Lecture 9: Tree-Based Methods and Ensemble Learning

Readings: ESL (Ch. 9-10, 15), ISL (Ch. 8), Bach (Ch. 10); code

Soon Hoe Lim

September 23, 2025

Outline

The Limitation of Linear Models

Linear Model Assumptions

Linear models assume: $\mathbb{E}[Y|X] = \beta_0 + \sum_{j=1}^{p} \beta_j X_j$

- Additive, linear relationships between features and response
- Constant effects across the entire feature space
- Limited ability to capture interactions automatically

The Curse of Dimensionality: Fully non-parametric approaches (kernel methods, k-NN) require exponentially more data as dimension p increases. To maintain density, we need N^p observations.

Real-world relationships are often:

- Non-linear: $f(x) = x^2 + \sin(x)$
- Interactive: $f(x_1, x_2) = x_1 \cdot x_2$
- Piecewise: Different behavior in different regions

? Can we build models that adapt their complexity to the data in a natural, interpretable way?

Additive Models: A Flexible Compromise

Definition 1: Additive Model

An additive model assumes:

$$\mathbb{E}[Y|X_1, \ldots, X_p] = \alpha + \sum_{j=1}^{p} f_j(X_j)$$

where each f_j is an unknown smooth univariate function.

Key advantages:

- **Interpretability:** Effect of X_j on Y captured by $f_j(X_j)$ alone
- **Flexibility:** Each f_j can be any smooth function
- **Dimensionality:** Avoids curse by modeling univariate functions
- **Identifiability:** Require $\sum_{i=1}^{N} f_j(x_{ij}) = 0$ for each j

Extensions: Can include selected interactions: $f_{jk}(X_j, X_k)$

The Backfitting Algorithm

Minimize RSS: $\sum_{i=1}^{N}\left(y_i - \alpha - \sum_{j=1}^{p} f_j(x_{ij})\right)^2$

Backfitting Algorithm

Initialize: $\hat{\alpha} = \bar{y}$, $\hat{f}_j \equiv 0$ for $j = 1, \ldots, p$
repeat
    for $j = 1, \ldots, p$ **do**
        Compute partial residuals:

$$r_{ij} = y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik})$$

        Smooth partial residuals:

$$\hat{f}_j \leftarrow \mathcal{S}_j[\{(x_{ij}, r_{ij})\}_{i=1}^{N}]$$

        Center: $\hat{f}_j(x) \leftarrow \hat{f}_j(x) - \frac{1}{N}\sum_{i=1}^{N} \hat{f}_j(x_{ij})$
    end for
until convergence

Common smoothers: splines, local regression, kernel methods

Introduction to Decision Trees

Definition 2: Tree-Based Model

Partition feature space into M disjoint regions $R_1, \ldots, R_M$:

$$f(\mathbf{x}) = \sum_{m=1}^{M} c_m \mathbb{I}(\mathbf{x} \in R_m)$$

where c_m is the prediction in region R_m.

Regional Predictions

- **Regression:** $c_m = \text{mean}(y_i : \mathbf{x}_i \in R_m)$
- **Classification:** $c_m = \text{mode}(y_i : \mathbf{x}_i \in R_m)$

Advantages and Disadvantages

Advantages:

- Simple interpretation (if-then rules)
- Handles mixed data types naturally
- Automatic variable selection
- Captures interactions naturally
- No assumptions about data distribution

Disadvantages:

- High variance (instability)
- Limited smoothness
- Greedy construction may miss optimal splits

Growing Regression Trees

CART Algorithm for Regression

Input: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, stopping criteria
Initialize: Root node containing all data
for each internal node t **do**
    if stopping criterion not met **then**
        for each variable X_j and split point s **do**
            Define: $t_L = \{\mathbf{x} \in t : x_j \leq s\}$, $t_R = \{\mathbf{x} \in t : x_j > s\}$
            Compute improvement:

$$\Delta(s, j) = \text{RSS}(t) - \text{RSS}(t_L) - \text{RSS}(t_R)$$

        end for
        Choose: $(s^*, j^*) = \arg\max_{s,j} \Delta(s, j)$
        Split node if $\Delta(s^*, j^*) >$ threshold
    end if
end for

Where $\text{RSS}(t) = \sum_{\mathbf{x}_i \in t} (y_i - \bar{y}_t)^2$

Classification Trees: Impurity Measures

For K classes, let $\hat{p}_{mk}$ = proportion of class k in node m.

Node Impurity Measures

1. **Misclassification Error:** $1 - \max_k \hat{p}_{mk}$
2. **Gini Index:** $\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$
3. **Cross-Entropy:** $-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$

Why Gini/Entropy over Misclassification?

- More sensitive to changes in node probabilities
- Differentiable (better for optimization)
- Lead to better intermediate splits during construction

The choice between Gini and entropy rarely affects final performance significantly.

Tree Pruning: Cost-Complexity Method

Problem: Large trees overfit; simple stopping rules are suboptimal.

Definition 3: Cost-Complexity Pruning

For subtree $T \subseteq T_0$ (full tree), minimize:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T| \text{ , where}$$

- $|T|$ = number of terminal nodes
- N_m = number of observations in node m
- $Q_m(T)$ = node impurity
- α = complexity parameter

Pruning Strategy

1: Grow large tree T_0
2: For each α, find $T(\alpha) = \arg\min_T C_\alpha(T)$
3: Use cross-validation to select optimal α^*
4: Return $T(\alpha^*)$

Understanding Tree Instability

High Variance Problem: Small changes in training data can lead to very different tree structures and predictions.

Sources of instability:

- **Hierarchical Effect:** Errors in early splits propagate throughout tree
- **Greedy Selection:** Locally optimal splits may be globally suboptimal
- **Discrete Splits:** Small data changes can move split points dramatically
- **Hard Boundaries:** Abrupt transitions between regions

Bias-Variance Trade-off for Trees

- **Low Bias:** Can approximate complex decision boundaries
- **High Variance:** Very sensitive to training data
- **Result:** Poor generalization despite good training fit

Intuition: If $\mathbf{x}$ is near split point s, small training changes can move $\mathbf{x}$ to different sides, causing completely different predictions.

MARS: A More Stable Alternative

Multivariate Adaptive Regression Splines combine tree flexibility with linear stability.

Definition 4: MARS Model

$f(\mathbf{x}) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(\mathbf{x})$, where each basis function is:

$$h_m(\mathbf{x}) = \prod_{k=1}^{K_m} [s_{km} \cdot (x_{v(k,m)} - t_{km})]_+,$$

- $[\cdot]_+ = \max(0, \cdot)$ (hinge function)
- $s_{km} \in \{-1, +1\}$ (direction)
- $v(k, m)$ selects variable, t_{km} is knot

Key advantages over trees:

- Continuous everywhere (vs. discontinuous)
- Piecewise linear (vs. piecewise constant)
- More stable to data perturbations
- Built-in model selection via GCV

The Ensemble Philosophy

Key Insight: Instead of finding one "perfect" model, combine many "good" models to achieve better performance than any individual model.

Why ensembles work:

- **Variance Reduction:** Averaging reduces variance without increasing bias
- **Bias Reduction:** Sequential methods can reduce bias
- **Improved Stability:** Less sensitive to outliers and data peculiarities
- **Better Generalization:** Combine different "views" of the data

Two main approaches:

1. **Parallel:** Train models independently, then combine (Bagging, Random Forest)
2. **Sequential:** Train models sequentially, each learning from previous mistakes (Boosting)

Bagging: Bootstrap Aggregating

Definition 5: Bagging Algorithm

1. For $b = 1, \ldots, B$:
2. Draw bootstrap sample $\mathcal{Z}^{*b}$ of size N (sampling with replacement)
3. Train model $\hat{f}_b$ on $\mathcal{Z}^{*b}$

Final prediction:

$$\hat{f}_{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(\mathbf{x})$$

Why bagging helps:

- **Variance reduction:** If models are independent with variance σ^2, ensemble variance is σ^2/B
- **Bias unchanged:** $\mathbb{E}[\hat{f}_{\text{bag}}] = \mathbb{E}[\hat{f}]$
- **Most effective:** For high-variance, low-bias models (like unpruned trees)

Out-of-Bag (OOB) Error: Each bootstrap sample omits $\sim 37\%$ of data; use these for validation.

Random Forest: Adding Decorrelation

Problem with bagged trees: When predictors are correlated, trees become correlated, reducing variance reduction benefits.

Definition 6: Random Forest Algorithm

1. For $b = 1, \ldots, B$:
2. Draw bootstrap sample $\mathcal{Z}^{*b}$
3. Train tree on $\mathcal{Z}^{*b}$ with modification:
4. **At each split:** Randomly select $m \ll p$ features to consider
5. Choose best split from these m features only

Key parameters:

- B: Number of trees (larger usually better, diminishing returns)
- m: Features per split ($\sqrt{p}$ for classification, $p/3$ for regression)
- Tree depth: Usually grown deep (minimal pruning)

Benefits: Combines variance reduction (bagging) with decorrelation (random features)

Random Forest: Variable Importance

Two methods for measuring feature importance:

1. Mean Decrease in Impurity (Gini Importance)
   - For each tree, sum impurity decreases for splits using feature j
   - Average across all trees
   - Fast to compute, but can be biased toward high-cardinality features

2. Permutation Importance (Preferred)
   - Record baseline OOB error
   - For each feature j: randomly permute feature j in OOB samples
   - Recompute OOB error
   - Importance = increase in error due to permutation
   - More reliable, model-agnostic measure

Applications: Feature selection, understanding model behavior, identifying redundant features

The Boosting Philosophy

Central Question: "Can weak learners be combined into a strong learner?"

Definition 7: Weak vs Strong Learners

- **Weak Learner:** Slightly better than random (error ¡ 0.5 for binary classification)
- **Strong Learner:** Arbitrarily low error rate achievable

Boosting strategy:

1. Start with equal weights on training observations
2. Fit weak learner to weighted data
3. Increase weights on misclassified observations
4. Repeat: each new learner focuses on "hard" cases
5. Combine learners with weighted voting

Theoretical guarantee: Training error decreases exponentially with number of rounds (Freund-Schapire, 1997).

AdaBoost: The Breakthrough Algorithm

Binary classification with $Y \in \{-1, +1\}$, weak learners $G_m(\mathbf{x}) \in \{-1, +1\}$.

AdaBoost Algorithm

1: **Initialize:** $w_i^{(1)} = 1/N$ for $i = 1, \ldots, N$
2: **for** $m = 1$ to M **do**
3: **Fit classifier:** $G_m = \arg\min_G \sum_{i=1}^N w_i^{(m)} \mathbb{I}(y_i \neq G(\mathbf{x}_i))$
4: **Compute error:** $\text{err}_m = \frac{\sum_{i=1}^N w_i^{(m)} \mathbb{I}(y_i \neq G_m(\mathbf{x}_i))}{\sum_{i=1}^N w_i^{(m)}}$
5: **Compute weight:** $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
6: **Update weights:** $w_i^{(m+1)} = w_i^{(m)} \exp[\alpha_m \mathbb{I}(y_i \neq G_m(\mathbf{x}_i))]$
7: **Normalize:** $w_i^{(m+1)} \leftarrow w_i^{(m+1)} / \sum_j w_j^{(m+1)}$
8: **end for**
9: **Output:** $G(\mathbf{x}) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(\mathbf{x})\right]$

AdaBoost: Statistical Foundation

AdaBoost minimizes the exponential loss:

$$L(y, f) = \exp(-yf)$$

Why exponential loss?

- Differentiable (unlike 0-1 loss)
- Emphasizes misclassified examples exponentially
- Leads to closed-form weight updates
- Population minimizer: $f^*(\mathbf{x}) = \frac{1}{2} \log \frac{P(Y=1|\mathbf{x})}{P(Y=-1|\mathbf{x})}$

Connection to logistic regression: As $f \to \infty$: $\frac{\exp(-yf)}{1+\exp(-yf)} \to 0$

Limitation: Sensitive to outliers and noise (exponential penalty grows rapidly).

Gradient Boosting: The General Framework

Key insight: Boosting can be viewed as gradient descent in function space.

Definition 8: Forward Stagewise Additive Modeling

Fit models of the form: $f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m b(\mathbf{x}; \gamma_m)$
At each step m:

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma))$$

Definition 9: Gradient Boosting Principle

Approximate the solution by fitting to negative gradients (pseudo-residuals):

$$r_{im} = -\left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f=f_{m-1}}$$

This generalizes boosting to any differentiable loss function.

Common Loss Functions and Gradients

Loss Functions for Different Problems

- **Squared Loss:** $L(y, f) = \frac{1}{2}(y - f)^2 \Rightarrow r_i = y_i - f(\mathbf{x}_i)$
- **Absolute Loss:** $L(y, f) = |y - f| \Rightarrow r_i = \text{sign}(y_i - f(\mathbf{x}_i))$
- **Huber Loss:** Robust combination of squared and absolute loss
- **Logistic Loss:** $L(y, f) = \log(1 + \exp(-yf))$ for classification
- **Exponential Loss:** $L(y, f) = \exp(-yf)$ (AdaBoost special case)

Generic Gradient Boosting

Initialize: $f_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$
for $m = 1$ to M **do**
  Compute pseudo-residuals: $r_{im} = -\left.\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right|_{f=f_{m-1}}$
  Fit model: $h_m = \arg\min_h \sum_{i=1}^N (r_{im} - h(\mathbf{x}_i))^2$
  Line search: $\rho_m = \arg\min_\rho \sum_{i=1}^N L(y_i, f_{m-1}(\mathbf{x}_i) + \rho h_m(\mathbf{x}_i))$
  Update: $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \rho_m h_m(\mathbf{x})$
end for

Why Trees as Weak Learners?

Trees are ideal weak learners because they:

- Handle mixed data types naturally (numerical + categorical)
- Capture interactions automatically
- Require no data preprocessing (scaling, normalization)
- Handle missing values naturally
- Are computationally efficient
- Provide good bias-variance trade-off when shallow

Definition 10: Gradient Tree Boosting

Use regression trees as base learners $h_m(\mathbf{x})$:

1. Fit tree to pseudo-residuals, creating J_m terminal regions R_{jm}
2. For each region, optimize separately:
   $\gamma_{jm} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, f_{m-1}(\mathbf{x}_i) + \gamma)$
3. Update: $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}(\mathbf{x} \in R_{jm})$

Key insight: Optimize a separate coefficient for each leaf, not just one global step size.

Gradient Boosted Trees: Complete Algorithm

Gradient Tree Boosting

Initialize: $f_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$
for $m = 1$ to M **do**
    Compute pseudo-residuals: $r_{im} = -\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\Big|_{f=f_{m-1}}$
    Fit regression tree: Train tree T_m on $\{(\mathbf{x}_i, r_{im})\}_{i=1}^N$
    Create regions: Tree produces terminal regions $R_{jm}, j = 1, \ldots, J_m$
    for $j = 1$ to J_m **do**
        Optimize leaf values: $\gamma_{jm} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, f_{m-1}(\mathbf{x}_i) + \gamma)$
    end for
    Update model: $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}(\mathbf{x} \in R_{jm})$
end for

Key Hyperparameters and Tuning

1. Number of Boosting Rounds (M)

- Controls model complexity and fitting capacity
- Use early stopping on validation set (don't fix via CV)
- Typical range: 100–10,000 depending on learning rate

2. Learning Rate/Shrinkage (ν)

- Scales the contribution of each tree: $f_m = f_{m-1} + \nu \cdot h_m$
- Smaller ν requires larger M but often improves generalization
- Typical values: $\nu \in [0.01, 0.3]$, default often 0.1
- Trade-off: $\nu \times M \approx$ constant for similar performance

3. Tree Complexity (J)

- $J - 1 =$ number of interactions modeled
- Common values: $J = 2$ (stumps), $J = 6$ (5-way interactions)
- Rarely need $J > 10$; validation helps select optimal value

Advanced Regularization Techniques

Stochastic Gradient Boosting (Subsampling)

At each iteration, train on random subsample (50–80%) of data:

- Reduces overfitting through variance reduction
- Improves computational efficiency
- Acts as implicit regularization (similar to dropout)

Feature Subsampling

Randomly sample features at each:

- **Tree level:** Different random subset per tree
- **Split level:** Different random subset per split (more aggressive)
- Prevents overfitting to irrelevant features
- Especially helpful when p is large

Early Stopping

Monitor validation error and stop when it plateaus:
$M^* = \arg\min_M \text{ValidationError}(M)$. More reliable than fixed cross-validation for model selection.

Statistical View: Boosting as Regularization

Implicit regularization: Boosting solves approximately:
$\min_f \sum_{i=1}^N L(y_i, f(\mathbf{x}_i)) + \Omega(f)$

Sources of Regularization

1. **Shrinkage (ν):** Smaller steps create smoother paths in function space
2. **Early Stopping:** Limits model complexity before overfitting
3. **Tree Constraints:** Depth limits control interaction complexity
4. **Subsampling:** Adds stochastic regularization

Connection to Sparsity

Forward stagewise with infinitesimal steps (linear case) is equivalent to:
$\min_\beta \frac{1}{2}\|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda\|\beta\|_1$ This suggests boosting performs implicit feature selection.

Margin Theory and Generalization (Optional)

Definition 11: Classification Margin

For binary classification, the margin of example $(\mathbf{x}_i, y_i)$ is: $\text{margin}_i = y_i f(\mathbf{x}_i)$ where $f(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})$ and $y_i \in \{-1, +1\}$.

Theorem 1: Margin-Based Generalization Bound

The test error is bounded by: $P(\text{error}) \leq P(\text{margin} \leq \theta) + \mathcal{O}\left(\frac{\sqrt{d \log^2(N/d)}}{N\theta^2}\right)$ where d relates to the complexity of the weak learner class.

Key implications:

- Boosting increases margins of training examples over time
- Larger margins lead to better generalization bounds
- Explains why boosting rarely overfits even with many rounds
- Zero training error doesn't imply overfitting if margins keep growing

Loss Function Robustness

Robustness to Outliers

Different loss functions have varying sensitivity to outliers:

Loss Function	Robustness	Comments
Exponential	Poor	Very sensitive to outliers
Logistic	Moderate	Better than exponential
Squared	Poor	Quadratic penalty
Absolute (LAD)	Good	Linear penalty
Huber	Good	Combines squared + absolute

Huber Loss (Robust Alternative)

$L_\delta(y, f) = \begin{cases} \frac{1}{2}(y-f)^2 & \text{if } |y-f| \leq \delta \\ \delta|y-f| - \frac{1}{2}\delta^2 & \text{if } |y-f| > \delta \end{cases}$ Combines quadratic loss (small errors) with linear loss (large errors).

Computational Considerations

Time Complexity: $\mathcal{O}(M \times N \times p \times J \log J)$ per training, where:

- M = number of boosting rounds
- N = training set size
- p = number of features
- J = maximum tree depth

Optimization Strategies

- **Approximate split finding:** Use quantiles instead of all splits
- **Parallelization:** Across features and data points
- **Histogram methods:** Pre-bin continuous features (LightGBM)
- **Column sampling:** Random feature subsets
- **Early stopping:** Avoid unnecessary iterations
- **Memory optimization:** Efficient tree storage, gradient caching

Modern implementations: XGBoost, LightGBM, CatBoost provide highly optimized versions with additional regularization and system improvements.

Ensemble Method Comparison

Aspect	**Boosting**	**Bagging**	**Random Forest**
Construction	Sequential	Parallel	Parallel
Primary Goal	Bias reduction	Variance reduction	Variance + decorrelation
Base Learners	Weak (simple)	Strong (complex)	Medium complexity
Overfitting Risk	Moderate	Low	Very low
Interpretability	Moderate	Low	Low
Noise Sensitivity	High	Low	Low
Parallelization	Limited	Full	Full
Hyperparameter Tuning	Complex	Simple	Simple
Typical Performance	Excellent (tabular)	Good	Very good

Practical guide: Try both boosting and Random Forest. Boosting often wins on structured/tabular data competitions, while Random Forest is more robust for general practitioners.

Variable Importance in Boosted Trees

Definition 12: Relative Variable Importance

For variable X_j, sum squared improvements across all splits: $\hat{I}_j^2 = \sum_{m=1}^{M} \sum_{t \in \text{splits}(T_m)} \hat{i}_t^2 \cdot \mathbb{I}(\text{split uses } X_j)$ where $\hat{i}_t^2$ is the improvement in loss at split t.

Partial Dependence Plots

Show marginal effect of feature X_j: $\bar{f}_j(x_j) = \frac{1}{N} \sum_{i=1}^{N} f(x_j, \mathbf{x}_{i,\setminus j})$

Procedure:

1. Fix $X_j = x_j$ for all observations
2. Keep other features at their observed values
3. Compute average prediction
4. Plot $\bar{f}_j(x_j)$ vs x_j

Limitations: Assumes independence between features (can be misleading with strong interactions).

SHAP Values for Model Explanation

Definition 13: SHAP (Shapley Additive Explanations)

For any prediction $f(\mathbf{x})$, SHAP values satisfy: $f(\mathbf{x}) = \phi_0 + \sum_{j=1}^{p} \phi_j(\mathbf{x})$ where $\phi_0 = \mathbb{E}[f(\mathbf{X})]$ and $\phi_j(\mathbf{x})$ is feature *jscontribution*.

Desirable properties (Shapley axioms):

- **Efficiency:** Contributions sum to prediction
- **Symmetry:** Equal features get equal credit
- **Dummy:** Irrelevant features get zero credit
- **Additivity:** Consistent across model combinations

TreeSHAP Algorithm

- Computes exact SHAP values for tree ensembles efficiently
- Time complexity: $\mathcal{O}(TLD)$ where T = trees, L = leaves, D = depth
- Provides both local (individual) and global (feature importance) explanations

Key Takeaways

From Linear to Flexible Models

- **Additive models** balance flexibility with interpretability
- **Decision trees** capture interactions naturally but are unstable
- **MARS** provides stable piecewise-linear alternative to trees

Ensemble Methods Revolution

- **Bagging/Random Forest:** Parallel combination reduces variance
- **Boosting:** Sequential combination reduces bias
- **Key insight:** Combining many weak learners beats single strong learner

- Use Random Forest for robustness and ease of use
- Use gradient boosting for maximum accuracy on tabular data
- Always validate hyperparameters properly
- Consider interpretability requirements in method selection
- Modern implementations (XGBoost, LightGBM) provide excellent performance

Exercises

Exercise 9

1. **ESL 9.2.**
2. **ESL 9.3.**
3. **ESL 10.2.**
4. **ESL 10.9.**
5. **Experimental.** Explore the effect of different loss functions (squared, absolute, Huber) on boosting performance with outliers in the data.