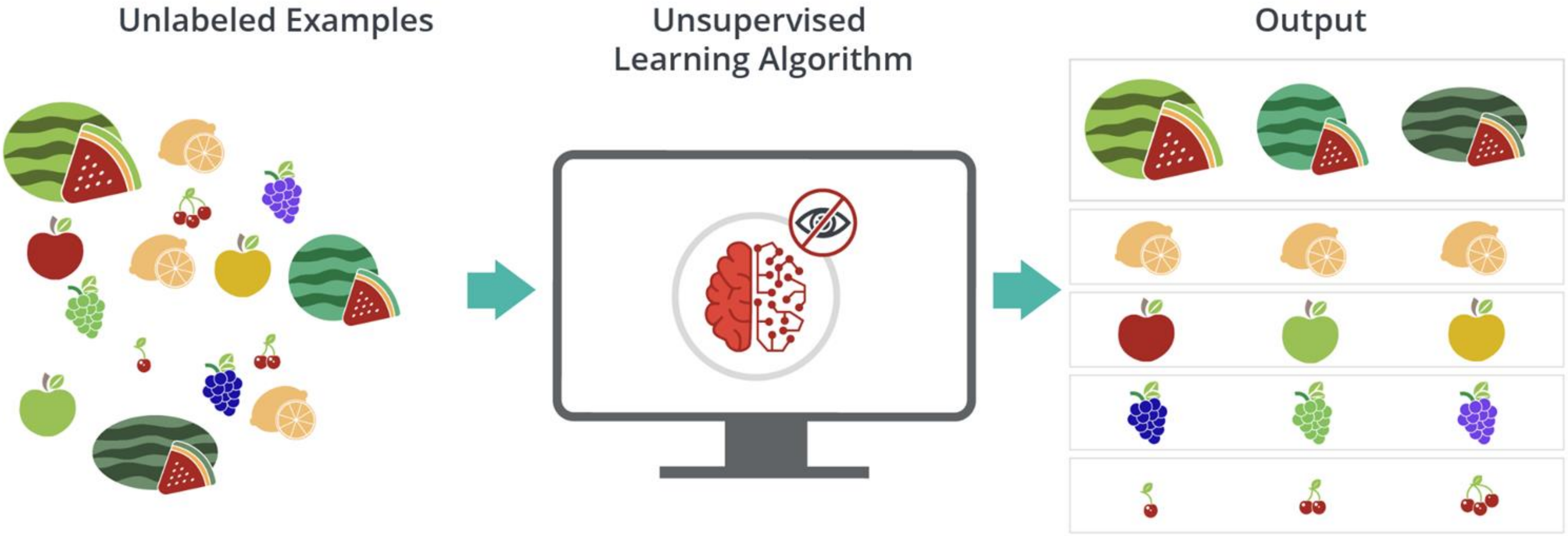

Training Set
y
x
10
0
-10
Initialize $f(X) = T_0(X)$
Iteration 1
$f(X)$
Compute residual
r
Fit a shallow decision tree $f(X)$
$T_1(X)$
Update $f(X) \leftarrow f(X) + \gamma v_1 T_1(X)$
Iteration 2
Compute residual
Fit a shallow decision tree $f(X)$
$T_2(X)$
Update $f(X) \leftarrow f(X) + \gamma v_2 T_2(X)$
Iteration 3
Compute residual
Fit a shallow decision tree $f(X)$
$T_3(X)$
Update $f(X) \leftarrow f(X) + \gamma v_3 T_3(X)$
Several iterations later...
Iteration n
$f(X) = \gamma \sum_{i=0}^{n} v_i T_i(X)$

Gradient Boosted Decision Trees





Unsupervised Learning: Clustering



Unlabeled Examples

Unsupervised Learning Algorithm

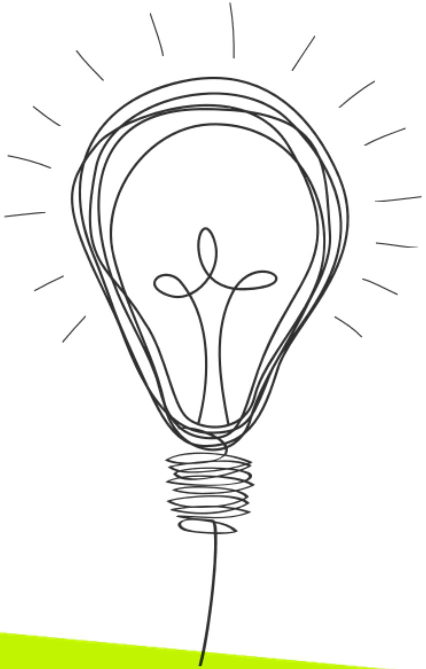
Output

Unsupervised Learning: Clustering Algorithms

- KMeans
- Hierarchical clustering

Questions & Answers

What questions do you have about the online content this week?



BREAK
THROUGH
TECH

Breakout Groups: Big Picture Questions

Big Picture Questions

You have 15 minutes to discuss the following questions within your breakout groups:

- Why does using an ensemble of models often perform better than a single model?
- What are some real-world applications of ensemble learning?
- What are some key differences and similarities between GBDT and Random Forest?
- When would it not be appropriate or optimal to use an ensemble model like Random Forest or GBDT?
- What are some disadvantages of using Decision Trees and how would you solve for them?
- Why is model stacking effective in improving performance?

Big Picture Responses

A. Why does using an ensemble of models often perform better than a single model ?

Using an ensemble of models often performs better than a single model for several reasons:

1. Reduction in Variance: Different models may make different errors on the same data points. By averaging or voting across multiple models, the overall variance of the predictions can be reduced, leading to more stable and reliable outcomes.
2. Reduction in Bias: Combining models that have different biases can lead to a more accurate overall model. While a single model might underfit or overfit the data, an ensemble of diverse models can balance out these biases.
3. Improved Generalization: Ensembles can generalize better to new, unseen data compared to individual models. By leveraging the strengths of multiple models, the ensemble can capture more nuances in the data.
4. Error Compensation: Different models might make different types of errors. An ensemble approach can compensate for the individual errors made by models, resulting in better overall performance.
5. Diverse Perspectives: Models trained on different subsets of data or using different algorithms bring diverse perspectives. This diversity helps in capturing various patterns and structures within the data that a single model might miss.
6. Robustness: Ensembles tend to be more robust to outliers and noise in the data. Since the final prediction is based on multiple models, the impact of noisy data points is often diminished.

Ensemble methods, such as bagging (e.g., Random Forest), boosting (e.g., Gradient Boosting), and stacking, exploit these benefits by combining the outputs of multiple models to produce superior predictive performance.

Big Picture Ques Responses– B.What are some real-world applications of ensemble learning ?

Ensemble learning is widely used across various domains due to its robustness and improved performance. Here are some real-world applications:

1. Healthcare and Medicine:
   - Disease Prediction and Diagnosis: Ensemble models can combine predictions from various algorithms to improve the accuracy of disease diagnosis and prognosis, such as in predicting the onset of diseases like diabetes or cancer.
   - Medical Imaging: In tasks like tumor detection in radiology images, ensembles help improve the accuracy by integrating multiple image processing models.
2. Finance:
   - Credit Scoring: Banks and financial institutions use ensemble methods to assess the creditworthiness of loan applicants by combining predictions from different scoring models.
   - Algorithmic Trading: Ensembles can forecast stock prices or identify trading opportunities by combining predictions from multiple models analyzing historical and real-time market data.
3. Marketing and Sales:
   - Customer Segmentation: Marketers use ensembles to segment customers more effectively by combining insights from different clustering and classification algorithms.
   - Churn Prediction: Predicting customer churn can be improved with ensembles, helping companies retain customers by identifying those at risk of leaving.
4. Natural Language Processing (NLP):
   - Sentiment Analysis: Ensembles can improve the accuracy of sentiment analysis by combining models trained on different datasets or using different techniques.
   - Machine Translation: Combining multiple translation models can produce more accurate and fluent translations.
5. Recommender Systems:
   - Personalized Recommendations: Online platforms like Netflix and Amazon use ensemble methods to combine various recommendation algorithms, leading to better product or content recommendations.
6. Image and Speech Recognition:
   - Facial Recognition: Ensemble models enhance the accuracy of facial recognition systems used in security and social media applications.
   - Speech-to-Text Systems: Combining different speech recognition models can improve the accuracy and reliability of transcriptions.
7. Autonomous Vehicles:
   - Object Detection: Ensuring safety in self-driving cars involves using ensembles to detect and classify objects on the road by integrating outputs from multiple sensors and models.
   - Path Planning: Ensembles help in planning the safest and most efficient route by combining predictions from various navigation models.
8. Cybersecurity:
   - Anomaly Detection: Ensemble models improve the detection of unusual patterns that might indicate security breaches or fraudulent activities.
   - Spam Filtering: Email systems use ensembles to filter out spam by combining the strengths of various filtering algorithms.

Big Picture Responses- C. What are some differences/similarities between GBDT and Random Forest ?

Gradient Boosting Decision Trees (GBDT) and Random Forests are both ensemble learning methods based on decision trees. While they share some similarities, they differ significantly in their approach and implementation. Random Forest and GBDT both leverage the power of multiple decision trees to enhance predictive performance, but they do so in fundamentally different ways. Random Forest emphasizes reducing variance through parallel tree construction, while GBDT focuses on reducing both bias and variance through sequential error correction.

Similarities:

1. Decision Trees: Both GBDT and Random Forest use decision trees as their base learners.
2. Ensemble Learning: Both methods build multiple trees and combine their predictions to improve overall performance.
3. Handling Various Types of Data: Both methods can handle various types of data, including numerical and categorical features.
4. Feature Importance: Both techniques can provide insights into feature importance, helping to understand which features contribute most to the predictions.

Differences:

1. Model Construction:
   - Random Forest: Builds trees independently in parallel. Each tree is built using a bootstrap sample of the data and a random subset of features at each split.
   - GBDT: Builds trees sequentially. Each tree is constructed to correct the errors made by the previous trees, focusing on the residuals of the combined previous trees' predictions.
2. Combining Predictions:
   - Random Forest: Uses averaging (for regression) or majority voting (for classification) to combine the predictions of the individual trees.
   - GBDT: Uses a weighted sum of the predictions from all the trees. Each tree’s contribution is scaled by a learning rate parameter.
3. Bias-Variance Trade-off:
   - Random Forest: Primarily reduces variance by averaging the predictions of many trees, which makes it less prone to overfitting compared to individual decision trees.
   - GBDT: Reduces both bias and variance by sequentially correcting errors, but it can be more prone to overfitting, especially if the number of trees is too large or the learning rate is too high.
4. Training Time:
   - Random Forest: Generally faster to train because trees are built independently and can be parallelized.
   - GBDT: Usually slower to train because trees are built sequentially, and each tree depends on the previous ones.
5. Hyperparameters:
   - Random Forest: Key hyperparameters include the number of trees, the maximum depth of trees, and the number of features considered for splitting.
   - GBDT: Key hyperparameters include the number of trees, the learning rate, the maximum depth of trees, and the subsample ratio.
6. Use Cases:
   - Random Forest: Often used when the primary goal is robustness and ease of use, as it requires less hyperparameter tuning and is less prone to overfitting.
   - GBDT: Preferred when higher predictive accuracy is needed and when the model can be fine-tuned through extensive hyperparameter tuning.

Big Picture Responses D-When would it not be appropriate/optimal to use ensemble models like Random Forest or GBDT ?

While ensemble models like Random Forest and Gradient Boosting Decision Trees (GBDT) are powerful, there are scenarios where their use might not be appropriate or optimal:

1. Small Datasets:
   - Overfitting Risk: Ensemble methods, especially GBDT, can overfit small datasets. The complex models they build can capture noise instead of the underlying pattern.
   - Computational Overhead: The computational cost may not justify the benefits when the dataset is small, as simpler models might perform just as well with less effort.
2. High-Dimensional Sparse Data:
   - Efficiency Issues: For datasets with a large number of features and many zero values (e.g., text data represented as word counts), tree-based methods may be inefficient.
   - Alternative Models: Linear models (like logistic regression or linear SVM) and models specifically designed for high-dimensional data (like certain types of neural networks) might perform better.
3. Real-Time Predictions:
   - Latency: Ensemble methods, especially those with a large number of trees, can be computationally intensive and slow, making them unsuitable for real-time prediction requirements.
   - Simpler Alternatives: Simpler models or pre-trained models optimized for fast inference might be more appropriate for real-time applications.
4. Interpretability:
   - Complexity: Ensemble models, particularly GBDT, can be complex and difficult to interpret. If the application requires easily interpretable models for decision-making or regulatory reasons, simpler models like linear regression, decision trees, or rule-based systems might be preferred.
5. Resource Constraints:
   - Computational Resources: Training and tuning ensemble models can be resource-intensive in terms of CPU/GPU and memory. If computational resources are limited, simpler models might be more feasible.
   - Deployment Constraints: The complexity and resource requirements of deploying ensemble models might be prohibitive in environments with limited computational power or memory.
6. Baseline Model Sufficiency:
   - Performance Needs: If a simpler model already achieves satisfactory performance, the additional complexity and resource consumption of an ensemble model may not be justified.
   - Cost-Benefit Analysis: The trade-off between the marginal improvement in performance and the increased complexity and cost should be considered.
7. Hyperparameter Tuning:
   - Time and Expertise: Ensemble models, especially GBDT, often require extensive hyperparameter tuning to achieve optimal performance. This process can be time-consuming and requires expertise. Automated machine learning (AutoML) solutions/simpler models are more practical in situations where time/expertise are limited.

Big Picture Responses- E.What are some disadvantages of using Decision Trees and how would you solve for them ?

1. Overfitting: (Decision trees can easily overfit the training data, especially if they are allowed to grow very deep and complex)
   - Solutions:
     - Pruning: Reduce the size of the tree by removing branches that have little importance, which can be done using cost complexity pruning.
     - Set Maximum Depth: Limit the maximum depth of the tree to prevent it from becoming too complex.
     - Min Samples per Leaf/Split: Set a minimum number of samples required to be at a leaf node or to split an internal node, which can help reduce overfitting.
2. High Variance: (Decision trees are sensitive to small variations in the training data, which can lead to high variance)
   - Solutions:
     - Ensemble Methods: Use methods like Random Forest or Gradient Boosting which combine multiple trees to reduce variance.
     - Cross-Validation: Use cross-validation to ensure that the tree generalizes well to unseen data.
3. Bias Towards Dominant Features: (Decision trees can be biased towards features with more levels or categories)
   - Solutions:
     - Feature Engineering: Ensure proper feature scaling and encoding.
     - Balanced Datasets: Balance the dataset by addressing class imbalance through techniques like SMOTE (Synthetic Minority Over-sampling Technique).
4. Instability: (Small changes in the data can result in significantly different tree structures)
   - Solutions:
     - Ensemble Methods: Using Random Forest or boosting methods can stabilize predictions by averaging multiple trees.
     - Bagging: Use bootstrapped samples of the data to build multiple trees and average their predictions.
5. Interpretability with Large Trees: (Large trees can become complex and hard to interpret)
   - Solutions:
     - Pruning: Prune the tree to reduce its size and make it more interpretable.
     - Simpler Models: Use simpler models or restrict the depth and size of the tree.
6. Difficulty in Capturing Smooth Relationships: (Decision trees are piecewise constant and may not capture smooth relationships well)
   - Solutions:
     - Ensemble Methods: Boosting methods like Gradient Boosting can model complex relationships better by combining multiple trees.
     - Hybrid Models: Combine decision trees with other models like linear regression or use techniques like model stacking.
7. Sensitivity to Irrelevant Features: (Decision trees can be sensitive to irrelevant features, which can affect the model's performance)
   - Solutions:
     - Feature Selection: Perform feature selection to remove irrelevant or redundant features before training the model.
     - Regularization: Use regularization techniques to penalize the complexity of the tree.

By addressing these disadvantages through various techniques and strategies, decision trees can be made more robust and effective in practical applications.

Big Picture Responses –F. Why does using an ensemble of models often perform better than a single model ?

By leveraging the complementary strengths and reducing the weaknesses of individual models, stacking can improve overall model performance and robustness.

1. Combining Strengths of Different Models:
   - Diverse Algorithms: Stacking leverages the strengths of multiple algorithms, each of which may capture different patterns or relationships in the data. By combining these diverse models, the final stacked model can achieve better performance than any individual model.
   - Complementary Errors: Different models often make different errors. Stacking can combine these models in a way that mitigates the weaknesses of individual models, leading to more accurate overall predictions.
2. Reduction of Overfitting:
   - Blending Models: By combining multiple models, stacking can reduce the risk of overfitting associated with any single model. This is because the ensemble can smooth out the idiosyncrasies of individual models, leading to better generalization on unseen data.
3. Incorporating Meta-Learning:
   - Meta-Model: In stacking, a meta-model (or second-level model) is trained to predict the target variable using the outputs of the base models as inputs. This meta-model learns to optimally combine the predictions of the base models, further enhancing predictive performance.
   - Learning from Predictions: The meta-model can capture interactions between the predictions of the base models, which individual base models cannot capture alone.
4. Flexibility and Adaptability:
   - Different Feature Spaces: Stacking allows the use of different feature spaces or representations in the base models. For instance, one model might use raw features, while another might use engineered features or transformed versions of the data.
   - Custom Weighting: The meta-model can assign different weights to the base models, effectively prioritizing models that perform better for certain aspects of the data.
5. Robustness to Model Selection:
   - Model Averaging: By averaging or combining the predictions of multiple models, stacking becomes less sensitive to the choice of any single model, providing a more robust and stable prediction.
   - Error Compensation: If some base models are prone to specific types of errors, other models in the stack can compensate for these errors, leading to improved overall performance.
6. Enhanced Predictive Power:
   - Boosting Weak Models: Even weak models can contribute valuable information in a stacking framework. The meta-model can effectively harness this information, leading to enhanced predictive power.
   - Capturing Complex Patterns: Stacking can capture complex patterns and interactions in the data that single models might miss, as the combined knowledge from multiple models provides a more comprehensive understanding of the data.

SUMMARY – ENSEMBLE METHODS - STACKING

Example of Stacking:

1. Base Models: Train several different types of models (e.g., linear regression, decision trees, neural networks) on the training data.
2. Generate Predictions: Use these base models to generate predictions on a validation set.
3. Meta-Model Training: Use the predictions from the base models as input features to train a meta-model on the validation set.
4. Final Prediction: For new data, use the base models to generate predictions, feed these predictions into the meta-model, and obtain the final prediction.

SUMMARY – ENSEMBLE METHODS - BAGGING

Model bagging, short for "bootstrap aggregating," is an ensemble learning technique where multiple models (often of the same type) are trained on different subsets of the data, and their predictions are aggregated to produce a final prediction. Here are some examples of model bagging:

Examples of Model Bagging:

1. Random Forest:
   - Description: Random Forest is one of the most well-known examples of bagging. It builds multiple decision trees using bootstrapped samples of the training data and a random subset of features for each split.
   - Aggregation: The final prediction is made by averaging the predictions (for regression) or majority voting (for classification) from all the trees.
2. Bagged Decision Trees:
   - Description: This method involves creating multiple decision trees using different bootstrapped samples of the training data.
   - Aggregation: The predictions from these trees are aggregated through averaging (for regression) or majority voting (for classification).
3. Bagging for Regression (Bagged Linear Regression):
   - Description: Multiple linear regression models are trained on different bootstrapped samples of the training data.
   - Aggregation: The final prediction is obtained by averaging the predictions from all the linear regression models.
4. Bagged k-Nearest Neighbors (k-NN):
   - Description: Multiple k-NN models are trained on different bootstrapped samples of the training data.
   - Aggregation: The predictions from the different k-NN models are averaged (for regression) or majority voted (for classification) to get the final prediction.
5. Bagged Support Vector Machines (SVM):
   - Description: Multiple SVM models are trained on different bootstrapped samples of the training data.
   - Aggregation: The predictions from these SVM models are combined, typically through averaging (for regression) or majority voting (for classification).
6. Bagged Neural Networks:
   - Description: Multiple neural network models are trained on different bootstrapped samples of the training data.
   - Aggregation: The final prediction is the average of the predictions from the different neural networks.
7. Bagged Logistic Regression:
   - Description: Multiple logistic regression models are trained on different bootstrapped samples of the training data.
   - Aggregation: The predictions (probabilities) from these logistic regression models are averaged to get the final prediction.

Benefits of Bagging:

- Reduction in Variance: Bagging helps to reduce the variance of the model by averaging the predictions of multiple models. This leads to more stable and reliable predictions.
- Robustness to Overfitting: Since each model is trained on a different subset of data, the ensemble is less likely to overfit compared to individual models.
- Improved Accuracy: By combining the predictions of multiple models, bagging can improve the overall predictive accuracy.

SUMMARY – ENSEMBLE METHODS - BOOSTING

Model boosting is another powerful ensemble learning technique where multiple models are trained sequentially, each one correcting the errors of its predecessor. Here are some popular examples of model boosting:

Examples of Model Boosting:

1. AdaBoost (Adaptive Boosting):

- **Description: AdaBoost sequentially trains weak learners (typically shallow decision trees) by focusing more on the samples that previous learners misclassified. Each subsequent model is trained on a modified version of the data where misclassified instances are given higher weights.**
- **Aggregation: The final prediction is a weighted sum of the predictions from all the learners, with weights reflecting each learner's accuracy.**

2. Gradient Boosting:

- **Description: Gradient Boosting builds models sequentially by training each new model to predict the residual errors (gradients) of the previous models. This process is repeated for a fixed number of iterations or until the residuals become sufficiently small.**
- **Aggregation: The final prediction is the sum of the predictions from all the models, each adjusted by a learning rate.**

3. XGBoost (Extreme Gradient Boosting):

- **Description: XGBoost is an optimized and scalable implementation of Gradient Boosting that includes regularization to reduce overfitting, efficient handling of sparse data, and other optimizations to speed up training.**
- **Aggregation: Similar to Gradient Boosting, the final prediction is the sum of the predictions from all the trees, each adjusted by a learning rate.**

4. LightGBM (Light Gradient Boosting Machine):

- **Description: LightGBM is another efficient and scalable implementation of Gradient Boosting, designed for large datasets and high-dimensional data. It uses techniques like histogram-based decision trees and leaf-wise tree growth to improve training speed and accuracy.**
- **Aggregation: The final prediction is the sum of the predictions from all the trees, each adjusted by a learning rate.**

5. CatBoost (Categorical Boosting):

- **Description: CatBoost is a gradient boosting algorithm that handles categorical features natively and efficiently. It uses ordered boosting to reduce prediction shift and provides robust handling of categorical variables.**
- **Aggregation: The final prediction is the sum of the predictions from all the trees, each adjusted by a learning rate.**

### Implementation Example in Python (XGBoost):

Here's a simple example of using XGBoost with the scikit-learn library in Python:

Why Boosting is Effective:

1. Focus on Difficult Instances: Boosting focuses on the samples that are hard to classify, improving the model's performance on these difficult cases.

2. Reduction of Bias and Variance: Boosting reduces both bias (by fitting the data better) and variance (through regularization techniques).

3. Improved Performance: Sequentially correcting errors allows boosting algorithms to achieve higher predictive accuracy compared to individual models or simpler ensemble methods like bagging.

By leveraging these advantages, boosting techniques like AdaBoost, Gradient Boosting, XGBoost, LightGBM, and CatBoost have become popular choices for many machine learning tasks.

SUMMARY – ENSEMBLE METHODS - STACKING VS BAGGING VS BOOSTING

Bagging (Bootstrap Aggregating)

When to use:

- High Variance Models: Bagging is effective with models that have high variance, such as decision trees. By creating multiple versions of the model using different subsets of the data and averaging their predictions, it reduces overfitting.
- Simple Base Learners: It works well with simpler base learners where each model might overfit the data if used alone.
- Parallel Computation: Bagging can be easily parallelized since each model is trained independently.

Examples:

- Random Forests: An ensemble of decision trees trained on different subsets of the data.

Boosting

When to use:

- High Bias Models: Boosting is beneficial for models that have high bias. It sequentially builds models, each one correcting the errors of the previous one, thus reducing bias and improving performance.
- Complex Relationships: It is useful when dealing with complex relationships in the data, as each model attempts to correct the mistakes of its predecessors.
- Accuracy Improvement: Boosting often leads to higher accuracy compared to bagging, especially when the data has complex patterns.

Examples:

- Gradient Boosting: Models the residuals of the previous models to improve performance.

Stacking

When to use:

- Combining Different Models: Stacking is used when you want to combine the predictions of different types of models to leverage their individual strengths. It uses a meta-model to learn how to best combine the base models.
- Complex Ensemble: Suitable when you have diverse models that individually capture different aspects of the data.
- Blending Predictions: It can blend predictions from different families of algorithms, potentially leading to better performance than using any single model.

Examples:

- A stacking ensemble that combines logistic regression, decision trees, and support vector machines, with a meta-model (often a simple model like logistic regression) that learns the best way to combine their outputs.

Summary

- Use bagging when you want to reduce variance and have models prone to overfitting (e.g., decision trees).
- Use boosting when you need to reduce bias and handle complex patterns in the data.
- Use stacking when you want to combine different types of models to leverage their individual strengths and capture different aspects of the data.

SUMMARY – EVALUATION METRICS

Root Mean Square Error (RMSE) and R-squared (R^2) are two different metrics used to evaluate the performance of a regression model. Here's a comparison:

RMSE (Root Mean Square Error)

- Definition: RMSE measures the average magnitude of the errors between predicted and actual values. It is the square root of the average of squared differences between prediction and actual observation.
- Interpretation:
  - RMSE is always non-negative, with a value of 0 indicating a perfect fit.
  - Lower RMSE values indicate a better fit to the data.
  - It is sensitive to outliers as it squares the errors before averaging.
- Scale: RMSE has the same units as the dependent variable.

R-squared (R^2)

- Definition: R^2, also known as the coefficient of determination, measures the proportion of the variance in the dependent variable that is predictable from the independent variables.
- Interpretation:
  - R^2 ranges from 0 to 1 (or 0% to 100%), with 0 indicating that the model explains none of the variability of the response data around its mean, and 1 indicating that it explains all the variability.
  - Higher R^2 values indicate a better fit.
  - It can be negative if the model is worse than a horizontal line.
- Scale: R^2 is unitless and represents the proportion of explained variance.

Key Differences

- Purpose: RMSE provides a measure of the average error in the same units as the dependent variable, useful for understanding the magnitude of prediction errors. R^2 indicates the goodness of fit, showing how well the model explains the variance in the data.
- Sensitivity: RMSE is sensitive to outliers, while R^2 is less sensitive to them.
- Interpretability: RMSE is more intuitive in terms of the actual error magnitude, while R^2 is more abstract but provides a percentage of explained variance.

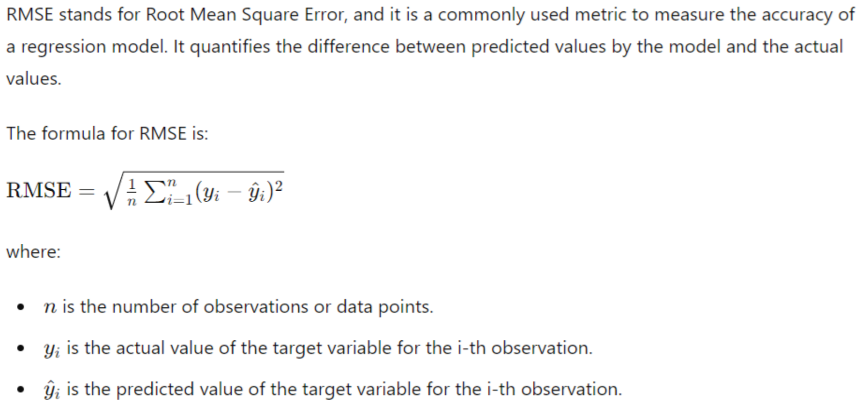
When to Use Each

- RMSE: Use RMSE when you need to know the actual prediction error in the same units as the response variable. It is particularly useful for comparing the predictive accuracy of different models.
- R^2: Use R^2 when you want to understand how well your model explains the variability of the data. It is commonly used to assess the overall fit of the model.

SUMMARY – EVALUATION METRICS

Root Mean Square Error (RMSE) and R-squared (R²) are two different metrics used to evaluate the performance of a regression model. Here's a comparison:

RMSE (Root Mean Square Error)



RMSE stands for Root Mean Square Error, and it is a commonly used metric to measure the accuracy of a regression model. It quantifies the difference between predicted values by the model and the actual values.

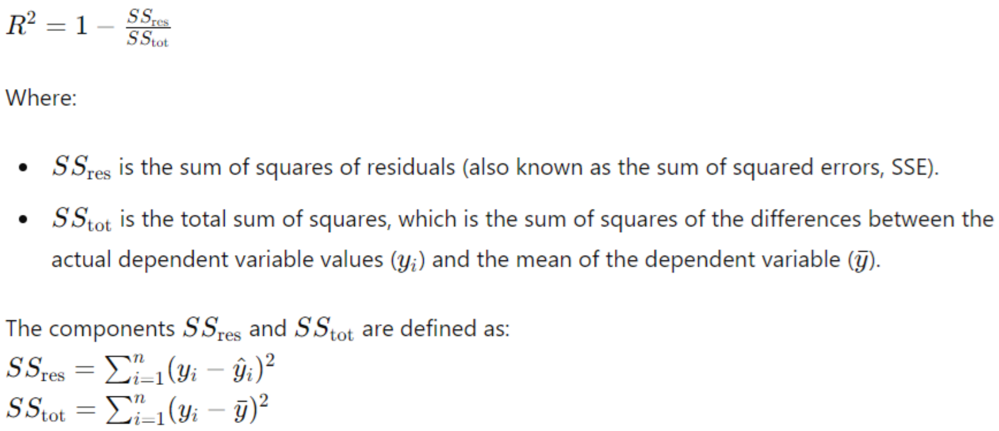
The formula for RMSE is:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where:

- n is the number of observations or data points.
- y_i is the actual value of the target variable for the i-th observation.
- $\hat{y}_i$ is the predicted value of the target variable for the i-th observation.

R-squared (R²)



$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

Where:

- SS_{res} is the sum of squares of residuals (also known as the sum of squared errors, SSE).
- SS_{tot} is the total sum of squares, which is the sum of squares of the differences between the actual dependent variable values (y_i) and the mean of the dependent variable ($\bar{y}$).

The components SS_{res} and SS_{tot} are defined as:

$$SS_{\text{res}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
$$SS_{\text{tot}} = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

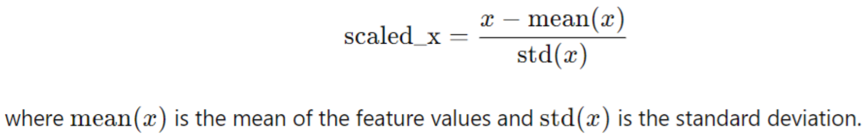
SUMMARY – Standard Scaler

How StandardScaler Works:

Fit: When you instantiate a StandardScaler object and then call its fit method on a dataset, it calculates the mean and standard deviation of each feature (or other statistics, depending on parameters) from the training data. These statistics are stored in the scaler object.

Transform: Once the scaler is fitted, you can transform the training data using the transform method. This centers the data by subtracting the mean calculated during the fit step and then scales it by dividing by the standard deviation (or other measures, depending on parameters).

Scaling Formula: The transformation applied to each feature xx is given by:



$$\text{scaled\_x} = \frac{x - \text{mean}(x)}{\text{std}(x)}$$

where $\text{mean}(x)$ is the mean of the feature values and $\text{std}(x)$ is the standard deviation.

```
from sklearn.preprocessing import StandardScaler
import numpy as np

data = np.array([[0, 0], [1, 0], [0, 1], [1, 1]])

# Create a StandardScaler instance
scaler = StandardScaler()

# Fit the scaler on the data (calculates mean and std)
scaler.fit(data)

# Transform the data based on the scaler's fit
scaled_data = scaler.transform(data)
```

BREAK
THROUGH
TECH

Class Discussion

Break

BREAK
THROUGH
TECH

BREAK THROUGH TECH

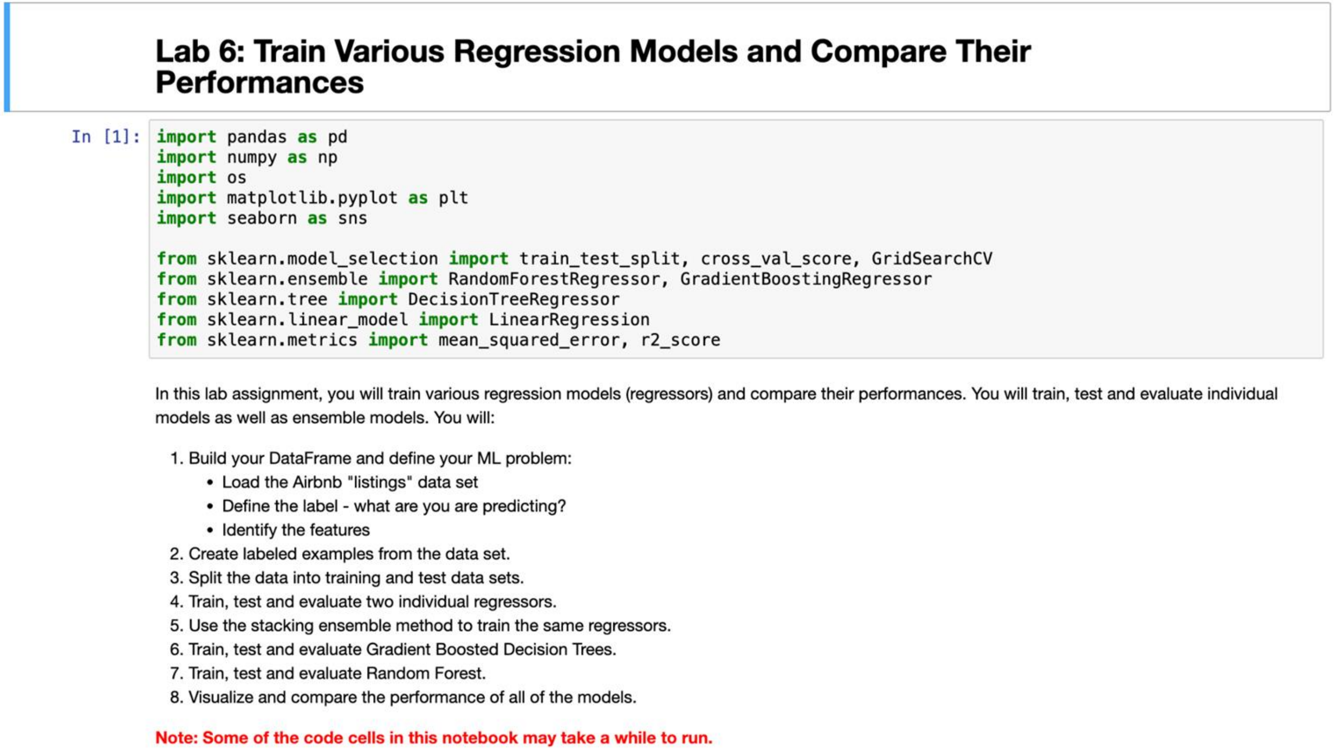
Breakout Groups: Lab Assignment

Lab 6

In this lab, you will:

- Build your DataFrame and define your ML problem
- Create labeled examples from the data set, and split the data into training and test data sets
- Train, test, and evaluate two individual regressors and three ensemble regressors to solve your ML problem
- Visualize and compare the performance of the individual models and the ensemble models

Lab 6



Lab 6: Train Various Regression Models and Compare Their Performances

In [1]:

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In this lab assignment, you will train various regression models (regressors) and compare their performances. You will train, test and evaluate individual models as well as ensemble models. You will:

1. Build your DataFrame and define your ML problem:
    - Load the Airbnb "listings" data set
    - Define the label - what are you are predicting?
    - Identify the features
2. Create labeled examples from the data set.
3. Split the data into training and test data sets.
4. Train, test and evaluate two individual regressors.
5. Use the stacking ensemble method to train the same regressors.
6. Train, test and evaluate Gradient Boosted Decision Trees.
7. Train, test and evaluate Random Forest.
8. Visualize and compare the performance of all of the models.

Note: Some of the code cells in this notebook may take a while to run.

BREAK
THROUGH
TECH

Working Session 1 Debrief

Lab Debrief

So far,

- What did you enjoy about this lab?
- What did you find difficult about this lab?

BREAK
THROUGH
TECH

Working Session 2 Debrief

Lab Debrief

- What did you enjoy about this lab?
- What did you find hard about this lab?
- What questions do you still have about this lab?
- How did you approach problem-solving during the exercise?
- What would you do differently if you were to repeat the exercise?

BREAK
THROUGH
TECH

Concluding Remarks

Concluding Remarks

- Key takeaways
- Additional resources

Next week

In the following week, you will:

- Explore the design of a neural network using the basic components of network architecture
- See how a neural network is trained and optimized
- Use Keras to implement a neural network to make predictions
- Explore the field of Computer Vision
- Implement a neural network for image classification

And in the lab, you will:

- Define your ML problem
- Import image data and split data into training and test data sets
- Inspect and visualize the data
- Prepare your data so that it is ready for modeling
- Construct and train a convolutional neural network model
- Evaluate the model's performance on the training and test data

BREAK
THROUGH
TECH

Content + Lab Feedback Survey

Content + Lab Feedback Survey

To complete your lab, please answer the following questions about BOTH your online modules and your lab experience. Your input will help pay it forward to the Break Through Tech student community by enabling us to continuously improve the learning experience that we provide to our community.

Thank you for your thoughtful feedback!

https://forms.gle/eUQQZgS6BPRpqgZ7A