Natural Language Processing for Law and Social Science

4. Supervised Learning with Text

Weekly Q&A Page

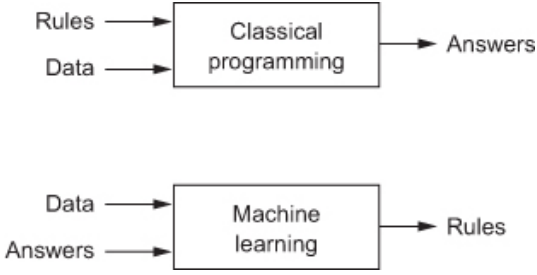
ML Essentials

- Overview
- Regression / Regularization
- Binary Classification
- Multi-Class Models

Osnabruegge, Ash, and Morelli 2021

Ensemble Learning with XGBoost

What is machine learning?



- In classical computer programming, humans input the rules and the data, and the computer provides answers.
- In machine learning, humans input the data and the answers, and the computer learns the rules.

What do ML Algorithms do? Fit a function to data points

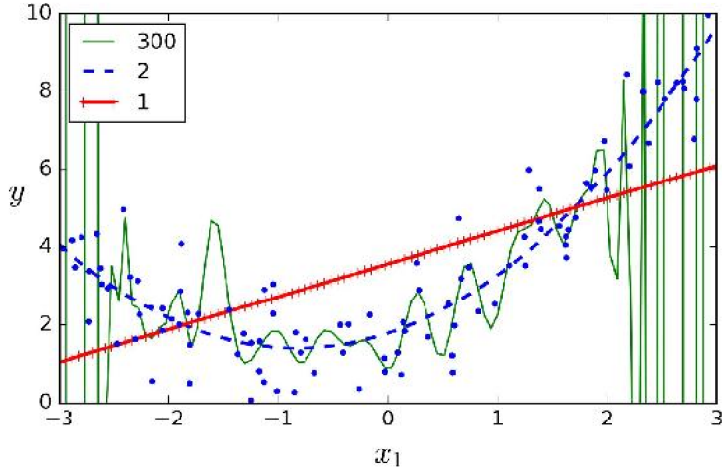


Figure 4-14. High-degree Polynomial Regression

What do ML Algorithms do? Minimize a cost function

- A typical cost function (or loss function) for regression problems is Mean Squared Error (MSE):

$$\text{MSE}(\theta) = \frac{1}{n_D} \sum_{i=1}^{n_D} (h(x_i; \theta) - y_i)^2$$

  - n_D, the number of rows/observations
  - x, the matrix of predictors, with row x_i
  - y, the vector of outcomes, with item y_i
  - $h(x_i; \theta) = \hat{y}$ the model prediction (hypothesis)

The **data** (x, y) are taken as given, and the ML algorithm searches for **parameters** θ to minimize the cost function.

Linear Regression is Machine Learning

- Ordinary Least Squares Regression (OLS) assumes the functional form $f(x;\theta) = x_i'\theta$ and minimizes the mean squared error (MSE)

$$\min_{\hat{\theta}} \frac{1}{n_D} \sum_{i=1}^{n_D} (x_i'\hat{\theta} - y_i)^2$$

Linear Regression is Machine Learning

- Ordinary Least Squares Regression (OLS) assumes the functional form $f(x;\theta) = x_i'\theta$ and minimizes the mean squared error (MSE)

$$\min_{\hat{\theta}} \frac{1}{n_D} \sum_{i=1}^{n_D} (x_i'\hat{\theta} - y_i)^2$$

- This minimand has a closed form solution

$$\hat{\theta} = (\boldsymbol{x}'\boldsymbol{x})^{-1}\boldsymbol{x}'\boldsymbol{y}$$

  - most machine learning models do **not** have a closed form solution $\rightarrow$ use numerical optimization instead (gradient descent).

$$\text{MSE}(\theta) = \frac{1}{n_D} \sum_{i=1}^{n_D} (h(\theta; \boldsymbol{x}_i) - y_i)^2$$

- The partial derivative for feature j is

$$\frac{\partial \text{MSE}}{\partial \theta_j} = \frac{2}{n_D} \sum_{i=1}^{n_D} \Big(\underbrace{h(\theta; \boldsymbol{x}_i) - y_i}_{\text{error for this obs}} \Big) \underbrace{\frac{\partial h(\theta; \boldsymbol{x}_i)}{\partial \theta_j}}_{\text{how } \theta_j \text{ shifts } h(\cdot)}$$

  - $\rightarrow$ estimates how changing θ_j would reduce the error across the whole dataset.

$$\text{MSE}(\theta) = \frac{1}{n_D} \sum_{i=1}^{n_D} (h(\theta; \boldsymbol{x}_i) - y_i)^2$$

- The partial derivative for feature j is

$$\frac{\partial \text{MSE}}{\partial \theta_j} = \frac{2}{n_D} \sum_{i=1}^{n_D} (\underbrace{h(\theta; \boldsymbol{x}_i) - y_i}_{\text{error for this obs}}) \underbrace{\frac{\partial h(\theta; \boldsymbol{x}_i)}{\partial \theta_j}}_{\text{how } \theta_j \text{ shifts } h(\cdot)}$$

  - $\rightarrow$ estimates how changing θ_j would reduce the error across the whole dataset.

- The ***gradient*** ∇ gives the vector of these partial derivatives for all features:

$$\nabla_\theta \text{MSE} = \begin{bmatrix} \frac{\partial \text{MSE}}{\partial \theta_1} \\ \frac{\partial \text{MSE}}{\partial \theta_2} \\ \vdots \\ \frac{\partial \text{MSE}}{\partial \theta_{n_x}} \end{bmatrix}$$

- ***Gradient descent*** nudges θ against the gradient (the direction that reduces MSE):

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \text{MSE}$$

- η = learning rate

$$\text{MSE}(\theta) = \frac{1}{n_D} \sum_{i=1}^{n_D} (h(\theta; \boldsymbol{x}_i) - y_i)^2$$

- The partial derivative for feature j is

$$\frac{\partial \text{MSE}}{\partial \theta_j} = \frac{2}{n_D} \sum_{i=1}^{n_D} \Big(\underbrace{h(\theta; \boldsymbol{x}_i) - y_i}_{\text{error for this obs}} \Big) \underbrace{\frac{\partial h(\theta; \boldsymbol{x}_i)}{\partial \theta_j}}_{\text{how } \theta_j \text{ shifts } h(\cdot)}$$

  - $\rightarrow$ estimates how changing θ_j would reduce the error across the whole dataset.

- The ***gradient*** ∇ gives the vector of these partial derivatives for all features:

$$\nabla_\theta \text{MSE} = \begin{bmatrix} \frac{\partial \text{MSE}}{\partial \theta_1} \\ \frac{\partial \text{MSE}}{\partial \theta_2} \\ \vdots \\ \frac{\partial \text{MSE}}{\partial \theta_{n_x}} \end{bmatrix}$$

- ***Gradient descent*** nudges θ against the gradient (the direction that reduces MSE):

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \text{MSE}$$

- η = learning rate

- If the cost function is convex, gradient descent is guaranteed to find the global minimum.

Machine Learning with Text Data

- We have a corpus (or dataset) D of $n_D \geq 1$ documents d_i (or data points).

Machine Learning with Text Data

- We have a corpus (or dataset) D of $n_D \geq 1$ documents d_i (or data points).
- Each document i has an associated outcome or label $\boldsymbol{y}_i$ with dimensions $n_y \geq 1$

Machine Learning with Text Data

- We have a corpus (or dataset) D of $n_D \geq 1$ documents d_i (or data points).
- Each document i has an associated outcome or label $\boldsymbol{y}_i$ with dimensions $n_y \geq 1$
- Some documents are labeled and some are unlabeled $\rightarrow$
  - we would like to learn a function $\hat{\boldsymbol{y}}(d_i)$ based on the labeled data ...
  - ... to machine-classify the unlabeled data.

First Problem

- Each document is a sequence of symbols d_i, while (standard) ML algorithms work on numbers.

First Problem

- Each document is a sequence of symbols d_i, while (standard) ML algorithms work on numbers.
- The solution: all the methods from Weeks 1, 2, 3 for extracting informative numerical information from documents:
  - style features
  - counts over dictionary patterns
  - tokens
  - n-grams
  - principal components
  - topic shares
  - etc.
- documents can thus be **featurized** – represented as a matrix of vectors $\boldsymbol{x}$ with $n_x \geq 1$ features.

Three Types of (Standard) Machine Learning Problems

Determined by the data type of the outcome variable (or label):

- **Binary classification**: two choices, normalized to zero and one.
  - e.g., guilty or innocent

Three Types of (Standard) Machine Learning Problems

Determined by the data type of the outcome variable (or label):

- **Binary classification**: two choices, normalized to zero and one.
  - e.g., guilty or innocent
- **Regression**: a one-dimensional, continuous, real-valued outcome.
  - e.g., number of days of prison assigned

Three Types of (Standard) Machine Learning Problems

Determined by the data type of the outcome variable (or label):

- **Binary classification**: two choices, normalized to zero and one.
  - e.g., guilty or innocent
- **Regression**: a one-dimensional, continuous, real-valued outcome.
  - e.g., number of days of prison assigned
- **Multinomial Classification**: Three or more discrete, un-ordered outcomes.
  - e.g., predict what judge is assigned to a case: Alito, Breyer, or Cardozo

Loss functions, more generally

- The loss function $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ assigns a score based on prediction and truth:
  - Should be bounded from below, with the minimum attained only for cases where the prediction is correct.
- The average loss for the test set is

$$\mathcal{L}(\theta) = \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i)$$

- The estimated parameter matrix θ solves

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta)$$

  $\hookrightarrow$ optimizes over parameter space; treats the data as constants.

Gradient Descent

- even when cost function is not convex (eg neural nets), gradient descent often gets decent results.
- ***Stochastic* gradient descent (SGD)** computes the gradient for a single randomly sampled instance (at each iteration).
  - Much faster, still works well.

Data Prep for Machine Learning

- Data Pre-Processing: See Geron Chapter 2 for pandas and sklearn syntax:
  - imputing missing values.
  - feature scaling (often helpful/necessary for ML models to work well)
    - if predictors are sparse (e.g. bag-of-words), use `StandardScaler(with_mean=False)`.
  - encoding categorical variables.
  - **Best practice: reproducible data pipeline**.

Data Prep for Machine Learning

- Data Pre-Processing: See Geron Chapter 2 for pandas and sklearn syntax:
  - imputing missing values.
  - feature scaling (often helpful/necessary for ML models to work well)
    - if predictors are sparse (e.g. bag-of-words), use `StandardScaler(with_mean=False)`.
  - encoding categorical variables.
  - **Best practice: reproducible data pipeline**.
- Train/Test Split:
  - ML models can achieve arbitrarily high accuracy in-sample, so performance should be evaluated out-of-sample.

Data Prep for Machine Learning

- Data Pre-Processing: See Geron Chapter 2 for pandas and sklearn syntax:
  - imputing missing values.
  - feature scaling (often helpful/necessary for ML models to work well)
    - if predictors are sparse (e.g. bag-of-words), use `StandardScaler(with_mean=False)`.
  - encoding categorical variables.
  - **Best practice: reproducible data pipeline**.
- Train/Test Split:
  - ML models can achieve arbitrarily high accuracy in-sample, so performance should be evaluated out-of-sample.
  - standard approach: randomly sample 80% training dataset to learn parameters, form predictions in 20% testing dataset for evaluating performance.

Use Cross-Validation During Model Training

- Within the training set:
  - Use cross-validation with grid search to get model performance metrics across subsets of data using different hyperparameter specs.
  - Find the best hyperparameters for out-of-fold prediction in the training set.
- Then evaluate model performance in the test set using these hyperparameters.

Use Cross-Validation During Model Training

- Within the training set:
  - Use cross-validation with grid search to get model performance metrics across subsets of data using different hyperparameter specs.
  - Find the best hyperparameters for out-of-fold prediction in the training set.
- Then evaluate model performance in the test set using these hyperparameters.
- Cross-validation is less common in deep learning, where training multiple models is too computationally expensive.
  - instead, use dropout and early stopping (next week).

Model Evaluation in Test Set

Evaluating a "good" model is context-dependent. Here are some basics.

Regression:

- mean squared error (MSE)
- R-squared (same ranking as MSE, but units are more interpretable)
- mean absolute error (MAE, $\sum |\hat{y}(\theta) - y|$) is less sensitive to outliers.

Model Evaluation in Test Set

Evaluating a "good" model is context-dependent. Here are some basics.

Regression:

- mean squared error (MSE)
- R-squared (same ranking as MSE, but units are more interpretable)
- mean absolute error (MAE, $\sum |\hat{y}(\theta) - y|$) is less sensitive to outliers.

Classification:

- more complicated, but accuracy is a good baseline:
  accuracy = (# correct test-set predictions) / (# of test-set observations)

Model Evaluation in Test Set

Evaluating a "good" model is context-dependent. Here are some basics.

Regression:

- mean squared error (MSE)
- R-squared (same ranking as MSE, but units are more interpretable)
- mean absolute error (MAE, $\sum |\hat{y}(\theta) - y|$) is less sensitive to outliers.

Classification:

- more complicated, but accuracy is a good baseline:
  accuracy = (# correct test-set predictions) / (# of test-set observations)
- What if one of the outcomes is over-represented – e.g., 19 out of 20? Then I can guess the modal class and get 95% accuracy.
  - Some alternative classifier metrics designed to address class imbalance (more below).

Regression models $\leftrightarrow$ Continuous outcome

- If the outcome is continuous (e.g., Y = tax revenues collected, or criminal sentence imposed in months of prison):
  - Need a regression model.
- Problems with OLS:
  - tends to over-fit training data.
  - cannot handle multicollinearity.

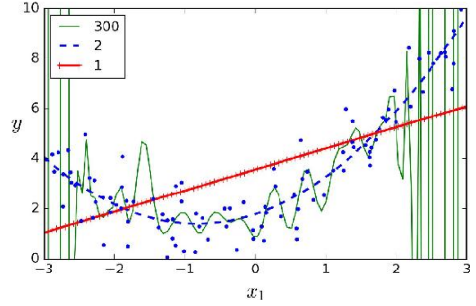


Figure 4-14. High-degree Polynomial Regression

Regression models $\leftrightarrow$ Continuous outcome

- If the outcome is continuous (e.g., Y = tax revenues collected, or criminal sentence imposed in months of prison):
  - Need a regression model.
- Problems with OLS:
  - tends to over-fit training data.
  - cannot handle multicollinearity.

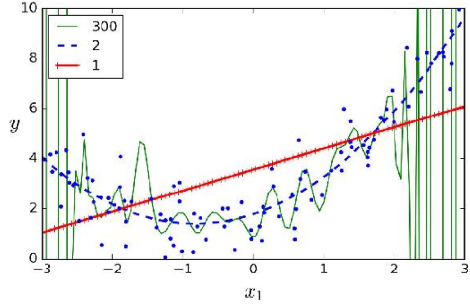


Figure 4-14. High-degree Polynomial Regression

- ***Regularization***: model training methods designed to reduce/prevent over-fitting.

Regularized Loss Function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\theta} \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i) + \lambda R(\theta)$$

- $R(\theta)$ is a "regularization function" or "regularizer", designed to reduce over-fitting.
- λ is a hyperparameter where higher values increase regularization.

Regularized Loss Function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\theta} \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i) + \lambda R(\theta)$$

- $R(\theta)$ is a "regularization function" or "regularizer", designed to reduce over-fitting.
- λ is a hyperparameter where higher values increase regularization.

In particular:

- "Lasso" (or L1) penalty:

$$R_1 = \|\theta\|_1 = \sum_{j=1}^{n_x} |\theta_j|$$

  - shrinks coefficents toward zero. automatically performs feature selection and outputs a sparse model.

Regularized Loss Function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\theta} \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i) + \lambda R(\theta)$$

- $R(\theta)$ is a "regularization function" or "regularizer", designed to reduce over-fitting.
- λ is a hyperparameter where higher values increase regularization.

In particular:

- "Lasso" (or L1) penalty:

$$R_1 = \|\theta\|_1 = \sum_{j=1}^{n_x} |\theta_j|$$

  - shrinks coefficents toward zero. automatically performs feature selection and outputs a sparse model.

- "Ridge" (or L2) penalty:

$$R_2 = \|\theta\|_2^2 = \sum_{j=1}^{n_x} (\theta_j)^2$$

  - shrinks coefficients toward zero and helps select between collinear predictors.

Regularized Loss Function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\theta} \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i) + \lambda R(\theta)$$

- $R(\theta)$ is a "regularization function" or "regularizer", designed to reduce over-fitting.
- λ is a hyperparameter where higher values increase regularization.

In particular:

- "Lasso" (or L1) penalty:

$$R_1 = \|\theta\|_1 = \sum_{j=1}^{n_x} |\theta_j|$$

  - shrinks coefficents toward zero. automatically performs feature selection and outputs a sparse model.

- "Ridge" (or L2) penalty:

$$R_2 = \|\theta\|_2^2 = \sum_{j=1}^{n_x} (\theta_j)^2$$

  - shrinks coefficients toward zero and helps select between collinear predictors.

- Elastic Net: $R_{\text{enet}} = \lambda_1 R_1 + \lambda_2 R_2$

Binary Outcome $\leftrightarrow$ Binary Classification

- Binary classifiers try to match a boolean outcome $y \in \{0, 1\}$.
  - The standard approach is to apply a transformation (e.g. sigmoid/logit) to normalize $\hat{y} \in [0, 1]$.
  - Prediction rule is 0 for $\hat{y} < .5$ and 1 otherwise.

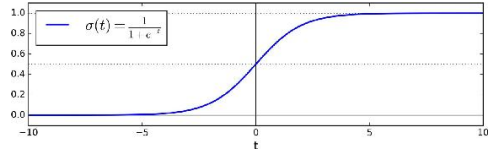
Binary Outcome $\leftrightarrow$ Binary Classification

- Binary classifiers try to match a boolean outcome $y \in \{0,1\}$.
  - The standard approach is to apply a transformation (e.g. sigmoid/logit) to normalize $\hat{y} \in [0,1]$.
  - Prediction rule is 0 for $\hat{y} < .5$ and 1 otherwise.
- The binary cross-entropy (or log loss) is:

$$L(\theta) = \underbrace{-\frac{1}{n_D}}_{\text{negative}} \sum_{i=1}^{n_D} [\underbrace{y_i}_{y_i=1} \underbrace{\log(\hat{y}_i)}_{\log \text{ prob} y_i=1} + \underbrace{(1-y_i)}_{y_i=0} \underbrace{\log(1-\hat{y}_i)}_{\log \text{ prob} y_i=0}]$$

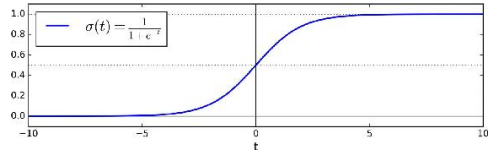
- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



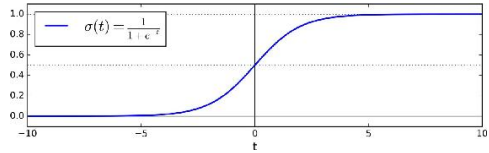
- Plugging into the binary-cross entropy loss gives the logistic regression cost objective:

$$\min_{\theta} \sum_{i=1}^{n_D} -y_i \log(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta)) - [1 - y_i] \log(1 - \text{sigmoid}(\boldsymbol{x}_i \cdot \theta))$$

  - does not have a closed form solution, but it is convex (guaranteeing that gradient descent will find the global minimum).

- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



- Plugging into the binary-cross entropy loss gives the logistic regression cost objective:

$$\min_{\theta} \sum_{i=1}^{n_D} -y_i \log(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta)) - [1 - y_i] \log(1 - \text{sigmoid}(\boldsymbol{x}_i \cdot \theta))$$

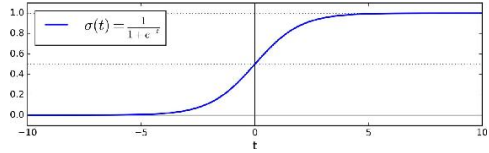
  - does not have a closed form solution, but it is convex (guaranteeing that gradient descent will find the global minimum).

- The gradient for one data point is

$$\frac{\partial L(\theta)}{\partial \theta_j} = \underbrace{(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta) - y_i)}_{\text{error for obs } i} \underbrace{x_i^j}_{\text{input } j}$$

- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



- Plugging into the binary-cross entropy loss gives the logistic regression cost objective:

$$\min_{\theta} \sum_{i=1}^{n_D} -y_i \log(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta)) - [1 - y_i] \log(1 - \text{sigmoid}(\boldsymbol{x}_i \cdot \theta))$$

  - does not have a closed form solution, but it is convex (guaranteeing that gradient descent will find the global minimum).

- The gradient for one data point is

$$\frac{\partial L(\theta)}{\partial \theta_j} = \underbrace{(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta) - y_i)}_{\text{error for obs } i} \underbrace{x_i^j}_{\text{input } j}$$

- Like linear regression, logistic regression can be regularized with L1 or L2 penalties.

A **Confusion Matrix** is a nice way to visualize classifier performance:

		Predicted Class	
		Negative	Positive
True Class	Negative	**# True Negatives**	# False Positives
	Positive	# False Negatives	**# True Positives**

- Cell values give counts in the test set.

A **Confusion Matrix** is a nice way to visualize classifier performance:

		Predicted Class	
		Negative	Positive
True Class	Negative	**# True Negatives**	# False Positives
	Positive	# False Negatives	**# True Positives**

- Cell values give counts in the test set.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives} + \text{True Negatives}}$$

A **Confusion Matrix** is a nice way to visualize classifier performance:

		Predicted Class	
		Negative	Positive
True Class	Negative	**# True Negatives**	# False Positives
	Positive	# False Negatives	**# True Positives**

- Cell values give counts in the test set.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives} + \text{True Negatives}}$$

$$\text{Precision (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- Precision decreases with false positives. "When I guess this outcome, I tend to guesses correctly."

A **Confusion Matrix** is a nice way to visualize classifier performance:

		Predicted Class	
		Negative	Positive
True Class	Negative	**# True Negatives**	# False Positives
	Positive	# False Negatives	**# True Positives**

- Cell values give counts in the test set.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives} + \text{True Negatives}}$$

$$\text{Precision (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- Precision decreases with false positives. "When I guess this outcome, I tend to guesses correctly."

$$\text{Recall (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- Recall decreases with false negatives. "When this outcome occurs, I don't miss it."

If labels are (almost) balanced, then accuracy is a decent metric.

- If not (say 90% in one category), accuracy will be uninformative/misleading.

If labels are (almost) balanced, then accuracy is a decent metric.

- If not (say 90% in one category), accuracy will be uninformative/misleading.

Balanced accuracy = the average recall in both classes:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}}\right)$$

- $\rightarrow$ equal to accuracy when classes are balanced, or when performance is the same across classes.

If labels are (almost) balanced, then accuracy is a decent metric.

- If not (say 90% in one category), accuracy will be uninformative/misleading.

Balanced accuracy = the average recall in both classes:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}}\right)$$

- $\rightarrow$ equal to accuracy when classes are balanced, or when performance is the same across classes.

F_1 **score** = the harmonic mean of precision and recall:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

- penalizes both false positives and false negatives; still ignores true negatives.

If labels are (almost) balanced, then accuracy is a decent metric.

- If not (say 90% in one category), accuracy will be uninformative/misleading.

Balanced accuracy = the average recall in both classes:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}}\right)$$

- $\rightarrow$ equal to accuracy when classes are balanced, or when performance is the same across classes.

F_1 **score** = the harmonic mean of precision and recall:

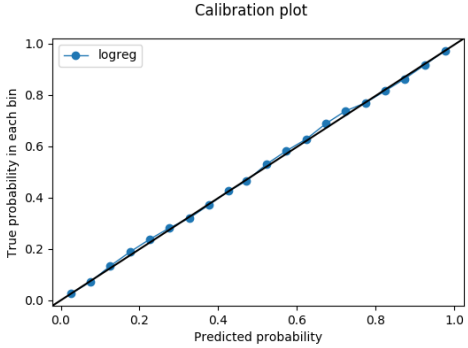
$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

- penalizes both false positives and false negatives; still ignores true negatives.

AUC-ROC = Area Under the Receiver Operating Characteristic Curve

- provides an aggregate measure of performance across all possible classification thresholds.
- Interpretation: randomly sample one positive and one negative example. AUC = probability that the model correctly guesses which is which.

Evaluating Classification Models: Calibration Curves



- Plotting the binned fraction in a category (Y axis) against the predicted probability in a category (X axis):
- Provides evidence of whether the classifer is replicating the conditional distribution of the outcome.

```
from seaborn import regplot
regplot(y_test, y_pred, x_bins=20)
```

Application: Predicting Political Party from Text

Andrew Peterson and Arthur Spirling, "Classification accuracy as a substantive quantity of interest: Measuring polarization in Westminster systems," *Political Analysis* (2018).

Application: Predicting Political Party from Text

Andrew Peterson and Arthur Spirling, "Classification accuracy as a substantive quantity of interest: Measuring polarization in Westminster systems," *Political Analysis* (2018).

- Machine Learning Problem:
  - Corpus $D = 3.5$M U.K. parliament speeches, 1935-2013.

Application: Predicting Political Party from Text

Andrew Peterson and Arthur Spirling, "Classification accuracy as a substantive quantity of interest: Measuring polarization in Westminster systems," *Political Analysis* (2018).

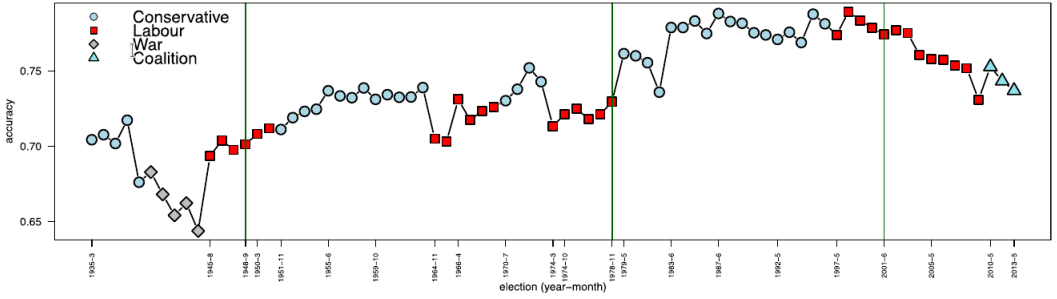
- Machine Learning Problem:
  - Corpus D = 3.5M U.K. parliament speeches, 1935-2013.
  - Label Y = party of speaker (Conservative or Labour)

Application: Predicting Political Party from Text

Andrew Peterson and Arthur Spirling, "Classification accuracy as a substantive quantity of interest: Measuring polarization in Westminster systems," *Political Analysis* (2018).

- Machine Learning Problem:
  - Corpus $D = 3.5$M U.K. parliament speeches, 1935-2013.
  - Label $Y =$ party of speaker (Conservative or Labour)

In years that classifier is more accurate, speech is more polarized:



Multiple Classes: Setup

- The outcome is $y_i \in \{1, ..., k, ..., n_y\}$ output classes, which can also be represented as a one-hot vector

$$\boldsymbol{y}_i = \{\mathbf{1}[y_i = 1], ..., \mathbf{1}[y_i = n_y]\}$$

Multiple Classes: Setup

- The outcome is $y_i \in \{1, ..., k, ..., n_y\}$ output classes, which can also be represented as a one-hot vector

$$\boldsymbol{y}_i = \{\mathbf{1}[y_i = 1], ..., \mathbf{1}[y_i = n_y]\}$$

- We want to learn a vector function

$$\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}, \theta)$$

taking text features $\boldsymbol{x}$ as inputs and outputing a vector of probabilities across outcome classes:

$$\hat{\boldsymbol{y}} = \{\hat{y}^1, ..., \hat{y}^{n_y}\}, \sum_{k=1}^{n_y} \hat{y}^k = 1, \hat{y}^k \geq 0 \ \forall k$$

- for prediction step, can select the highest-probability class:

$$\tilde{y} = \arg\max_k \hat{y}_{[k]}$$

Categorical Cross Entropy

- The standard loss function in multinomial classification is **categorical cross entropy**:

$$L(\theta) = -\sum_{k=1}^{n_y} \boldsymbol{y}^k \log(\hat{y}^k(\boldsymbol{x}, \theta))$$

  - measures dissimilarity between the true label distribution $\boldsymbol{y}$ and the predicted label distribution $\hat{\boldsymbol{y}}$.

Categorical Cross Entropy

- The standard loss function in multinomial classification is **categorical cross entropy**:

$$L(\theta) = -\sum_{k=1}^{n_y} \boldsymbol{y}^k \log(\hat{y}^k(\boldsymbol{x}, \theta))$$

  - measures dissimilarity between the true label distribution $\boldsymbol{y}$ and the predicted label distribution $\hat{\boldsymbol{y}}$.

- Since there is just one true class ($y = 1$ for one class k^*, and zero for others), simplifies to

$$L(\theta) = -\log(\hat{y}^{k^*}(\boldsymbol{x}, \theta))$$

  - Rewards putting higher probability on the true class, ignores distribution of probabilities on other classes.
  - function is convex $\rightarrow$ gradient descent will find the optimum.

Multinomial Logistic Regression

Multinomial logistic regression computes probabilities for each class k using the softmax transformation

$$\hat{y}_k(\boldsymbol{x}_i) = \Pr(y_i = k) = \frac{\exp(\theta_k' \boldsymbol{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta_l' \boldsymbol{x}_i)}$$

- softmax is the multiclass generalization of sigmoid $\rightarrow$ can then interpret $\hat{y}$ as probabilities.
- n_x features and n_y output classes $\rightarrow$ there is a $n_y \times n_x$ parameter matrix Θ, where the parameters for each class θ_k are stored as rows.

Multinomial Logistic Regression

Multinomial logistic regression computes probabilities for each class k using the softmax transformation

$$\hat{y}_k(\boldsymbol{x}_i) = \Pr(y_i = k) = \frac{\exp(\theta_k' \boldsymbol{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta_l' \boldsymbol{x}_i)}$$

- softmax is the multiclass generalization of sigmoid $\rightarrow$ can then interpret $\hat{y}$ as probabilities.
- n_x features and n_y output classes $\rightarrow$ there is a $n_y \times n_x$ parameter matrix Θ, where the parameters for each class θ_k are stored as rows.

The **L2-penalized logistic regression** has loss function

$$\mathcal{L}(\theta) = -\frac{1}{n_D} \sum_{i=1}^{n_D} \log \frac{\exp(\theta_{k^*}' \boldsymbol{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta_l' \boldsymbol{x}_i)} + \lambda \sum_{j=1}^{n_x} \sum_{k=1}^{n_y} (\theta_{[j,k]})^2$$

- λ = strength of L2 penalty (could also add lasso penalty)
  - as before, predictors should be scaled to the same variance.

		Predicted Class		
		Class A	Class B	Class C
True Class	Class A	Correct A	A, classed as B	A, classed as C
	Class B	B, classed as A	Correct B	B, classed as C
	Class C	C, classed as A	C, classed as B	Correct C

More generally, with **multi-class confusion matrix** M with items M_{ij} (row i, column j):

$$\text{Precision for } k = \frac{\text{True Positives for } k}{\text{True Positives for k} + \text{False Positives for } k} = \frac{M_{kk}}{\sum_l M_{lk}}$$

$$\text{Recall for } k = \frac{\text{True Positives for } k}{\text{True Positives for } k + \text{False Negatives for } k} = \frac{M_{kk}}{\sum_l M_{kl}}$$

$$F_1(k) = 2 \times \frac{\text{precision}(k) \times \text{recall}(k)}{\text{precision}(k) + \text{recall}(k)}$$

		Predicted Class		
		Class A	Class B	Class C
True Class	Class A	Correct A	A, classed as B	A, classed as C
	Class B	B, classed as A	Correct B	B, classed as C
	Class C	C, classed as A	C, classed as B	Correct C

More generally, with **multi-class confusion matrix** M with items M_{ij} (row i, column j):

$$\text{Precision for } k = \frac{\text{True Positives for } k}{\text{True Positives for k} + \text{False Positives for } k} = \frac{M_{kk}}{\sum_l M_{lk}}$$

$$\text{Recall for } k = \frac{\text{True Positives for } k}{\text{True Positives for } k + \text{False Negatives for } k} = \frac{M_{kk}}{\sum_l M_{kl}}$$

$$F_1(k) = 2 \times \frac{\text{precision}(k) \times \text{recall}(k)}{\text{precision}(k) + \text{recall}(k)}$$

Can average these metrics across classes to get aggregate metrics.

- e.g., balanced accuracy = unweighted average of recalls across classes.
- can weight classes by their frequency in dataset

Cross-Domain (Transfer) Learning

- A recent but now widespread approach to machine learning is **transfer learning**:
  - train a model in a big labeled dataset
  - apply in a smaller (mostly) unlabeled dataset

Cross-Domain (Transfer) Learning

- A recent but now widespread approach to machine learning is **transfer learning**:
  - train a model in a big labeled dataset
  - apply in a smaller (mostly) unlabeled dataset
- In NLP:
  - transfer learning is intuitive because NLP tasks share common knowledge about language.
  - labeled data is scarce/expensive, so learn tasks on tons of unlabeled data.
  - reflected in success of pre-trained models, e.g. BERT and GPT.

Osnabruegge, Ash, and Morelli 2021

This paper takes the idea of transfer learning to the political science context.

- Learn to predict political topics from text in a labeled corpus (party manifestos from Comparative Manifesto Project)

Osnabruegge, Ash, and Morelli 2021

This paper takes the idea of transfer learning to the political science context.

- Learn to predict political topics from text in a labeled corpus (party manifestos from Comparative Manifesto Project)
- Apply model to classify topics in unlabeled corpus (parliamentary speeches).

Osnabruegge, Ash, and Morelli 2021

This paper takes the idea of transfer learning to the political science context.

- Learn to predict political topics from text in a labeled corpus (party manifestos from Comparative Manifesto Project)
- Apply model to classify topics in unlabeled corpus (parliamentary speeches).
- Use for empirical analysis of electoral institutions and speech content.

Overview of Text Analysis Methods (Osnabruegge et al 2021

	Dictionaries (Custom)	Dictionaries (Generic)	Topic Modeling	Within-Domain Supervised Learning	Cross-Domain Supervised Learning
Design/Annotation Costs	High	Low	Low	High	Moderate
Specificity	High	Moderate	Low	High	Moderate
Interpretability	High	High	Moderate	High	High
Validatability	Low	Low	Low	High	High

Widmer, Galletta, and Ash 2022

Another transfer learning paper:

- Learn to predict the probability that a document comes from right-wing cable news (Fox) or left-wing cable news (MSNBC/CNN)

Widmer, Galletta, and Ash 2022

Another transfer learning paper:

- Learn to predict the probability that a document comes from right-wing cable news (Fox) or left-wing cable news (MSNBC/CNN)
- Apply model to local newspaper articles to measure influence of cable channels in local news markets.

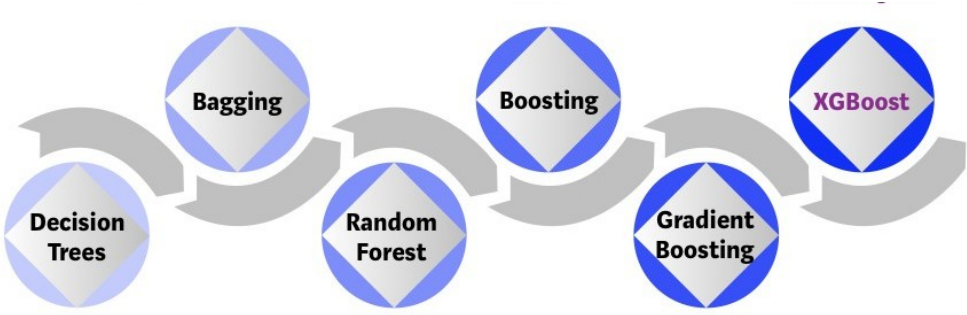
Widmer, Galletta, and Ash 2022

Another transfer learning paper:

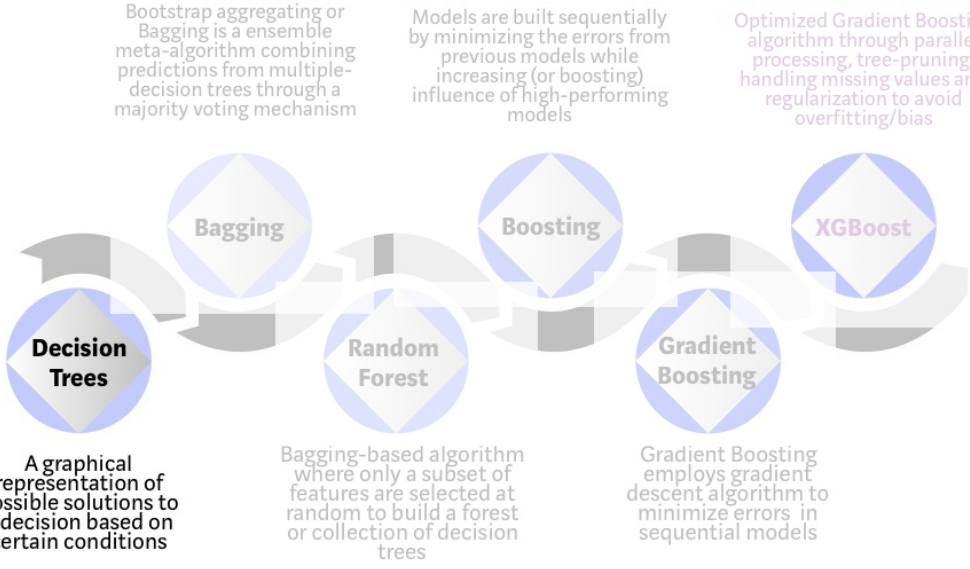
- Learn to predict the probability that a document comes from right-wing cable news (Fox) or left-wing cable news (MSNBC/CNN)
- Apply model to local newspaper articles to measure influence of cable channels in local news markets.
- Use for empirical analysis of the cable news viewership and local news content.

Group Activity

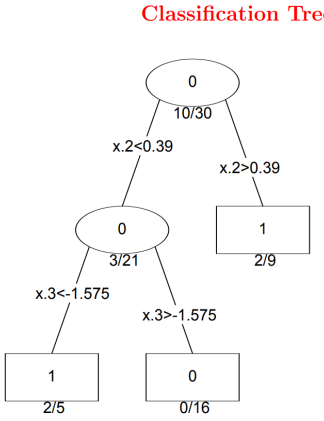
XGBoost: Overview



XGBoost Ingredients: Decision Trees

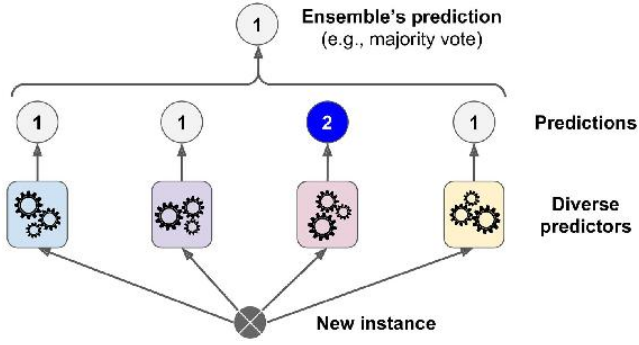


Decision Trees



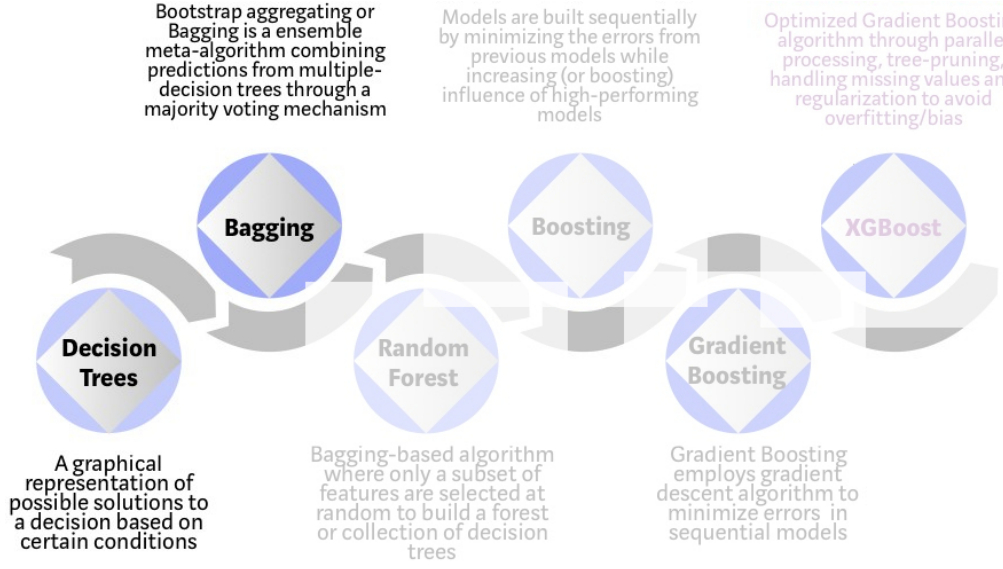
- Decision trees learn a series of binary splits in the data based on hard thresholds.
  - if yes, go right; if no, go left.
- Can have additional splits as you move through the tree.
- fast and interpretable, but performance is often poor.

Voting Classifiers



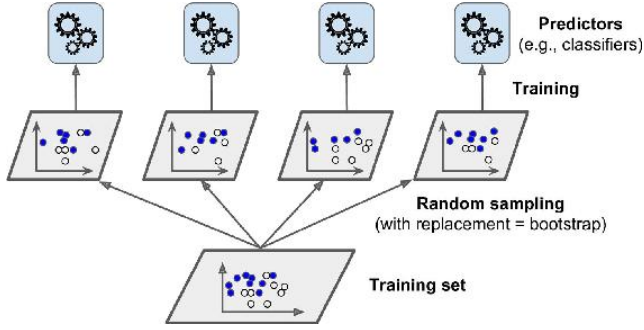
- voting classifiers (ensembles of different models that vote on the prediction) generally out-perform the best classifier in the ensemble.
  - more diverse algorithms will make different types of errors, and improve your ensemble's robustness.

XGBoost Ingredients: Bootstrapping



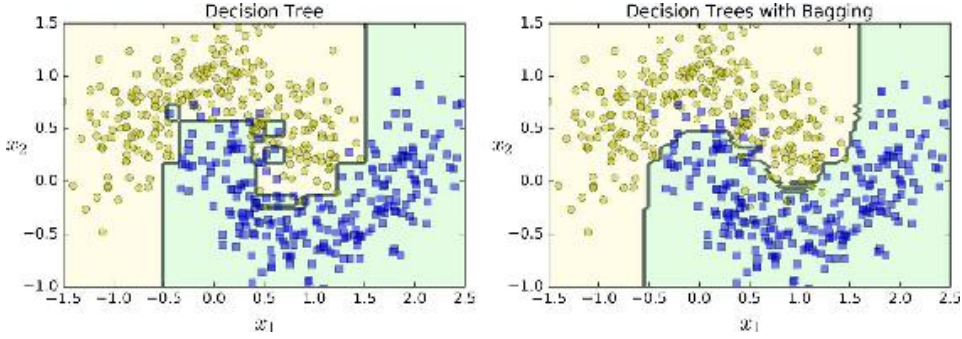
Bootstrapping

- Rather than use the same data on different classifiers, one can use different subsets of the data on the same classifier:



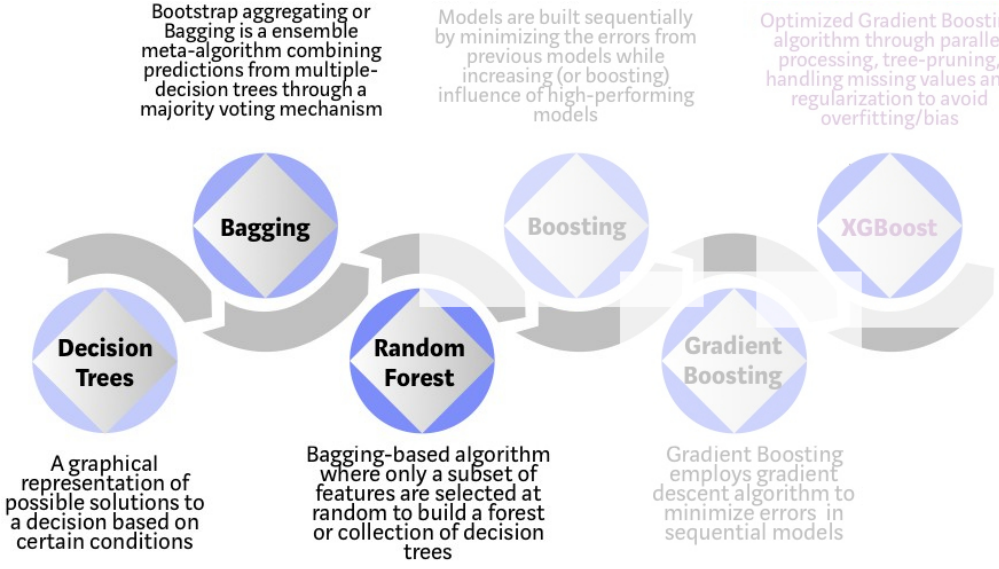
  - can also use different subsets of features across subclassifiers.

Bootstrapping Benefits



- A bootstraped ensemble generally has a similar bias but lower variance than a single predictor trained on all the data.
- Predictors can be trained in parallel using separate CPU cores.

XGBoost Ingredients: Random Forests



Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X,y)
```

Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X,y)
```

1. Each voting tree gets its own sample of data.

Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X,y)
```

1. Each voting tree gets its own sample of data.
2. At each tree split, a random sample of features is drawn, only those features are considered for splitting.

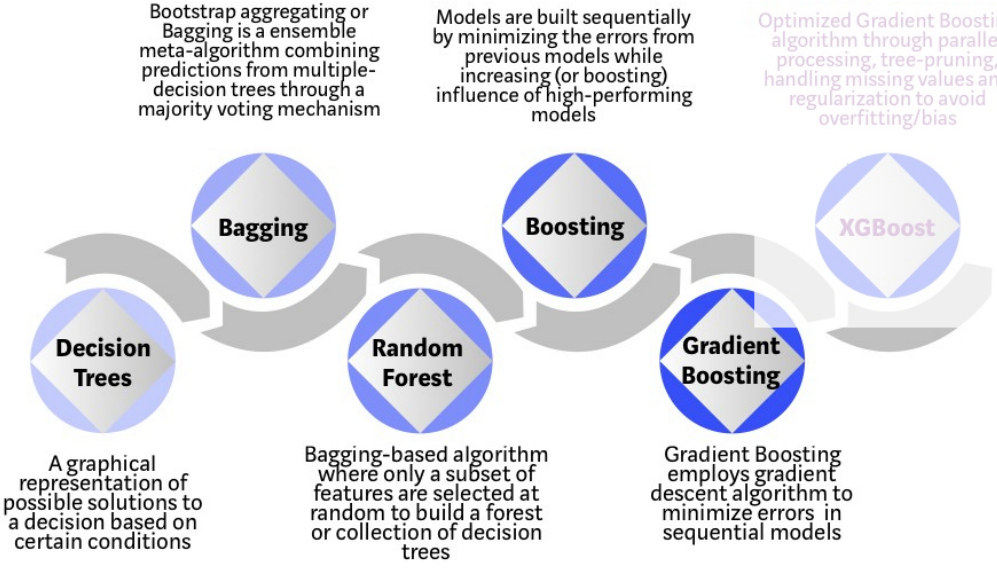
Random Forests

Random Forests are optimized ensembles of bootstrapped decision trees:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X,y)
```

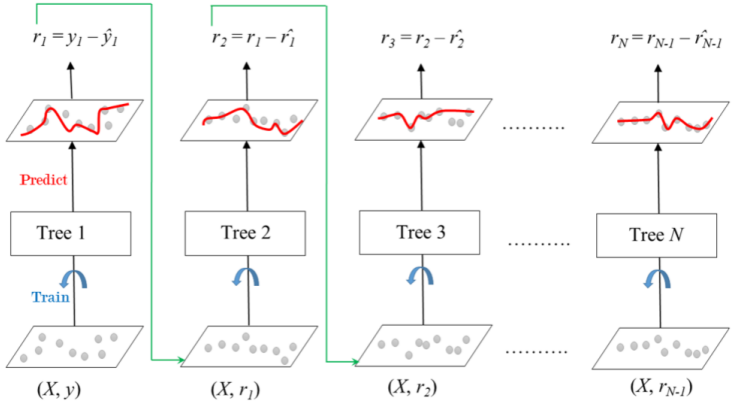
1. Each voting tree gets its own sample of data.
2. At each tree split, a random sample of features is drawn, only those features are considered for splitting.
3. For each tree, error rate is computed using data outside its bootstrap sample.

XGBoost Ingredients: Gradient Boosting



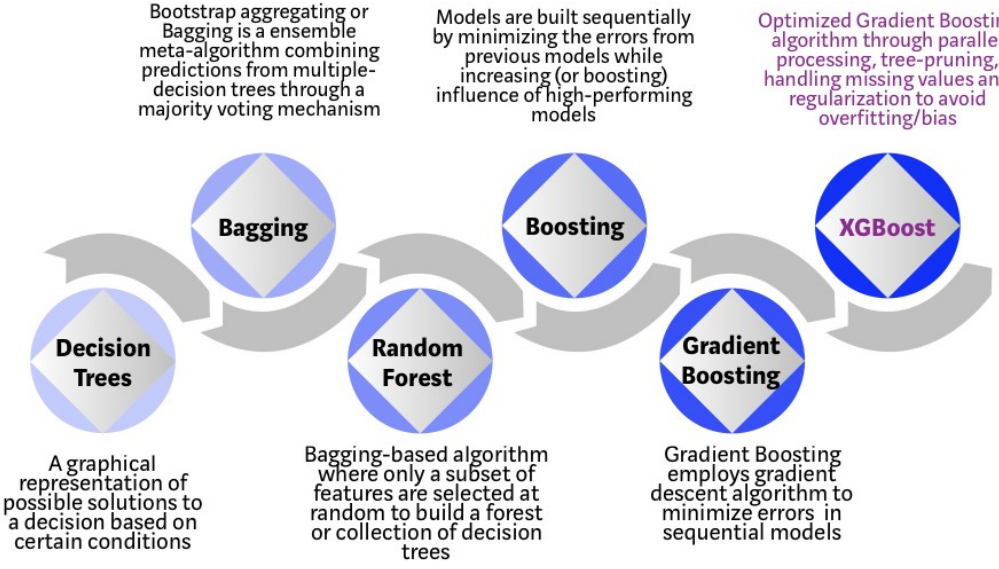
Gradient Boosting Machines

- Gradient boosting refers to an additive ensemble of trees:



- Adds additional layers of trees to fit the residuals of the first layers

XGBoost Ingredients



XGBoost

- Feurer et al (2018) find that XGBoost beats a sophisticated AutoML procedure with grid search over 15 classifiers and 18 data preprocessors.
- A good starting point for any machine learning task.

- easy to use
- actively developed
- efficient / parallelizable
- provides model explanations
- takes sparse matrices as input

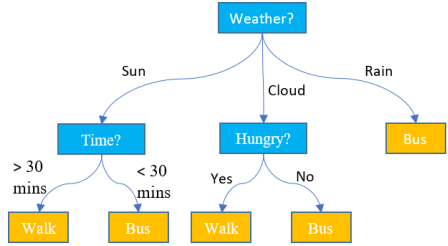
```
from xgboost import XGBClassifier
model = XGBClassifier()

model.fit(X_train, y_train,
          early_stopping_rounds=10,
          eval_metric="logloss",
          eval_set=[(X_eval, y_eval)]
          )

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

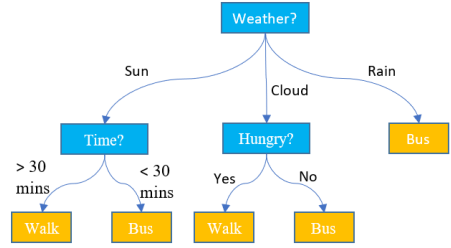
Tree Ensembles are Black Boxes

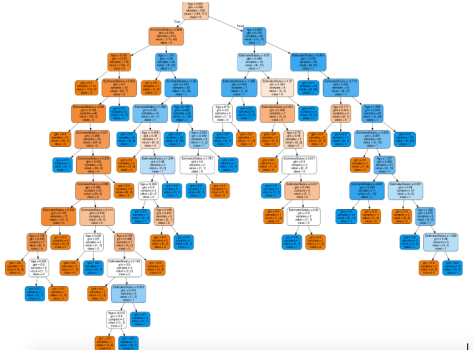
- Small decision trees have the advantage of being highly interpretable.



Tree Ensembles are Black Boxes

- Small decision trees have the advantage of being highly interpretable.





- Larger trees and ensembles (e.g. XGBoost) lose this nice feature.
- Best-performing ML models are hard to interpret because they use lots of features and exploit non-linearities and interactions.

Interpreting Tree Ensembles

XGBoost's Feature Importance Metric:

- At each decision node, compute **information gain** for feature j **(change in predicted probability**).
- Average across all nodes for each j.

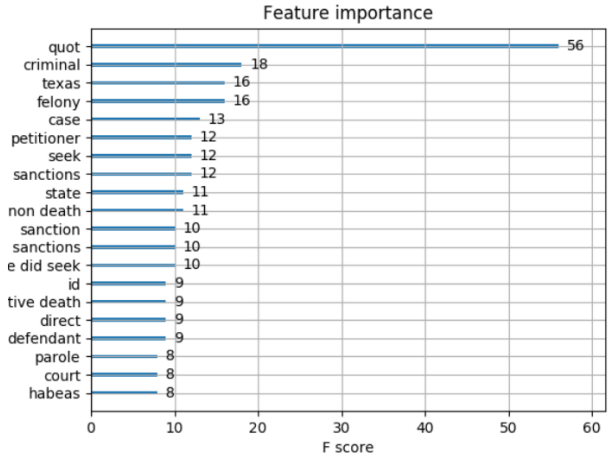
Ranks predictors by their relative contributions.

```
from xgboost import plot_importance
plot_importance(xgb_reg, max_num_features=10)
```

Feature Importance

```
from xgboost import plot_importance
plot_importance(xgb_reg, max_num_features=20)
```

```
<IPython.core.display.Javascript object>
```



- XGBoost provides a metric of feature importance that summarizes how well each feature contributes to predictive accuracy.

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.
   - *(If y is more complicated, e.g. a sequence of words, we use deep learning.)*

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.
   - *(If y is more complicated, e.g. a sequence of words, we use deep learning.)*
3. Use cross-validation grid search in training set to select model hyperparameters.
   - For classification, use cross entropy; for regression, use mean squared error.

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.
   - *(If y is more complicated, e.g. a sequence of words, we use deep learning.)*
3. Use cross-validation grid search in training set to select model hyperparameters.
   - For classification, use cross entropy; for regression, use mean squared error.
4. Evaluate model in held-out test set:
   - For classification, use balanced accuracy, confusion matrix, and calibration plot.
   - For regression, use R squared and binscatter plot.

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.
   - *(If y is more complicated, e.g. a sequence of words, we use deep learning.)*
3. Use cross-validation grid search in training set to select model hyperparameters.
   - For classification, use cross entropy; for regression, use mean squared error.
4. Evaluate model in held-out test set:
   - For classification, use balanced accuracy, confusion matrix, and calibration plot.
   - For regression, use R squared and binscatter plot.
5. Interpret the model predictions:
   - for gradient boosting, use feature importance ranking.
   - for linear models, examine coefficients
   - look at highest and lowest ranked documents for $\hat{y}$

A baseline for machine learning using text

1. Take POS-filtered bigrams as inputs X.
2. Select a machine learning model for predicting outcome y:
   - For regression (y is one-dimensional and continuous), elastic net or gradient boosted regressor.
   - For classification (y is discrete), L2-penalized logistic regression or gradient boosted classifier.
   - *(If y is more complicated, e.g. a sequence of words, we use deep learning.)*
3. Use cross-validation grid search in training set to select model hyperparameters.
   - For classification, use cross entropy; for regression, use mean squared error.
4. Evaluate model in held-out test set:
   - For classification, use balanced accuracy, confusion matrix, and calibration plot.
   - For regression, use R squared and binscatter plot.
5. Interpret the model predictions:
   - for gradient boosting, use feature importance ranking.
   - for linear models, examine coefficients
   - look at highest and lowest ranked documents for $\hat{y}$
6. Answer the research question!