BIOS 835: Ensemble Learning and Gradient Boosting Methods

Alexander McLain

October 26, 2023

Ensemble Learning

- Ensemble learning is a machine learning paradigm where multiple models (often called "learners" or "base models") are trained to solve the same problem and combined to get better results.
- The main principle behind ensemble learning is that a group of "weak learners" can come together to form a "strong learner".

Ensemble Learning

- Ensemble learning is a machine learning paradigm where multiple models (often called "learners" or "base models") are trained to solve the same problem and combined to get better results.
- The main principle behind ensemble learning is that a group of "weak learners" can come together to form a "strong learner".
- Why Use Ensemble Learning?
  - **Improve Accuracy:** Combining multiple models often results in a model with higher accuracy than any individual model.
  - **Reduce Overfitting:** By averaging out biases, the variance can decrease when combining multiple models.
  - **Enhance Robustness:** The ensemble is often more resilient to individual failures of any given model.

Types of Ensemble Techniques

- Bagging (Bootstrap Aggregating):
  - Example: Random Forest is an ensemble of Decision Trees using bagging and feature randomness.
- Boosting:
  - Models are trained sequentially, with each new model being trained to correct the errors of the previous ones.
  - Examples: AdaBoost, Gradient Boosting Machines (GBM), XGBoost.
- Stacking:
  - Multiple models are trained on the complete dataset.

Boosting

- Boosting is one of the most influential machine learning ideas of the last 25 years.
- The similarities between boosting and bagging are that they both are effective ways to make good classifiers from weak learners.
- Boosting, however, is used more generally than bagging (which is mostly used with CART).
- The most popular boosting method *"AdaBoost.M1"* was developed in Freund and Schapire (1997).

Boosting

- First let's consider a two-class classification problem where the outcome is $Y_i \in \{-1, 1\}$, $\boldsymbol{X} \in \mathbb{R}^p$ and $G(\boldsymbol{X}) \in \{-1, 1\}$.
- Here $G(\boldsymbol{X})$ will be a weak classifier (like CART).
- Suppose we have a sequence $G_m(\boldsymbol{X})$ for $m = 1, 2, \ldots, M$ of weak classifiers.
- The final prediction is going to be

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

where $\alpha_1, \alpha_2, \ldots, \alpha_M$ are weights that are computed by the boosting algorithm.

Boosting

- When applying boosting the observations are weighted by $w_1, w_2, \ldots, w_N$.
- These differentially weight each observation in the dataset.
- We start with $w_i = 1/N$ for all i, at each step the weights are changed.
- At step m, those observations that were misclassified at in $G_{m-1}(x)$ are given larger weights, those correctly classified are given smaller weights.
  - the weights for hard to predict observations get larger and larger
- The *AdaBoost.M1* algorithm follows
  - This is a general algorithm that can be applied to any method.

AdaBoost.M1 algorithm (Discrete AdaBoost)

1. Initialize the observation weights $w_i = 1/N$ for all i.
2. For $m = 1$ to M
   2.1 Fit a classifier $G_m(\boldsymbol{x})$ to the training data using weights w_i.
   2.2 Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I\left(y_i \neq G_m\left(\boldsymbol{x}_i\right)\right)}{\sum_{i=1}^{N} w_i}$$
   2.3 Compute
   $$\alpha_m = \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right).$$
   2.4 Set $w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot I\left(y_i \neq G_m\left(\boldsymbol{x}_i\right)\right)\right], i = 1, 2, \ldots, N.$
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x})\right]$

Boosting

- How and why AdaBoost works so well was (and in some circles is) a topic of great interest.
- Some have referred to it as the "*best off-the-shelf classifier in the world*"
- Consider a situation with $X_1, X_2, \ldots, X_{10}$ where

$$Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > \chi^2_{10}(0.5) \\ -1 & \text{otherwise} \end{cases}$$

- There are 2000 training cases, with approximately 1000 cases in each class, and 10,000 test observations.
- The weak classifier is just a "stump": a two terminal node classification tree.

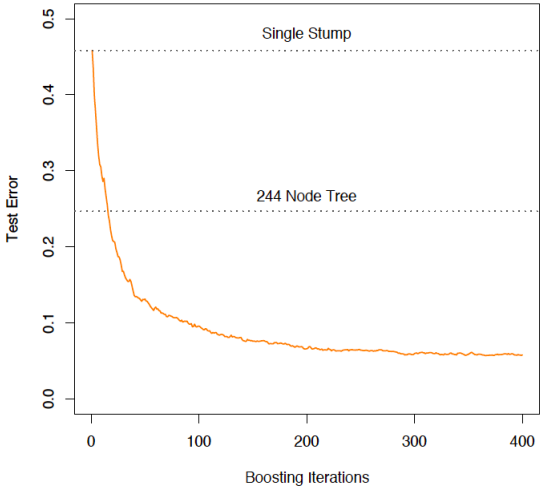


Figure: From page 340 in the online version of ESL.

Explaining Boosting

- Explaining how boosting works is important showing how we can effectively apply similar methods.
- Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions, where the basis functions are the classifiers $G_m(\boldsymbol{x})$.
- That is, our classifier is

$$f(x) = \sum_{m=1}^{M} \beta_m b\left(x; \eta_m\right)$$

  where β_m, $m = 1, 2, \ldots, M$ are the expansion coefficients (e.g., α_m) and $b(\boldsymbol{x}; \eta) \in \mathbb{R}$ is our simple classifier.
- Many classification methods can be put in this form (NN, CART, etc.).

Explaining Boosting

- These models are usually trained to minimize some loss function via

$$\min_{\{\beta_m,\eta_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b\left(x_i;\eta_m\right)\right)$$

- However, a simple alternative is often used

$$\min_{\beta,\eta} \sum_{i=1}^{N} L\left(y_i, \beta b\left(x_i;\eta\right)\right)$$

Forward Stagewise Additive Modeling

- Forward Stagewise Modeling also approximates the solution to this problem by sequentially adding new basis functions.
- This algorithm adds a new basis function and optimizes it with respect to some residual, without changing any of the previous basis functions.
- This process is repeated until some stopping criteria.

Forward Stagewise Additive Modeling

1. Initialize $f_0(\boldsymbol{x}) = 0$.
2. For $m = 1, \ldots, M$:
   2.1 Compute
   $$(\beta_m, \eta_m) = \arg\min_{\beta,\eta} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(x_i\right) + \beta b\left(x_i; \eta\right)\right).$$
   2.2 Set $f_m(x) = f_{m-1}(x) + \beta_m b\left(x; \eta_m\right)$.

Forward Stagewise Additive Modeling

- For example, suppose we're using squared error loss with

$$L(y, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2$$

then

$$\begin{aligned} L\left(y_i, f_{m-1}\left(x_i\right) + \beta b\left(x_i; \eta\right)\right) &= \left(y_i - f_{m-1}\left(x_i\right) - \beta b\left(x_i; \eta\right)\right)^2 \\ &= \left(r_{im} - \beta b\left(x_i; \eta\right)\right)^2 \end{aligned}$$

where $r_{im} = y_i - f_{m-1}\left(x_i\right)$ is the residual.

- Can anyone see why this is referred to as "forward stagewise"

Exponential Loss and AdaBoost

- The *AdaBoost.M1* algorithm is equivalent to forward stagewise model with

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

- That is, step 2.1 in the algothim is

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} \exp\left[-y_i \left(f_{m-1}\left(x_i\right) + \beta G\left(x_i\right)\right)\right]$$

or

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp\left(-\beta y_i G\left(x_i\right)\right)$$

with $w_i^{(m)} = \exp\left(-y_i f_{m-1}\left(x_i\right)\right)$.

Exponential Loss and AdaBoost

- The solution for G_m is

$$G_m = \arg\min_G \sum_{i=1}^{N} w_i^{(m)} I\left(y_i \neq G\left(x_i\right)\right)$$

and the solution for β_m is

$$\beta_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}$$

where

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I\left(y_i \neq G_m\left(x_i\right)\right)}{\sum_{i=1}^{N} w_i^{(m)}}$$

Exponential Loss and AdaBoost

- Why exponential loss?
- What does it estimate and how well is it being estimated?
- AdaBoost can be shown to be the following

$$f^*(x) = \arg\min_{f(x)} \mathrm{E}_{Y|x}\left(e^{-\mathrm{Y}f(x)}\right) = \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}$$

- It can be shown that the binomial negative log-likelihood or deviance are loss criterion with the same **population** minimizer (but not for finite data sets).

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

Boosting Trees

- As before, let's define a CART as

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I\left(x \in R_j\right)$$

  with parameters $\Theta = \{R_j, \gamma_j\}_1^J$.
- The boosted model is a sum of trees $f_M(x) = \sum_{m=1}^{M} T\left(x;\Theta_m\right)$ which can be solved with the forward stagewise procedure.
- At each step we must solve

$$\hat{\Theta}_m = \left\{\hat{R}_{jm}, \hat{\gamma}_{jm}\right\}_1^{J_m} = \arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(x_i\right) + T\left(x_i;\Theta_m\right)\right) \qquad (1)$$

Boosting Trees

- Given the regions R_m the γ_m values are the solutions to

$$\hat{\gamma}_{jm} = \arg \min_{\gamma_j \in R_{jm}} L\left(y_i, f_{m-1}\left(x_i\right) + \gamma_{jm}\right).$$

- Finding the regions to solve (1), however, is difficult (more so than before).
- For squared-error loss, however, the regression tree that best predicts the current residuals $y_i - f_{m-1}(\boldsymbol{x}_i)$ and $\hat{\gamma}_{jm}$ is the mean of these residuals in each corresponding region.
- For exponential loss it still works OK, but cannot be implemented in standard criteria.

Gradient Boosting

- To solve equation (1) with any loss function can be approximated using a weighted least squares regression tree.
- This method is derived by analogy to numerical optimization.
- The loss of $f(x)$ to predict y is

$$L(f) = \sum_{i=1}^{N} L\left(y_i, f\left(x_i\right)\right)$$

and we want $\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L(\mathbf{f})$ where the parameters can be viewed as

$$\mathbf{f} = \left\{f\left(x_1\right), f\left(x_2\right), \ldots, f\left(x_N\right)\right\}^T$$

Gradient Boosting

- Numerical optimization procedures solve $\hat{\mathbf{f}}$ as a sum of component vectors

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m, \quad \mathbf{h}_m \in \mathbb{R}^N$$

  where $\mathbf{f}_0 = \mathbf{h}_0$ is an initial guess, and each successive $\mathbf{f}_m$ is induced based on the current parameter vector $\mathbf{f}_{m-1}$.
- The methods will differ on how they compute the "step" $\mathbf{h}_m$.

Steepest Descent

- For example, steepest descent chooses $\mathbf{h}_m = -\rho_m \boldsymbol{g}_m$ where

$$g_{im} = \left[\frac{\partial L\left(y_i, f\left(x_i\right)\right)}{\partial f\left(x_i\right)} \right]_{f(x_i)=f_{m-1}(x_i)} \tag{2}$$

  where $\rho_m = \arg\min_\rho L\left(\mathbf{f}_{m-1} - \rho \mathbf{g}_m\right)$ and $\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m$.

Gradient Boosting

- The problem with steepest descent is that the gradient (2) is defined only at the training data points $\boldsymbol{x}_i$ and is thus not useful for other data.
- A solution to this is to fit a tree $T(x; \Theta_m)$ at the mth iteration whose predictions are close to the negative gradient.
- Using squared error loss, this leads to

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} \left(-g_{im} - T(x_i; \Theta)\right)^2$$

Gradients for commonly used loss functions.

- Every loss function has a corresponding gradient:

Setting	Loss Function	$-\partial L\left(y_i, f\left(x_i\right)\right) / \partial f\left(x_i\right)$
Regression	$\frac{1}{2}\left[y_i - f\left(x_i\right)\right]^2$	$y_i - f\left(x_i\right)$
Regression	$\lvert y_i - f\left(x_i\right)\rvert$	$\operatorname{sign}\left[y_i - f\left(x_i\right)\right]$
Classification	Deviance	kth component: $I\left(y_i = \mathcal{G}_k\right) - p_k\left(x_i\right)$

- All correspond to some type of residual.
- The original implementation of this algorithm was called MART for "multiple additive regression trees."

Gradient Tree Boosting Algorithm.

1. Initialize $f_0(\boldsymbol{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   - 2.1 For $i = 1, 2, \ldots, N$ compute

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

   - 2.2 Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, ..., J_m$.
   - 2.3 For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \gamma)$$

   - 2.4 Update $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \sum_{j=1}^{J_m} \gamma_{jm} I(\boldsymbol{x} \in R_{jm})$
3. Output $\hat{f}(\boldsymbol{x}) = f_M(\boldsymbol{x})$.

Gradient Boosting Method Details

- The right size to fit the trees can be chosen based on the maximum order of interactions (usually $4 \leq J \leq 8$).
- For classification, it's commonly a good idea to use a shrinkage factor ν when implementing the approach where

$$f_m(x) = f_{m-1}(x) + \nu \cdot \sum_{j=1}^{J} \gamma_{jm} I\left(x \in R_{jm}\right)$$

- When using *stochastic gradient boosting* (Friedman, 1999) a fraction of the training observations are sampled at each iteration (without replacement).

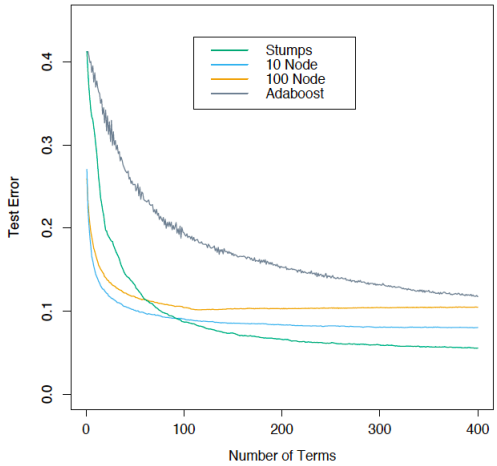


Figure: From page 363 in the online version of ESL.

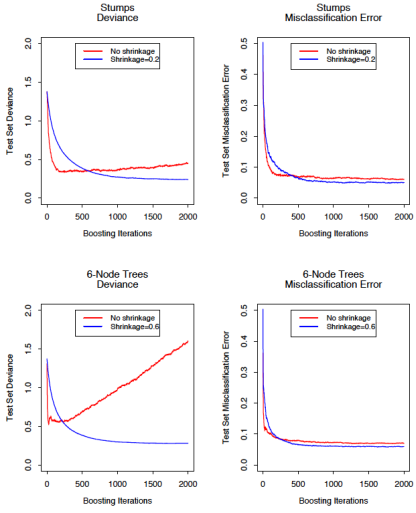


Figure: From page 366 in the online version of ESL.

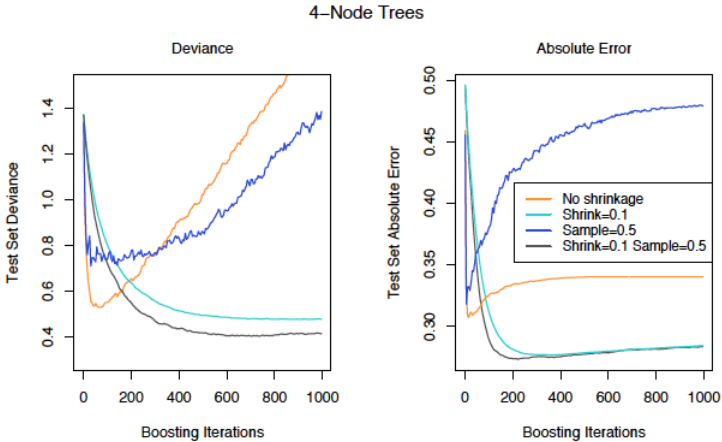


Figure: From page 367 in the online version of ESL.

Partial Dependence Plots

- Partial Dependence Plots for a group of variables $\boldsymbol{X}_{\mathcal{S}}$ can be made via

$$\overline{f}_{\mathcal{S}}(X_{\mathcal{S}}) = \frac{1}{N}\sum_{i=1}^{N} f(X_{\mathcal{S}}, x_{i\mathcal{C}})$$

  where $x_{i\mathcal{C}}$ are the values of $X_{\mathcal{C}}$ from the training set, and $X_{\mathcal{C}}$ is the complement of $\mathcal{S}$.

- Note that this estimates

$$f_{\mathcal{S}}(X_{\mathcal{S}}) = \mathrm{E}_{X_{\mathcal{C}}} f(X_{\mathcal{S}}, X_{\mathcal{C}})$$

  (averaged over) not (conditional on)

$$\tilde{f}_{\mathcal{S}}(X_{\mathcal{S}}) = \mathrm{E}(f(X_{\mathcal{S}}, X_{\mathcal{C}}) | X_{\mathcal{S}})$$

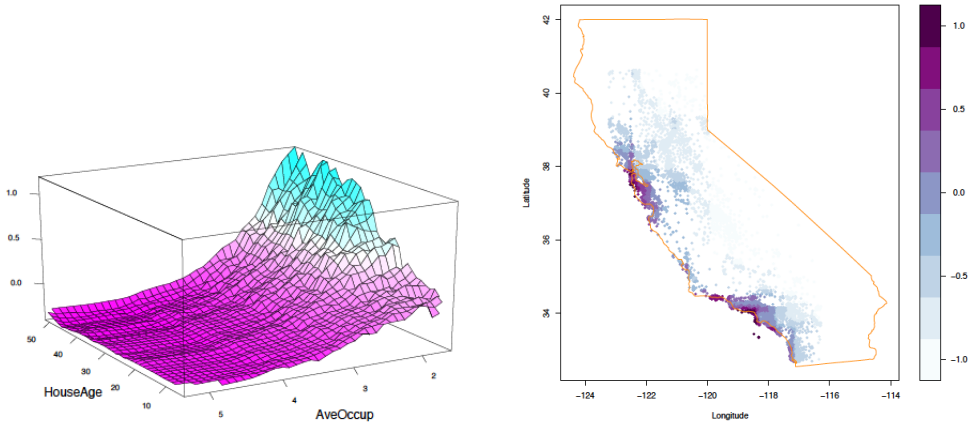


Figure: Partial dependence plots from an analysis of California housing data. From pages 374–375 in the online version of ESL.

MART versus Random Forests

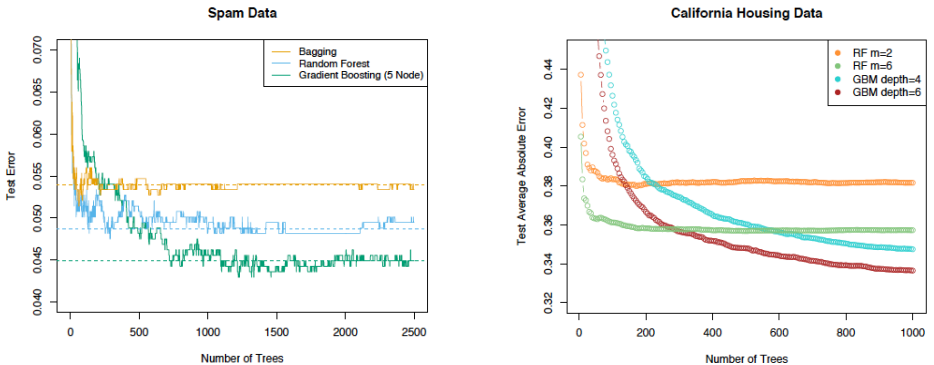


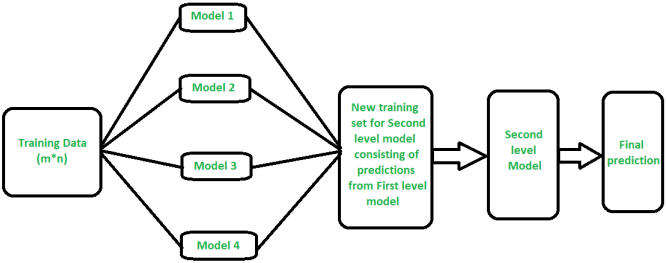
Figure: From pages 589 and 591, respectively, in the online version of ESL.

Boosting: pros and cons

- Boosting often provides higher accuracy than other algorithms, especially when combined with weak learners like decision trees.
- **Overfitting:** While boosting can overfit if run for too many iterations, it's generally less susceptible to overfitting than, say, a single decision tree, especially when weak learners are used.
- **Handles Missing Data:** Boosting algorithms (especially tree-based ones) can handle missing values without imputation.
- **Computational Cost:** Training usually takes longer because trees are built sequentially, also there are more hyperparameters to optimize.
- **Less Interpretable** partial dependence plots help, but overall hard to interpret.
- **Sensitive to Outliers:** Given that boosting tries hard to correct mistakes

Stacking

- Stacking (also known as "stacked generalization") is an ensemble learning technique that combines multiple models to produce a meta-model.
- The idea is to use predictions from individual models as input features for a higher-level model that aims to make better predictions.



Stacking: overview

1. Base Models Training:
   - Train multiple models (base models) on the training dataset.
   - These can be of diverse types, e.g., linear regression, decision trees, etc.
2. Generate Meta-Features:
   - Use the trained base models to make predictions on a validation set or out-of-bag samples.
   - These predictions are the "meta-features" for the next step.
3. Train Meta-Model:
   - Use the meta-features as input to train a higher-level model (meta-model).
   - The true target values from the validation set are used as the output.
4. Final Prediction:
   - For a new instance, first generate predictions using the base models.
   - Feed these predictions into the meta-model to get the final prediction.

Stacking: example

Say we want to predict Disease Outbreaks based on

- A dataset with weekly data from different regions.
- **Features:** average temperature, average humidity, population density, vaccination rate, cases reported in the past weeks, number of travelers entering the region, etc.
- **Target variable:** binary indicator of whether there was an outbreak (yes/no).

Base Models

- **Decision Trees:** They can capture non-linear relationships and interactions between features, such as certain combinations of weather conditions and vaccination rates.
- **Logistic Regression:** This can account for linear relationships in the data.
- **Random Forest:** To account for high-dimensionality and possible interactions between predictors.
- **SVM:** Useful when the decision boundary is not just linear or simple.

Procedure

1. Split the dataset into training, validation, and test sets.
2. Train all base models on the **training dataset**.
3. Predict the probability of an outbreak on the **validation set** using each base model.
4. The predictions from these models (for each observation in the validation set) are then combined to create a new "meta-dataset".
5. Train a meta-model, say Gradient Boosted Trees, on this meta-dataset.
6. Estimate predictive error with **test data**: first, generate predictions using all base models, then feed these into the meta-model to produce the final prediction and estimate the error.

Could be updated weakly

Stacking: overview

- By learning from the strengths of each base model, stacking often yields performance improvement over any single model.
- **Overfitting:** Since stacking involves training multiple layers of models, there's a risk of overfitting, especially if the base models are already overfitting or if the meta-model is too complex.
- **Computational Cost:** Training multiple models and then another model on top can be computationally intensive.