Building a Robot Judge: Data Science for Decision-Making

4. Classification and XGBoost

If we have any new students today, please say "Hello" in the chat.

HW3: News Articles on Correlation/Causation

- **https://www.yahoo.com/lifestyle/vitamin-could-lower-risk-covid-151539594.html**
- https://www.cnbc.com/2018/07/03/drinking-coffee-could-help-you-live-longer-study-says.html
- https://jezebel.com/soda-totally-turns-teens-into-killers-5853062
- https://www.latimes.com/archives/la-xpm-2010-dec-16-la-he-autism-20101217-story.html
- https://well.blogs.nytimes.com/2008/01/31/diehard-sports-fans-face-heart-risk/

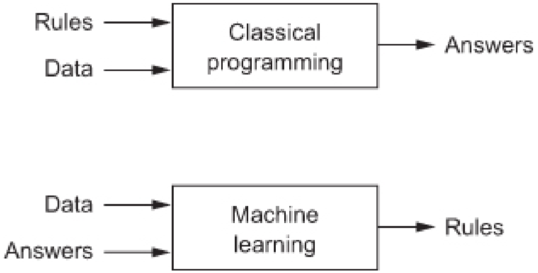
Activity: Private Zoom Chat (3 minutes)

- Imagine that cantons Zurich and Zug each enact a tax cut and you estimate a negative effect on local employment using **fixed effects regression**. What are some potential confounding factors that would bias this estimate?
  - chat answers to me privately by zoom.

Learning Objectives

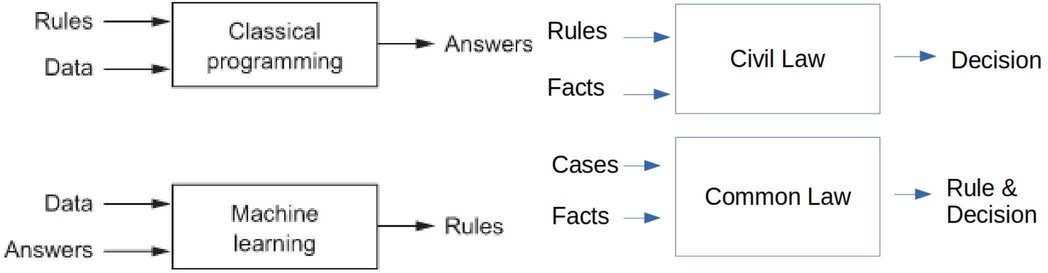
1. **Implement and evaluate machine learning pipelines.**
    - **Evaluate (find problems in) existing machine learning pipelines.**
    - **Design a pipeline to solve a given ML problem.**
    - **Implement some standard pipelines in Python.**
2. Implement and evaluate causal inference designs.
3. Understand how (not) to use data science tools (ML and CI) to support expert decision-making.

What is machine learning?



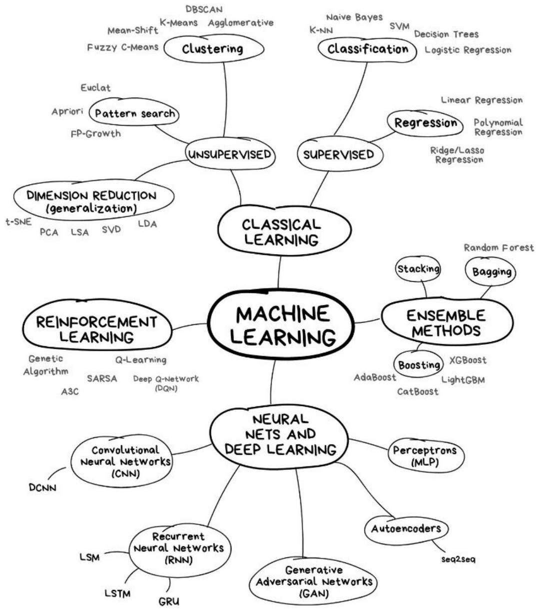
- In classical computer programming, humans input the rules and the data, and the computer provides answers.
- In machine learning, humans input the data and the answers, and the computer learns the rules.

A conceptual parallel

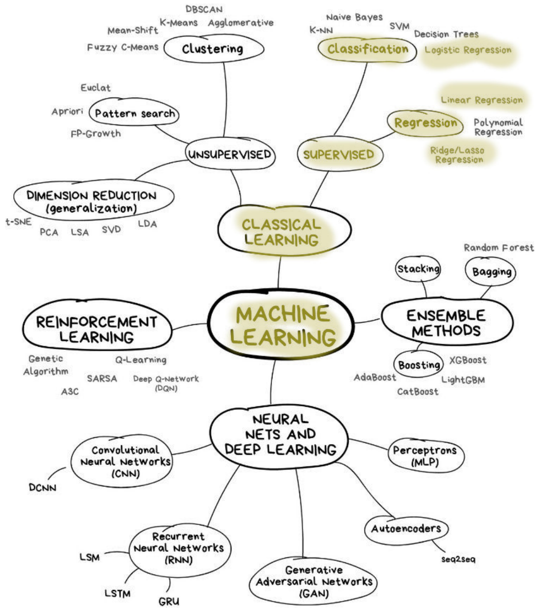


- In the civil law (code-based decision-making), a judge takes the statutory rules as given, applies them to the facts of a case, and makes a decision.
- In the common law (case-based decision-making), a judge takes the facts of a case, compares them to facts and decisions in previous cases, and infers a rule to make a new decision.

The Machine Learning Landscape



What we have done so far



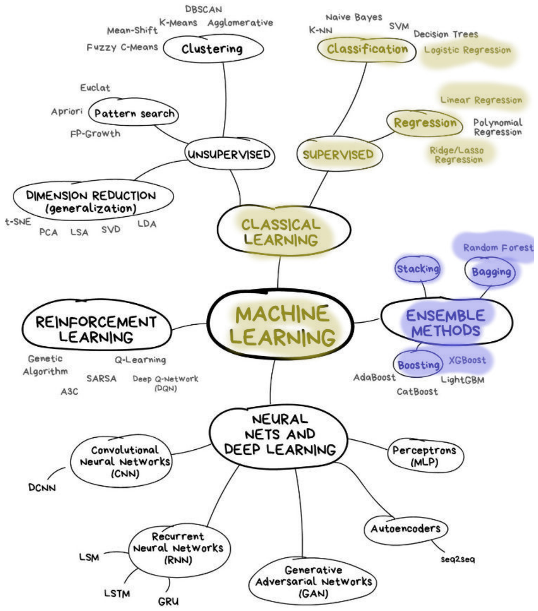
Bug Hunt: What line # has the problem?

Zoom Poll 4.1

```
1 X_train, X_test, y_train, y_test = train_test_split(X,y_true)
2 lasso = Lasso()
3 lasso.fit(X_train, y_train)
4 y_pred = lasso.predict(X)
5 mae = mean_absolute_error(y_true,y_pred)
```

- Note: mean absolute error ($\sum |y - \hat{y}|$) is an alternative regression metric that is less sensitive to outliers than mean squared error ($\sum (y - \hat{y})^2$).

What we will do today



Machine learning metrics

- For regression, evaluation is relatively straightforward – we have mean squared error, mean absolute error, or R-squared (see Week 2 slides).
- For classification, things are more complicated.
- As a (good) baseline, we have:
  - accuracy = (# correct test-set predictions) / (# of test-set observations)
- But what if one of the outcomes is rare – e.g., one out of 20?
  - Then I can guess the modal class and get 95% accuracy.
- As we will see, there are a range of other useful metrics besides accuracy for evaluating classifier performance.

Confusion Matrix

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.

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

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

Precision and Recall

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

```
from sklearn.metrics import precision_score, recall_score
precision_score(y_test,y_pred)
recall_score(y_test,y_pred)
```

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

- Precision decreases with false positives. “When I guess this outcome, I tend to guess 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.”

Note: sckit-learn will produce precision/recall for both classes by default.

Trading off precision and recall

- Recall our decision framework:
  - a decision-maker who has to make a decision W, that will produce some value or benefit, conditional on the value of Y:
$$V = u(W, Y)$$
- One can imagine decision contexts where precision/recall should be valued asymmetrically. This is part of the function $u(W, Y)$.
- **Zoom Poll 4.2: How to weight precision or recall.**

Balanced Accuracy and F1 Score

- If labels are (almost) balanced, then accuracy (share correct predictions) is a decent metric.
  - If not (say 90% in one category), accuracy will be uninformative/misleading.
- A standard metric in this case is **balanced accuracy** = the average recall in both classes:

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

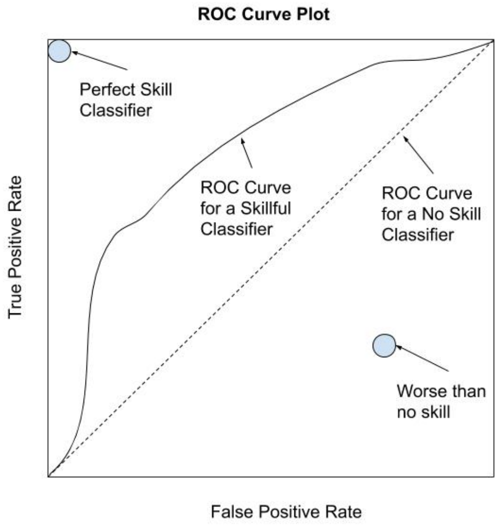
  - $\rightarrow$ equal to accuracy when classes are balanced, or performance is the same across classes.
- Another standard metric is F_1 score = the harmonic mean of precision and recall:

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

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

```
from sklearn.metrics import balanced_accuracy_score, f1_score
balanced_accuracy_score(y_test,y_pred)
f1_score(y_test,y_pred) # gives values for both classes
```

ROC Curve and AUC



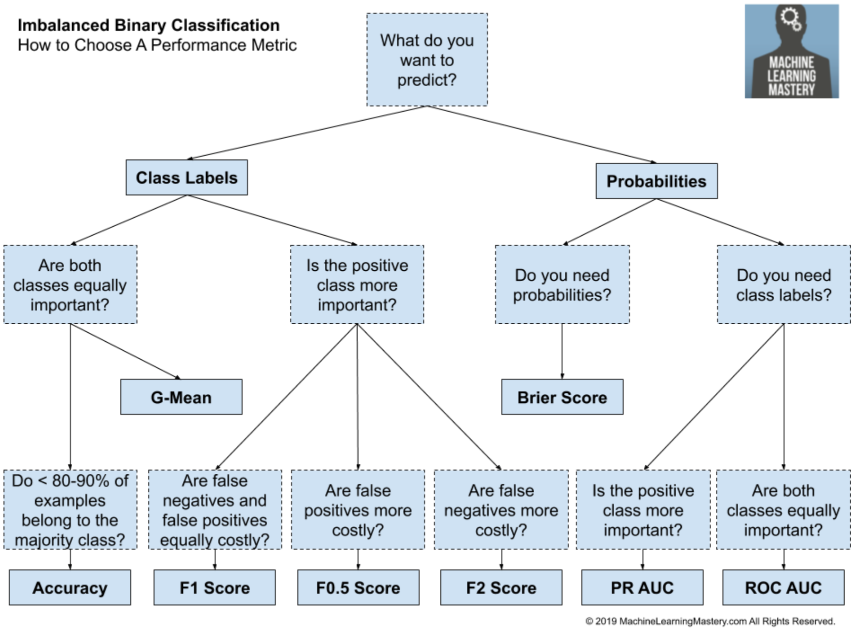
AUC = area under the (ROC) curve

- provides an aggregate measure of performance across all possible classification thresholds.

Interpreting AUC:

- = probability that the model (correctly) ranks a random positive example more highly than a random negative example.

```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred)
```


Imbalanced Binary Classification
How to Choose A Performance Metric
What do you want to predict?
Class Labels
Probabilities
Are both classes equally important?
Is the positive class more important?
Do you need probabilities?
Do you need class labels?
G-Mean
Brier Score
Do < 80-90% of examples belong to the majority class?
Are false negatives and false positives equally costly?
Are false positives more costly?
Are false negatives more costly?
Is the positive class more important?
Are both classes equally important?
Accuracy
F1 Score
F0.5 Score
F2 Score
PR AUC
ROC AUC
MACHINE LEARNING MASTERY
© 2019 MachineLearningMastery.com All Rights Reserved.

See details in "Tour of evaluation metrics for imbalanced classification" (on syllabus).

Breakout Rooms: Binary Classification Practice

- See link to python file template in zoom chat.
  - recommended: collaborate using atom teletype portal
  - alternative 1: work together on a code share (codeshare.io)
  - alternative 2: one person codes and share screen
- Will post solved PY file after lecture.

Multi-Class Models

Many interesting machine learning problems involve multiple un-ordered categories:

- categorizing a case by area of law.
- predicting the political party of a speaker in a multi-party system.
- predicting authorship from documents
- image classification / object detection

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

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

outputting a vector of probabilities across outcomes as a function of the inputs:

$$\hat{\boldsymbol{y}} = \{\hat{y}^1, ..., \hat{y}^{n_y}\}, \hat{y}^k \in [0, 1] \; \forall k$$

  - for prediction, 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**:

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

  - 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

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

  - 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. 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 prediction $y_i \in \{1, ..., n_y\}$ is determined by the highest-probability category.

Regularized Multinomial Logistic

- The L2-penalized logistic regression loss (the default in sklearn) is

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

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

```
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression()
logit.fit(X,y)
```

  - if y is categorical, sklearn automatically uses multinomial logistic

Multi-Class Confusion Matrix

		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, can have a confusion matrix M with items M_{ij} (row i, column j).

Multi-Class Performance Metrics

Confusion matrix M with items M_{ij} (row i, column j).

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

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

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

- in sklearn, syntax is the same as binary case.

Metrics for whole model

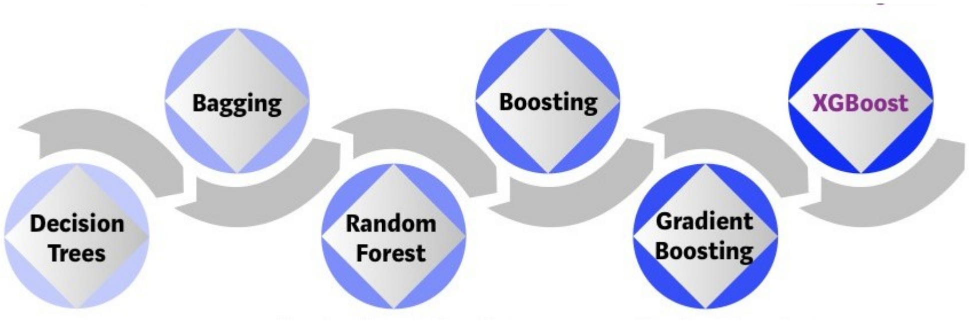
$$\text{Balanced Accuracy} = \frac{1}{n_y} \sum_k \text{Recall for } k$$

- **Macro-averaging**: average of the per-class precision, recall, and F1, e.g.

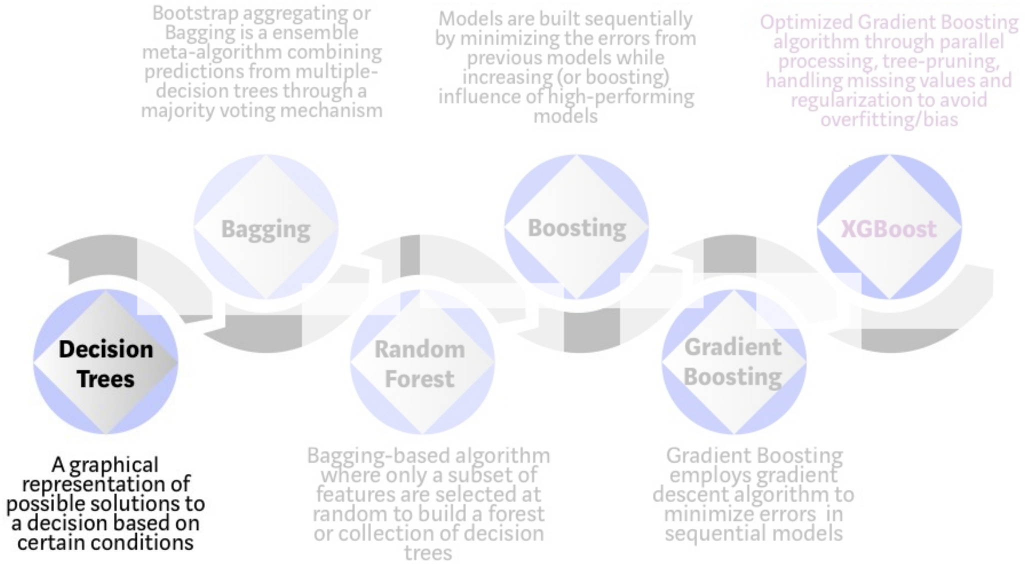
$$F_1 = \frac{1}{n_y} \sum_{k=1}^{n_y} F_1(k)$$

  - weights all classes equally
- Both of these approaches up-weight frequent classes:
  - **Micro-averaging**: Compute model-level sums for true positives, false positives, and false negatives; compute precision/recall from model sums.
  - "**Weighted**": computed like macro-averaging, but classes are weighted by true frequency.

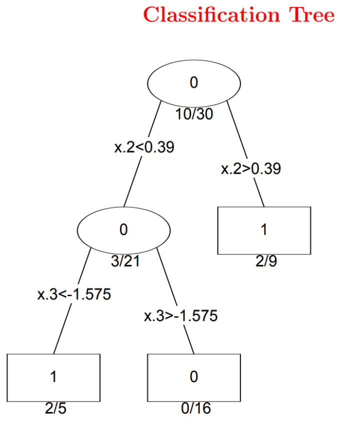
XGBoost: Overview



XGBoost Ingredients: Decision Trees

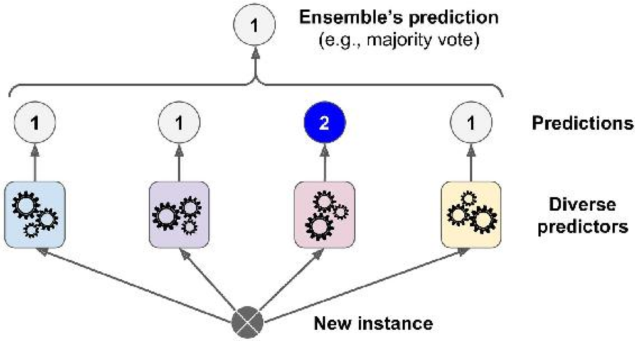

Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

Decision Trees



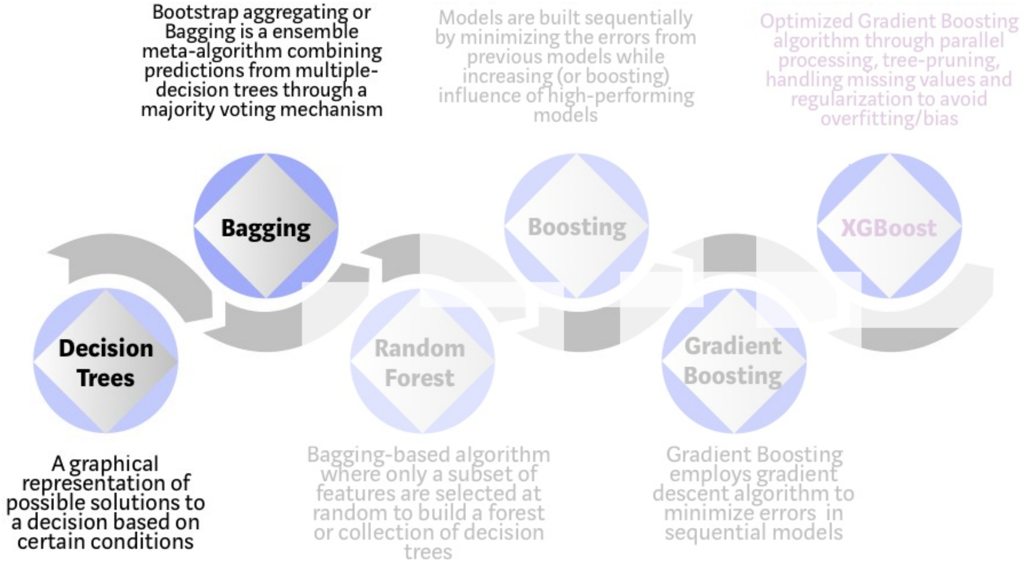
- Decision trees learn a series of binary splits in the data based on hard thresholds.
  - if yes, go right; if no, go left.
- Can have additional splits as you move through the tree.
- fast and interpretable, but performance is often poor.

Voting Classifiers



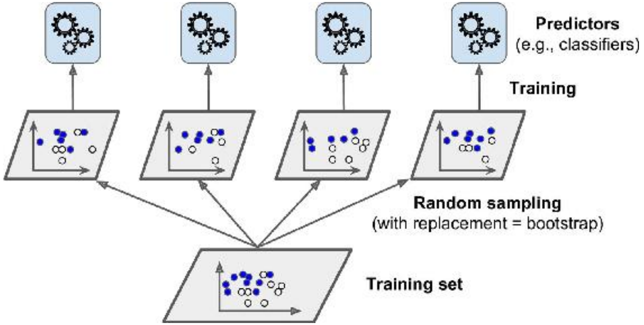
- voting classifiers (ensembles of different models that vote on the prediction) generally out-perform the best classifier in the ensemble.
  - more diverse algorithms will make different types of errors, and improve your ensemble's robustness.

XGBoost Ingredients: Bootstrapping


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

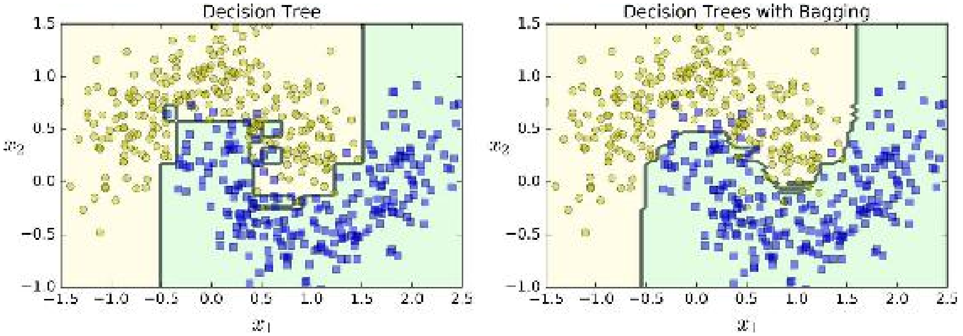
Bootstrapping

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



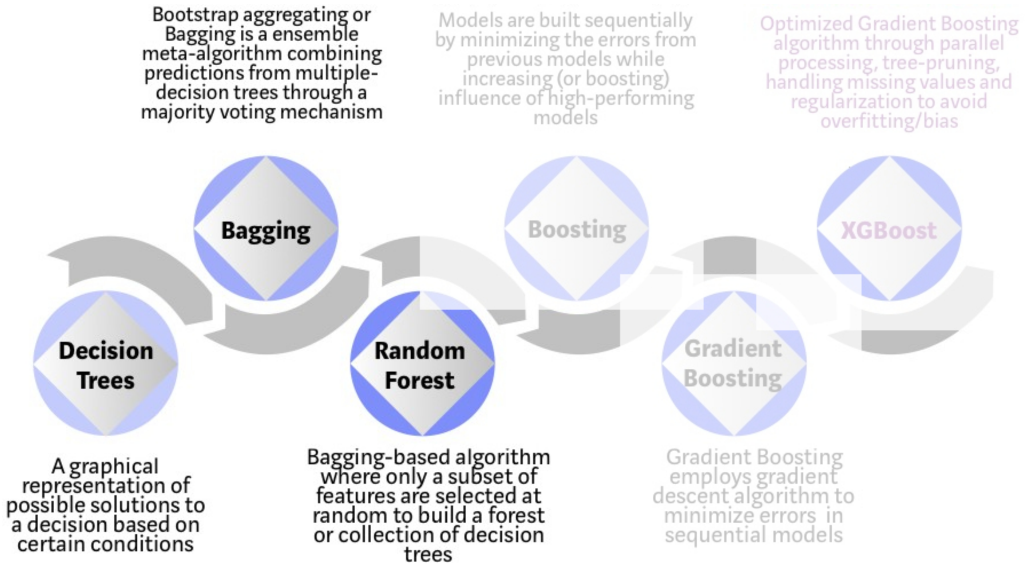
  - can also use different subsets of features across subclassifiers.

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


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

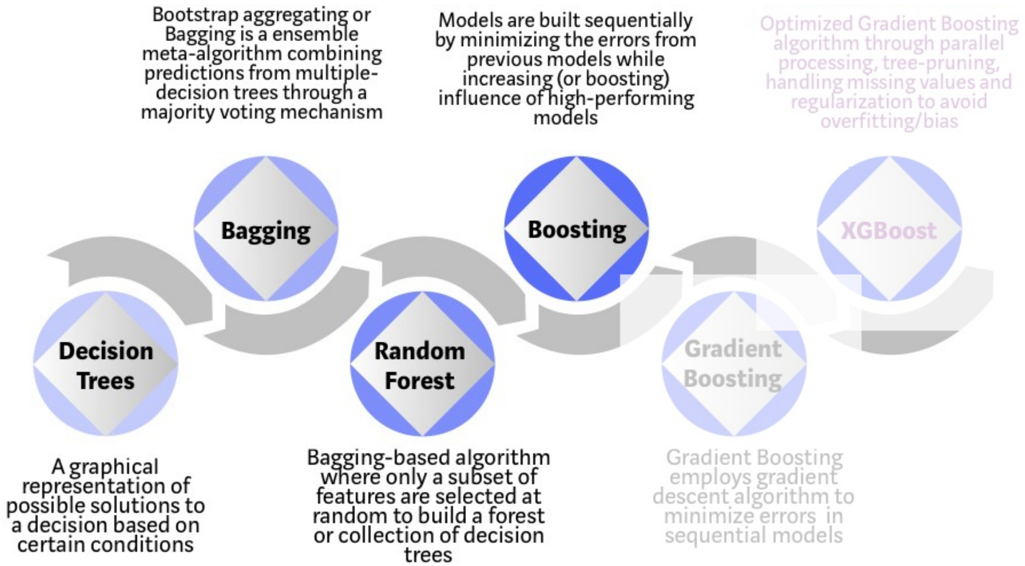
Random Forests

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.

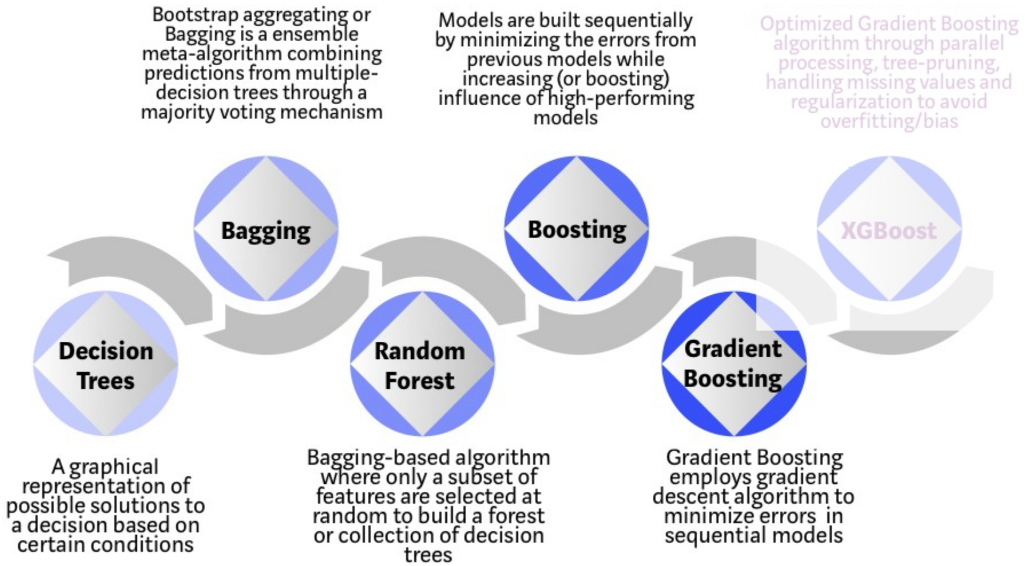
XGBoost Ingredients: Boosting


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

Boosting

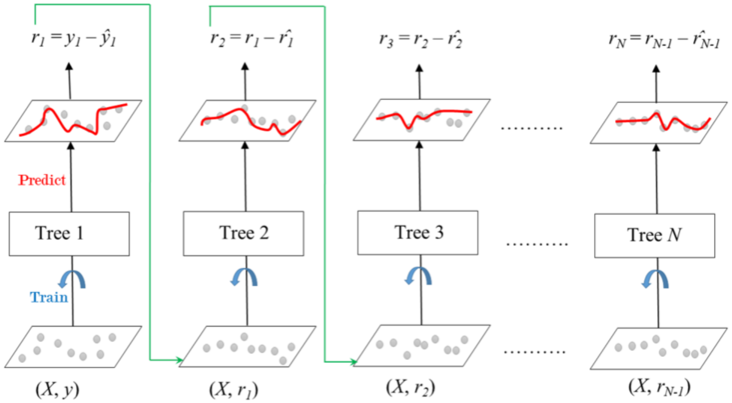
- works by sequentially adding predictors to an ensemble – fits the new predictor to the residual errors made by the previous predictor to gradually improve the model.

XGBoost Ingredients: Gradient Boosting


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

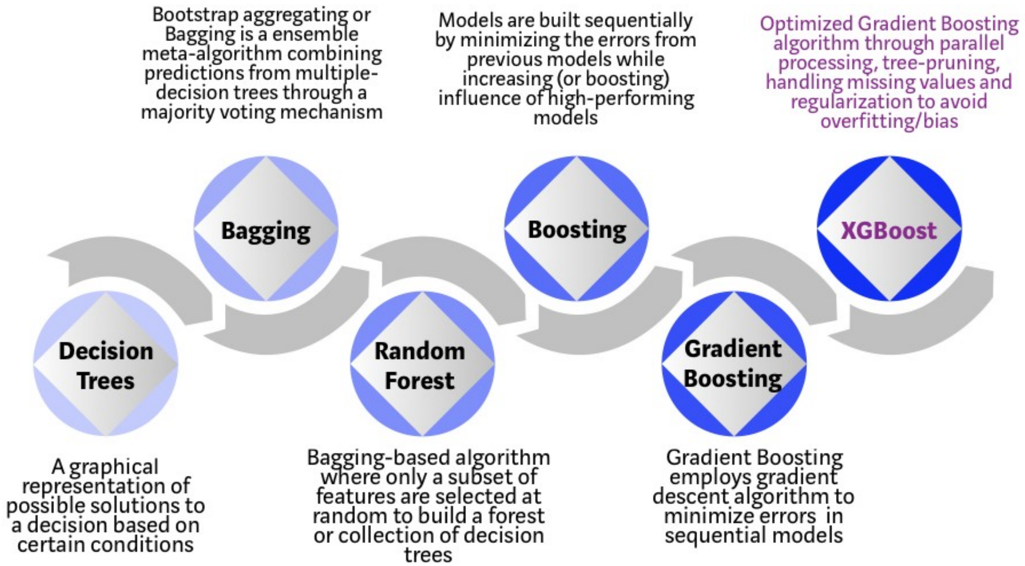
Gradient Boosting Machines

- Gradient boosting refers to an additive ensemble of trees:



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

XGBoost Ingredients

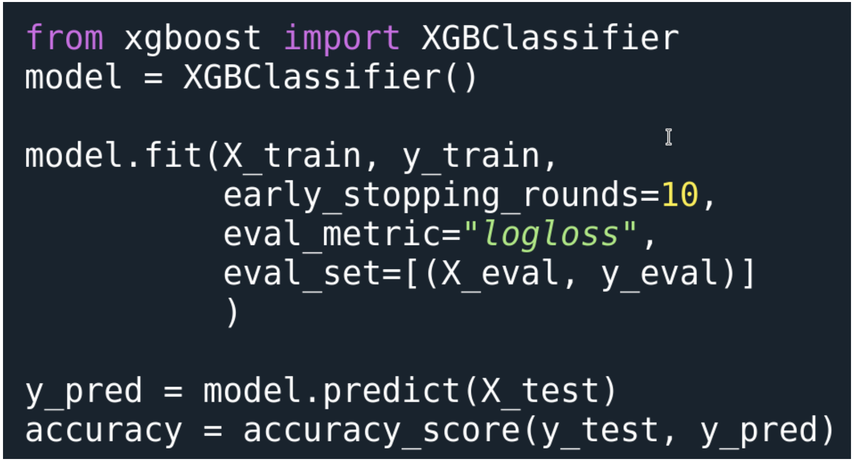


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

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

Complicated in Theory, Easy in practice



```
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)
```

- See more detailed examples/explanation in the “XGBoost Explained” reading, and this week’s example code notebook.

Early Stopping with Validation Set

- Early stopping is an elegant regularization technique, used in xgboost and in neural nets:
  - gradually train the model gradients while checking fit in a held-out validation set.
  - when model starts overfitting, stop training.
- Requires user to specify two additional parameters:
  - **`eval_set`** = validation set (separate from training set and test set)
  - **`early_stopping_rounds`**: stop training if we have done this many epochs without improving validation set performance.

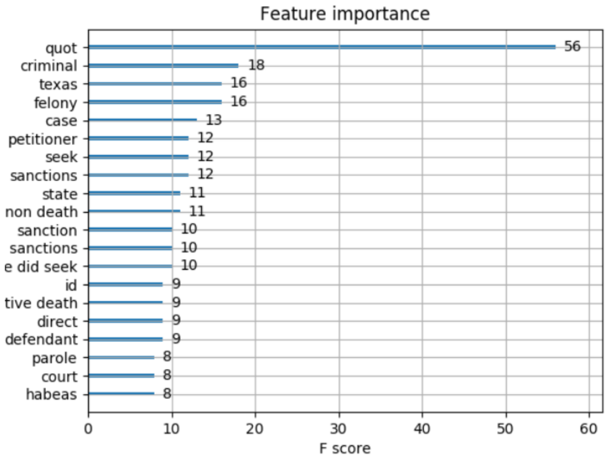
```
from sklearn.model_selection import train_test_split
X_val_train, X_test, y_val_train, y_test = train_test_split(X, y)
X_train, X_val, y_train, y_val = train_test_split(X_val_train, y_val_train)
```

```
from xgboost import XGBClassifier
xgb = XGBClassifier(eval_set = [X_val, y_val], early_stopping_rounds=10)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
balanced_accuracy_score(y_test, y_pred)
```

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.

Breakout Rooms: XGBoost Practice

- See link to python file template in zoom chat.
    - recommended: collaborate using atom teletype portal
    - alternative 1: work together on a code share (codeshare.io)
    - alternative 2: one person codes and share screen
- Will post solved PY file after lecture.

First Response Essay due next Tuesday

- Critically read and review an application paper.
- 300 words is the minimum for a passing grade but 500+ words would be expected for a 10/10.
- Please anonymize your submission – put your name in the filename, but not anywhere in the file. Submit as TXT or PDF.
  - Label your file **HW04_LastName_FirstName.txt (or .pdf)**

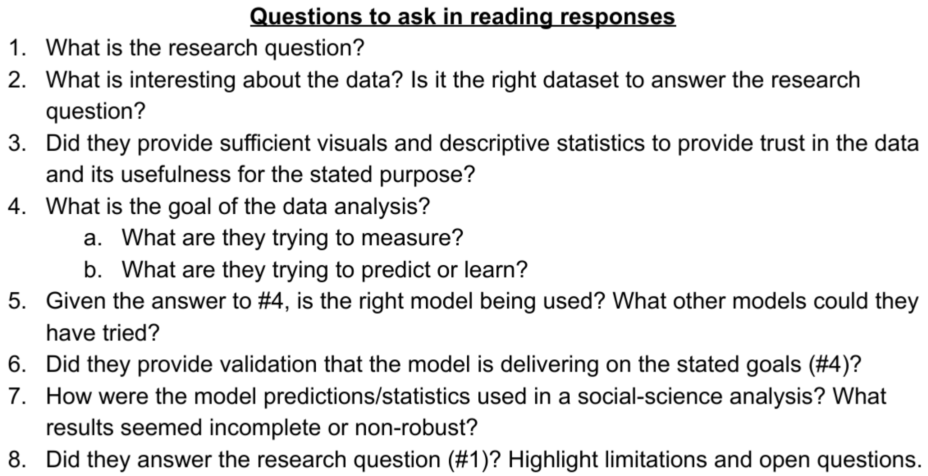
What can I write about?

Any "Applications" reading from weeks 1 through 4:

1. Bonica, Inferring Roll-Call Scores from Campaign Contributions using Supervised Machine Learning
2. Dunn, Sagun, Sirin, and Chen, "Early predictability of asylum court decisions."
3. Ash and MacLeod, Mandatory retirement reforms for judges improved performance on U.S. state supreme courts
4. Ash, Chen, and Naidu, Ideas Have Consequences: The Impact of Law and Economics on American Justice
5. Bansak, Ferwerda, Hainmueller, Dillon, Hangartner, Lawrence, and Weinstein, Improving refugee integration through data-driven algorithmic assignment.
6. Osnabruegge, Ash, and Morelli, "Cross-Domain Topic Classification for Political Texts"
7. Katz, Bommarito, and Blackman, "A general approach for predicting the behavior of the Supreme Court of the United States."

Or another article applying tools from one of the first three topics (machine learning, causal inference with linear regression, fixed effects models, or classification). Please confirm with me by email, preferably by tomorrow.

What to think about



Questions to ask in reading responses

1. What is the research question?
2. What is interesting about the data? Is it the right dataset to answer the research question?
3. Did they provide sufficient visuals and descriptive statistics to provide trust in the data and its usefulness for the stated purpose?
4. What is the goal of the data analysis?
   a. What are they trying to measure?
   b. What are they trying to predict or learn?
5. Given the answer to #4, is the right model being used? What other models could they have tried?
6. Did they provide validation that the model is delivering on the stated goals (#4)?
7. How were the model predictions/statistics used in a social-science analysis? What results seemed incomplete or non-robust?
8. Did they answer the research question (#1)? Highlight limitations and open questions.

- Annotated examples from previous years are available here: `http://bit.ly/BRJ_essay-examples`.