Ensemble Learning

Ensemble Learning

Géron, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

Tin Kam Ho, “Random Decision Forests,” *Proceedings of the Third International Conference on Document Analysis and Recognition* 1 (1995): 278.

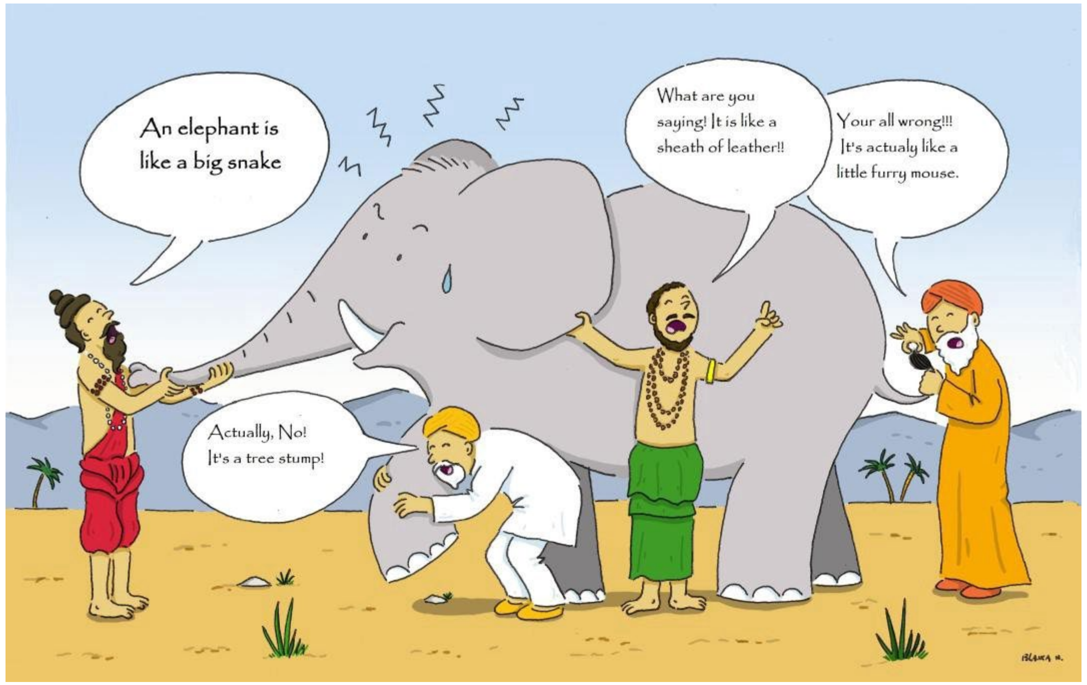
Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794).

Hastie, T., Rosset, S., Zhu, J., & Zou, H. (2009). Multi-class adaboost. *Statistics and its Interface*, *2*(3), 349-360.

Ensemble Learning

- So far, we utilized one machine learning algorithm for your predictions

- Ensemble learning combines several machine learning algorithms to increase the performance
  - Imaging the **fable of blind men and elephants.**
    All of the blind men had their descriptions of the elephant. Even though each of the descriptions was true, it would have been better to come together and discuss their understanding before concluding. This story perfectly describes the ***ensemble learning method.***


An elephant is like a big snake
Actually, No! It's a tree stump!
What are you saying! It is like a sheath of leather!!
Your all wrong!!! It's actualy like a little furry mouse.

Ensemble Learning

- Definitions of key terms:
  - A group of predictors (also referred to as classifiers or machine learning models) is called an *ensemble*
  - Using a group of predictors in machine learning is called e*nsemble learning*
  - An ensemble learning algorithm is called an *ensemble method*
- Example ensemble method:
  - Train a group of decision tree classifiers, each on a different random subset of the training set
  - To make predictions, obtain the predictions of all the individual trees, then predict the class that gets the most votes
  - This ensemble method refers to *random forest*
  - Despite its simplicity, this is one of the most powerful machine learning algorithms available today

Ensemble Learning

- When to use an ensemble method?
  - Ensemble methods are often used near the end of a machine learning project.
  - When you have already built a few good predictors, you can combine them into an even better predictor.

- In this chapter we will discuss the most popular ensemble methods with the most common implementation
  - *Voting*
    - *Implementations: voting classifier, voting regression*
  - *Bagging and Pasting*
    - *Implementations: random forest regression, random forest classification*
  - *Boosting*
    - Implementations: AdaBoost, gradient boosting, extreme gradient boosting

Ensemble Learning

Voting Classifiers

Bagging and Pasting

Bagging: Random Forrest

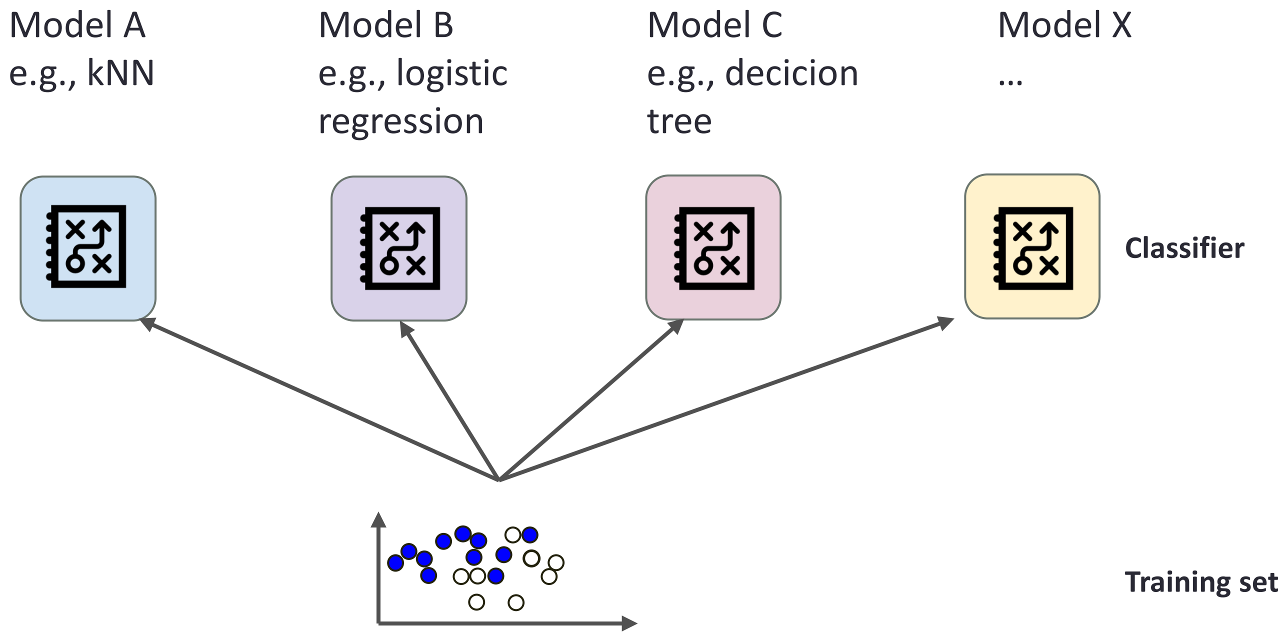
Boosting: AdaBoost

Boosting: Gradient Boosting

Boosting: Extreme Gradient Boosting

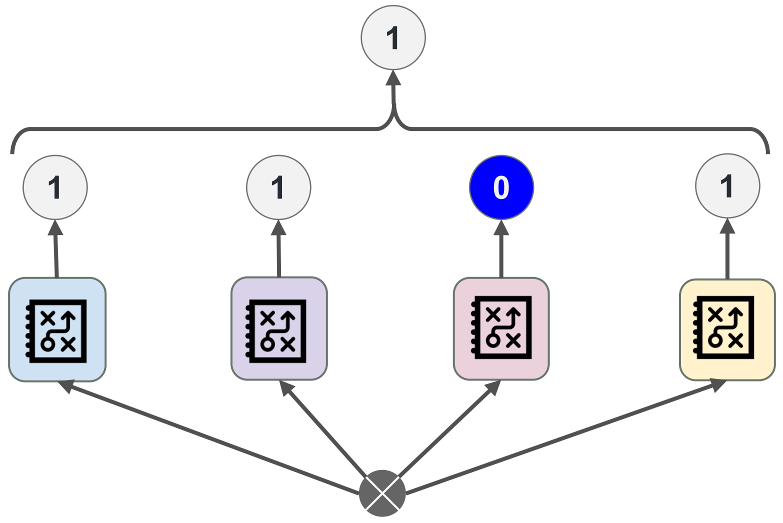
Voting Classifiers

- Voting classifiers
  - The models A, B, C, … could be anyone of the following machining learning models as we are familiar with:



Voting Classifiers

- Voting classifiers
  - A very simple way to create an even better classifier is to aggregate the predictions of each classifier and predict the class that gets the most votes.
  - This majority-vote classifier is called a *hard voting* classifier
  - Taking the highest class probability, averaged over all the individual classifiers refers to *soft voting*



Ensemble's prediction
(e.g., hard or soft voting)

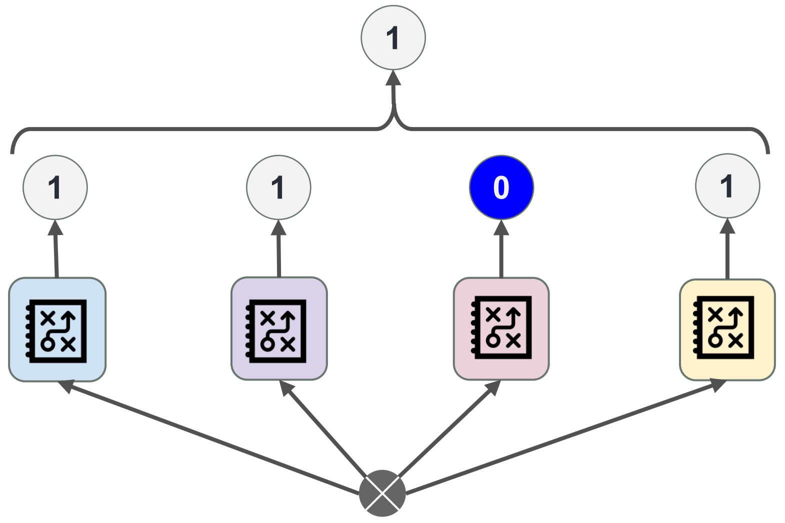
Predictions from the trained ML models (e.g., 1 or 0 for a binary classification problem)

Trained machining learning models (classifier)

New instance

Voting Classifiers

- Voting classifiers
  - A voting classifier works best when the predictors are as independent (from one another) as possible.
  - One way to get diverse classifiers is to train them using very different algorithms.
  - This increases the chance that they will make very different types of errors, improving the ensemble's performance.



Ensemble's prediction
(e.g., hard or soft voting)

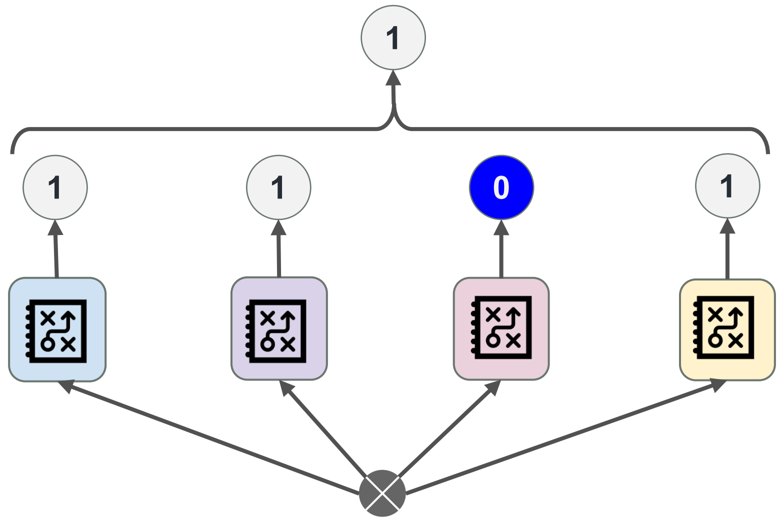
Predictions from the trained ML models (e.g., 1 or 0 for a binary classification problem)

Trained machining learning models

New instance

Voting Classifiers

- Ensemble methods work best when the predictors are as independent from one another as possible.
- One way to get diverse classifiers is to train them using very different algorithms.
- This increases the chance that they will make very different types of errors, improving the ensemble's accuracy.



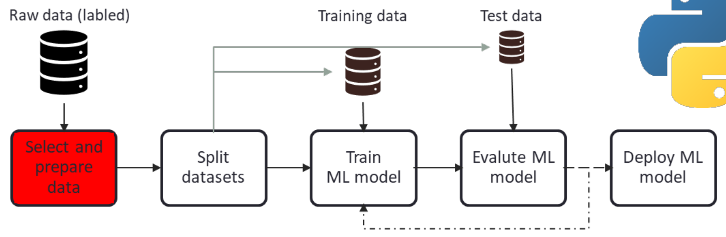
Ensemble's prediction
(e.g., hard or soft voting)

Predictions from the trained ML models (e.g., 1 or 0 for a binary classification problem)

Trained machining learning models

New instance

Voting Classifiers: Python Example

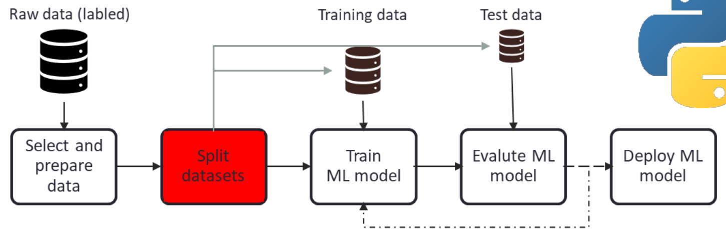


Voting Classifiers

Let's use the moons dataset

```python
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42) ## generate random data
```

Voting Classifiers: Python Example



Voting Classifiers

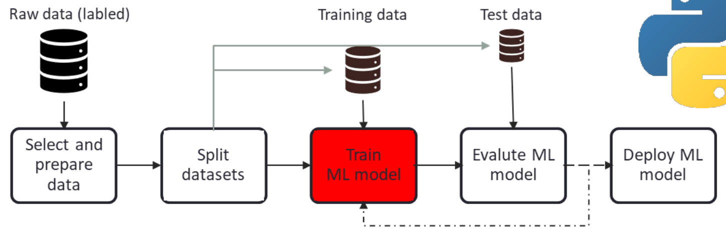
```
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

Voting Classifiers: Python Example

```
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

log_clf = LogisticRegression(solver="lbfgs", random_state=42)
dtr_clf = DecisionTreeClassifier(random_state=42)
knn_clf = KNeighborsClassifier(n_neigbors=3)

voting_clf = VotingClassifier(
estimators = [('lr', log_clf), ('dt', dtr_clf), ('knn', knn_clf)],
voting='hard')
```

```
voting_clf.fit(X_train,y_train)
```

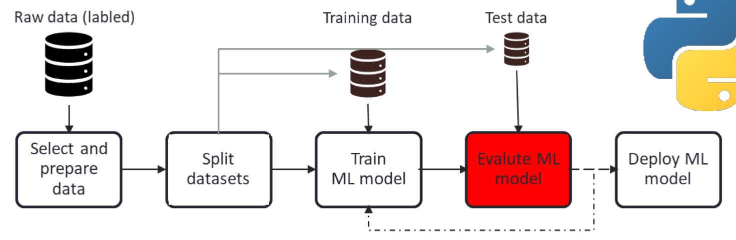
```
VotingClassifier(estimators=[('lr', LogisticRegression(random_state=42)),
                             ('dt', DecisionTreeClassifier(random_state=42)),
                             ('knn', KNeighborsClassifier(n_neighbors=3))])
```



Voting Classifiers: Python Example

```python
from sklearn.metrics import accuracy_score

for clf in (log_clf, dtr_clf, knn_clf, voting_clf):
    clf.fit(X_train,y_train)
    y_pred = clf.predict(X_test)
    print (clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.864
DecisionTreeClassifier 0.856
KNeighborsClassifier 0.896
VotingClassifier 0.904
```



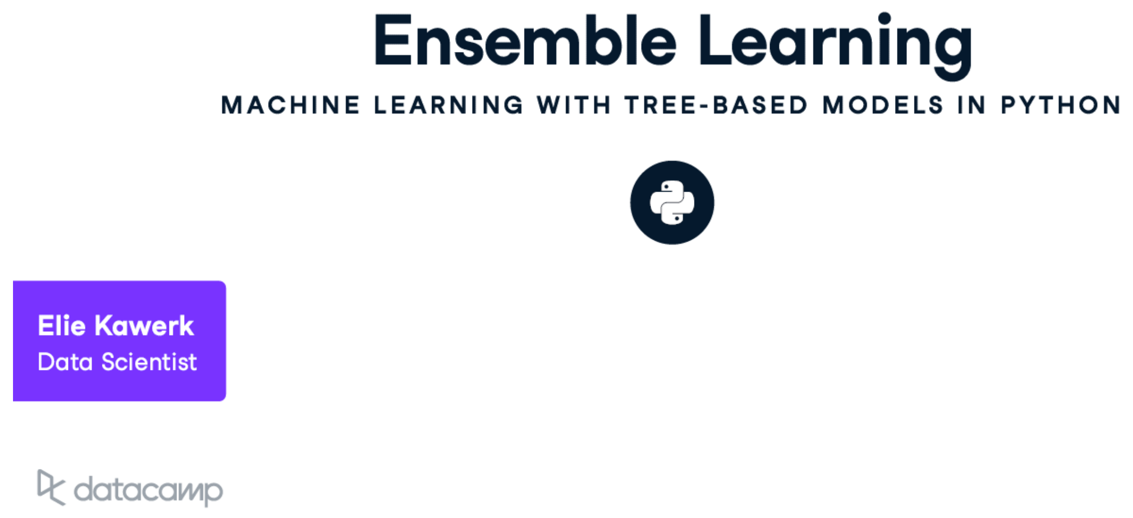
The voting classifier (0.904) slightly outperforms all the individual classifiers (range from 0.864-0.896).

Voting Classifiers

- Soft voting:
  - If all classifiers can estimate class probabilities (i.e., they all have a predict_proba() method), then you can tell Scikit-Learn to predict the class with the highest class probability, averaged over all the individual classifiers.
  - It often achieves higher performance than hard voting because it gives more weight to highly confident votes.
  - All you need to do is replace voting="hard" with voting="soft" and ensure that all classifiers can estimate class probabilities.

You can learn all this in an interactive online course!

https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/the-bias-variance-tradeoff?ex=9



Ensemble Learning

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

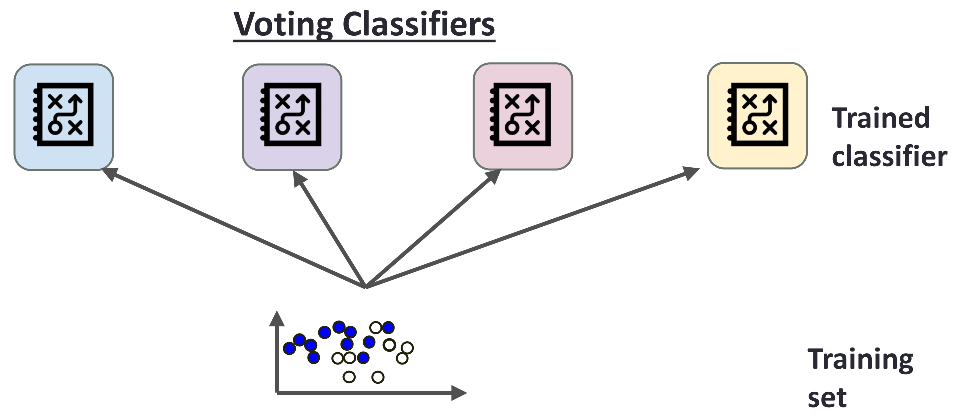
Elie Kawerk
Data Scientist

datacamp

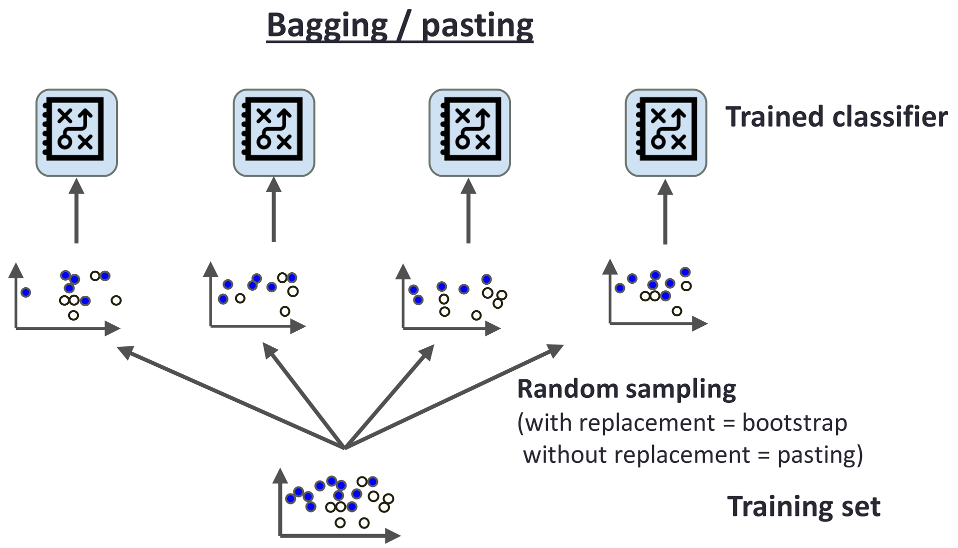
Ensemble Learning and Random Forests

Bagging and Pasting

- In the previous slides, we studied **voting classifiers**
  - Type of classifier used: different classifier
  - Data used: all trained on the entire training data.

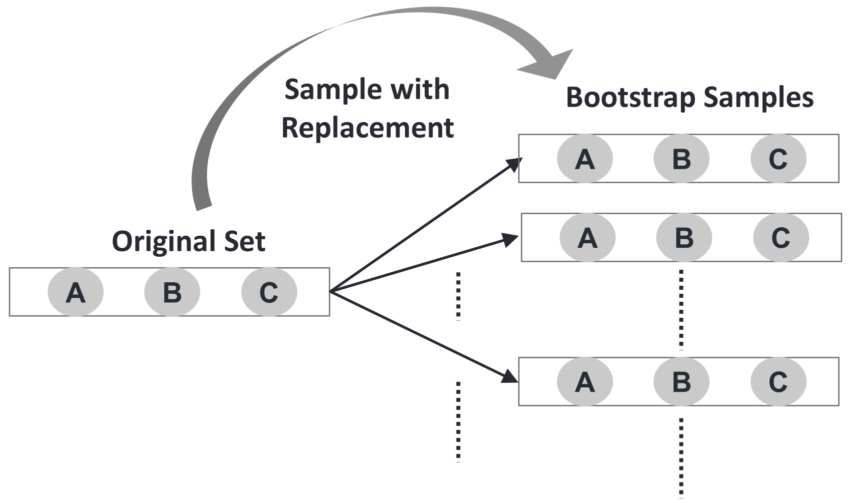


- Another approach is to use the same training algorithm for every classifier and train them on different random subsets of the training set.
  - Type of classifier: same classifier
  - Data used: trained on different subsamples of the training data
  - Bagging vs. pasting:
    - When sampling is performed *with* replacement, this method is called *bagging* (short for *bootstrap aggregating*).
    - When sampling is performed *without* replacement, it is called *pasting*



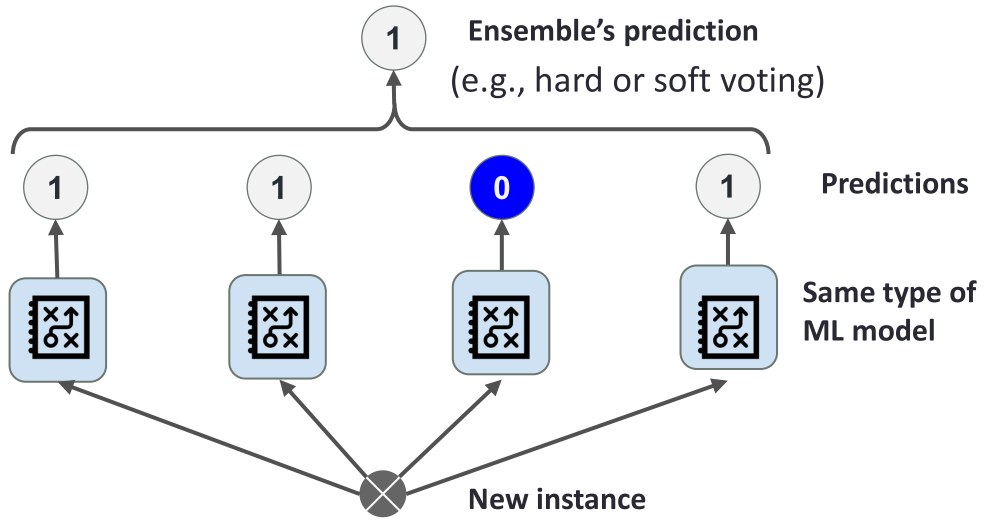
Bagging and Bootstraping

- Bagging relies on bootstrapping
  - A bootstrap sample is a sample drawn from this with replacement.
  - By replacement, any data of a sample can be drawn many times
  - For example:
    - In the first bootstrap sample shown in the diagram here, B was drawn three times in a row.
    - In the second bootstrap sample, A was drawn two times while B was drawn once.
      … and so on.



Bagging and Pasting

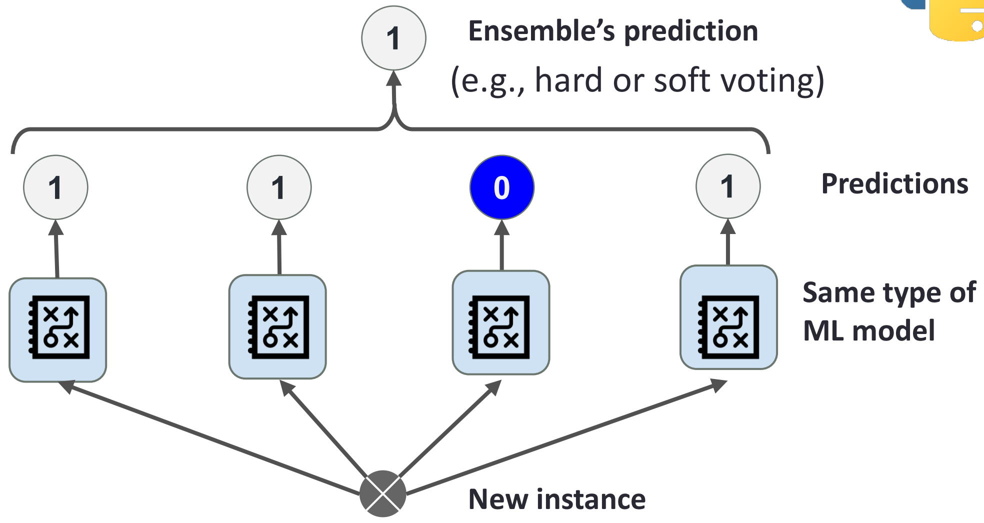
- Prediction with bagging/pasting
  - Once all predictors are trained, the ensemble can predict for a new instance by simply aggregating the predictions of all predictors.
  - The aggregation function is typically a *hard vote* (i.e., the most frequent prediction) for classification or the average for regression.
  - If the classifier provides class probabilities, also *soft vote* can be used.
  - Each predictor has a higher bias than trained on the original training set, but aggregation reduces both bias and variance.
  - Generally, the net result is that the ensemble has a similar bias but a lower variance than a single predictor trained on the original training set.



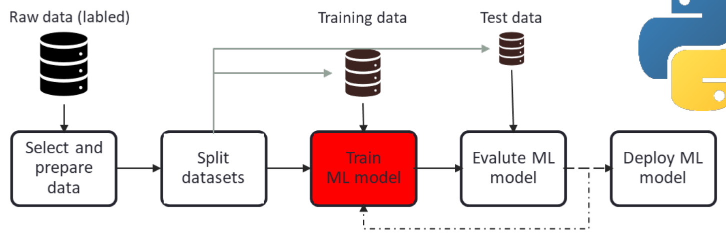
- Parallelization of bagging/pasting enables huge scalability
  - Predictors can all be trained in parallel
  - Predictions can be made in parallel:
    - via different CPU cores or
    - via different servers

Bagging and Pasting

- Bagging/pasting in Scikit-Learn
  - Sckit-learn offers a simple API for both bagging and pasting with the BaggingClassifier class
    - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html
    - Aggregates predictions by majority voting.
    - The BaggingClassifier automatically performs soft voting instead of hard voting if the base classifier can estimate class probabilities (i.e., if it has a predict_proba() method)
  - Can also be sued for regression with or BaggingRegressor class
    - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html
    - Aggregates predictions through averaging



Bagging: Python Example



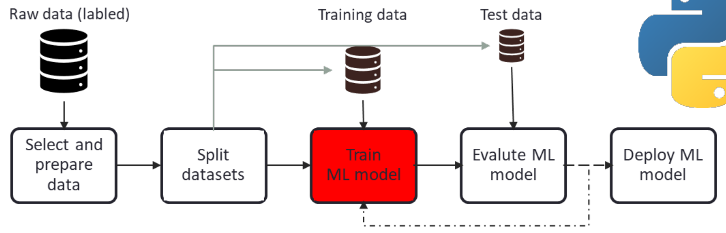
```python
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, random_state=42)
bag_clf.fit(X_train,y_train)
y_pred = bag_clf.predict(X_test)
```

- Explanation:
  - The code trains an ensemble of 500 decision tree classifiers
  - Each decision tree is trained on 100 training instances
  - The training instances are randomly sampled from the training set with replacement
  - This is an example of bagging, because we set bootstrap = True
  - To use pasting, we need to set bootstrap = False

Bagging: Python Example

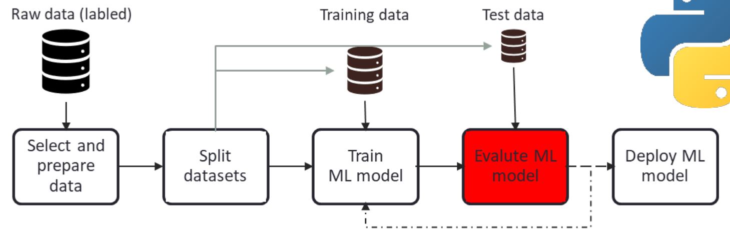
```
bag_clf.get_params()
```

```
{'base_estimator__ccp_alpha': 0.0,
 'base_estimator__class_weight': None,
 'base_estimator__criterion': 'gini',
 'base_estimator__max_depth': None,
 'base_estimator__max_features': None,
 'base_estimator__max_leaf_nodes': None,
 'base_estimator__min_impurity_decrease': 0.0,
 'base_estimator__min_impurity_split': None,
 'base_estimator__min_samples_leaf': 1,
 'base_estimator__min_samples_split': 2,
 'base_estimator__min_weight_fraction_leaf': 0.0,
 'base_estimator__random_state': None,
 'base_estimator__splitter': 'best',
 'base_estimator': DecisionTreeClassifier(),
 'bootstrap': True,
 'bootstrap_features': False,
 'max_features': 1.0,
 'max_samples': 100,
 'n_estimators': 500,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```



- Explanation:
  - There are various other hyperparameters that can be tuned

Bagging: Python Example



```python
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.904
```

The bagging ensembler reaches an accuracy of 0.904. When using the same data with a “simple” decision tree we only reach an accuracy of 0.856.

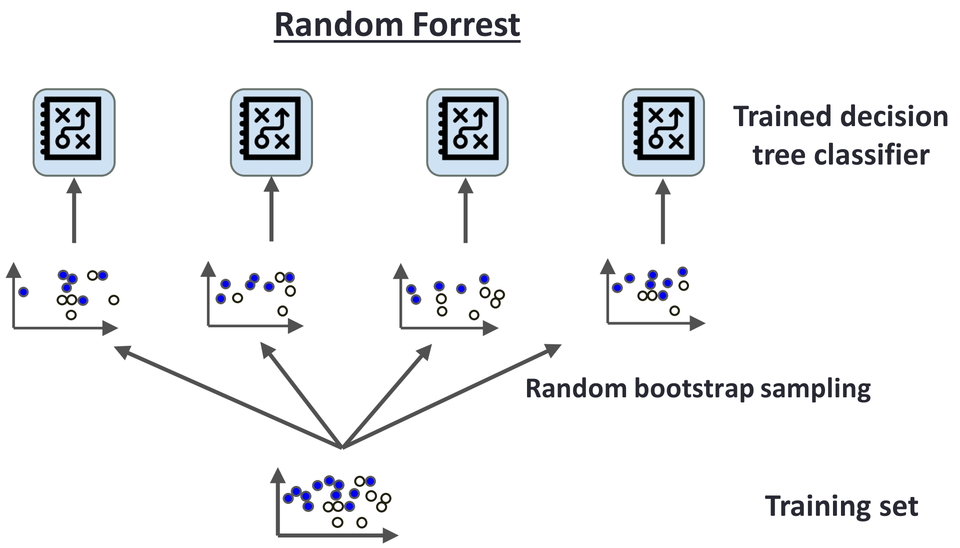
→ **bagging can often easily increase the performance**

Ensemble Learning and Random Forests

Random Forests as a Typical Bagging Classifier

- Random forest is an ensemble of decision trees
  - mostly trained via the bagging method
  - typically, with max_samples set to the size of the training set

- Random forest has a distinct class
  - Because of its popularity, random forests do not need to be built from scratch every time by building a BaggingClassifier and passing it a DecisionTreeClassifier
  - Instead, we can use the RandomForestClassifier class

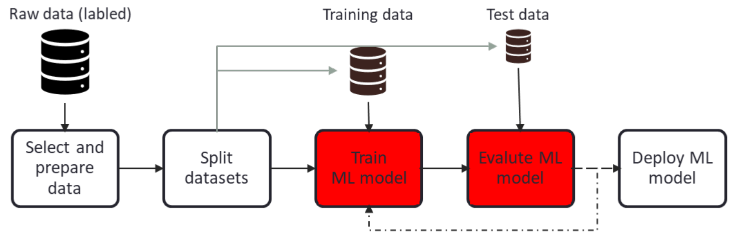


Random Forests

- Random forest and hyperparameter
  - Random forest classifier has all the hyperparameters of a DecisionTreeClassifier (to control how trees are grown)
  - Additionally, all the hyperparameters of a BaggingClassifier to control the ensemble itself can be used

- Random forest for classification and regression
  - Remember, just as decision trees, random forest can be used for classification as well as for regression
  - For classification: RandomForestClassifier
    https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
    - Aggregates predictions by majority voting
  - For regression: RandomForestRegressor
    https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
    - Aggregates predictions through averaging

Random Forests: Python Example



```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, random_state=42)
rnd_clf.fit(X_train,y_train)

y_pred = rnd_clf.predict(X_test)
print(accuracy_score(y_test, y_pred_rf))
```

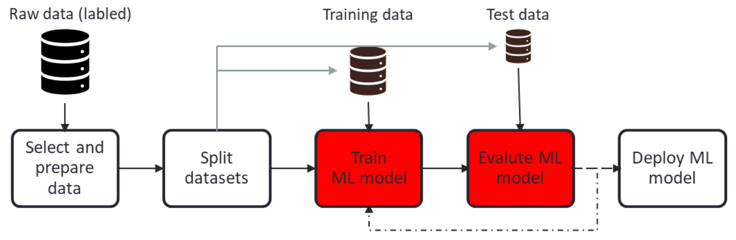
```
0.912
```

- Explanation:
  - Using the RandomForrestClassifier class instead of manually building it

Random Forests: Python Example

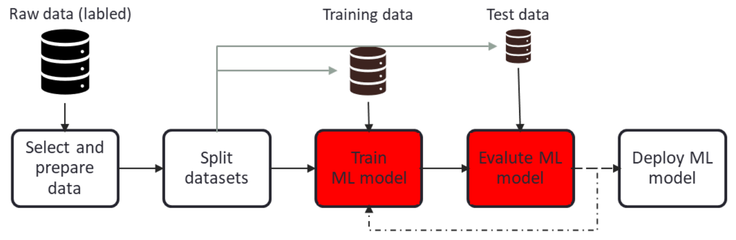
```
rnd_clf.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': 16,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 500,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```



- Explanation:
  - There are many more hyperparameters to tune

Random Forests: Python Example



```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(max_feautures = "sqrt", max_leaf_nodes=16),
    n_estimators=500, random_state=42)
```

```
bag_clf.fit(X_train,y_train)
y_pred = bag_clf.predict(X_test)

print(accuracy_score(y_test, y_pred))
```

```
0.912
```

- Explanation:
  - Building a random forest classifier manually as a bagging classifier

The results are the same, but using the RandomForestClassifier class is more convenient.

Random Forests: Explainable AI

- Feature importance
  - Yet another excellent quality of random forests is that they make it easy to measure the relative importance of each feature.
  - A measure of a feature’s importance can be measured by looking at how much the tree nodes that use that feature reduce impurity on average (across all trees in the forest).
  - More precisely, it is a weighted average, where each node’s weight is equal to the number of training samples that are associated with it

- Feature importance in Sckit-Learn
  - Scikit-Learn computes the feature importance automatically for each feature after training,
  - The feature importance is scaled so that the sum is equal to 1
  - You can access the result using the feature_importances_ variable

- The following slides demonstrate this by using the example of the diabetes dataset

Remember the Running Example

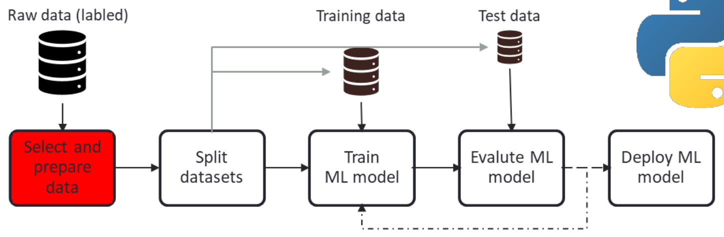
- As a healthcare data analyst, your job is to identify patients or sufferers that have a higher chance of a particular disease, for example, diabetes

- These predictions will help you to treat patients before the disease occurs

- As an analyst, you have first to define the problem that you want to solve using classification and then identify the potential features that predict the labels accurately
  - Features are the columns or attributes that are responsible for prediction.
  - In diabetes prediction problems, health analysts will collect patient information, such as the number of pregnancies the patient has had, their BMI, insulin level, age.

- Terminology:
  - Classification is the process of categorizing customers into two or more categories.
  - The classification model predicts the categorical class label, such as whether the customer is potential or not.
  - In the classification process, the model is trained on available data, makes predictions, and evaluates the model performance.
  - Developed models are called classifiers.

Random Forests: Python Example



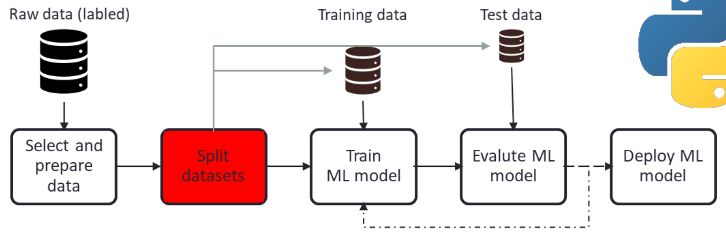
Diabetes data: Select and prepare data

```
from sklearn.datasets import load_iris
import pandas as pd
diabetes = pd.read_csv("diabetes.csv")
diabetes.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# Splitting dataset in two parts: feature set and target label
feature_set = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
features = diabetes[feature_set]
target = diabetes.label
```

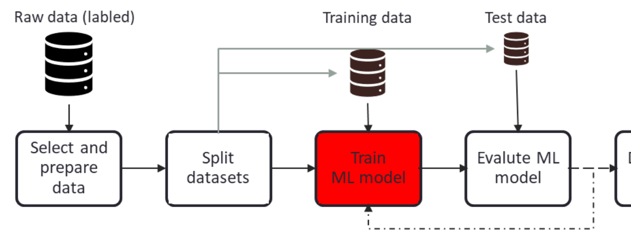
Random Forests: Python Example



Diabites Data: Split dataset

```
# Partition data into training and testing set
from sklearn.model_selection import train_test_split
feature_train,feature_test, target_train, target_test = train_test_split(features, target, test_size=0.3,
random_state=42)
```

Random Forests: Python Example

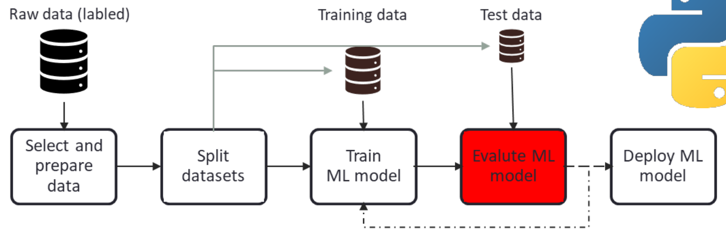


Diabetes Data: train random forest model

```
rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, random_state=42)
rnd_clf.fit(feature_train,target_train)
```

```
RandomForestClassifier(max_leaf_nodes=16, n_estimators=500, random_state=42)
```

Random Forests: Python Example



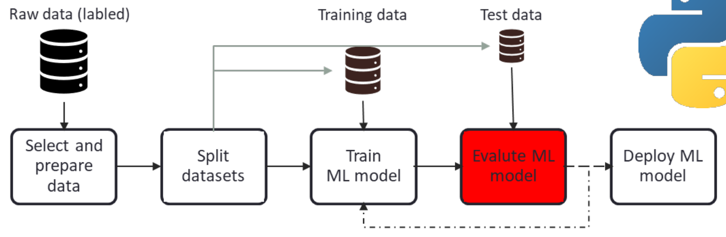
Diabetes Data: evaluate random forest model

```
target_pred = rnd_clf.predict(feature_test)
print(accuracy_score(target_test, target_pred))
```

```
0.7619047619047619
```

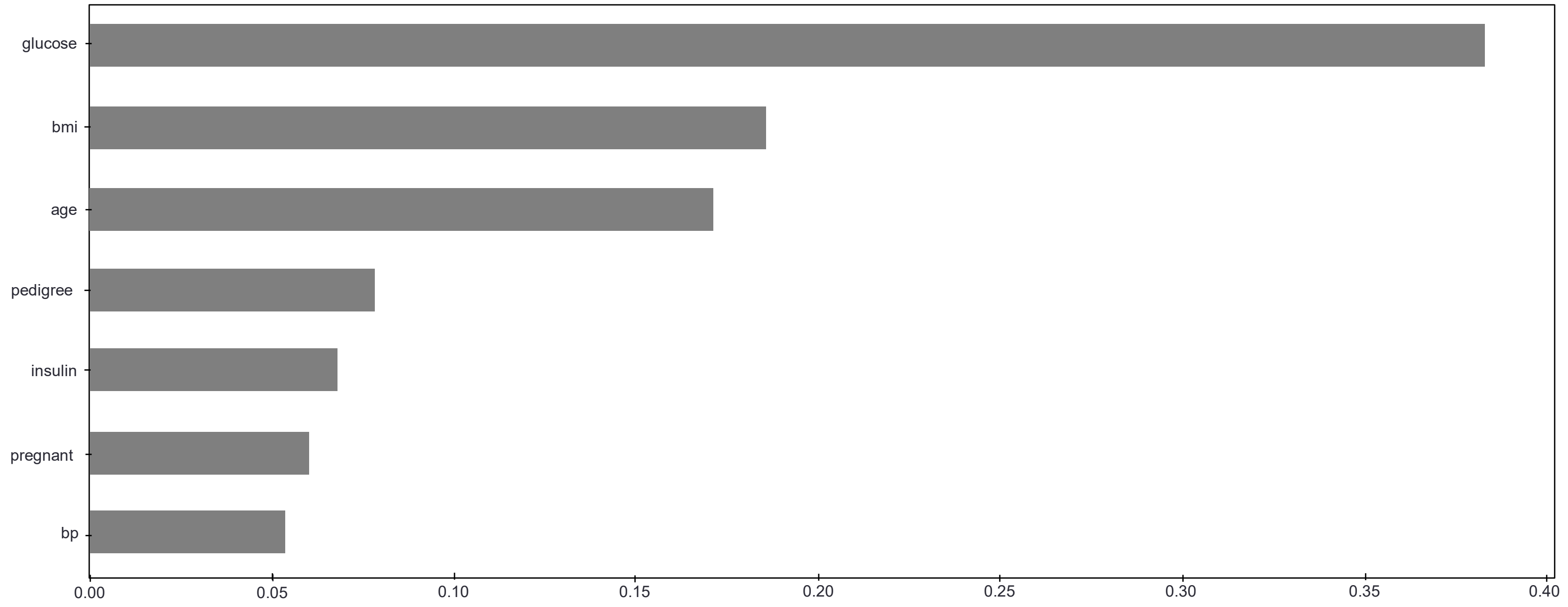
The model has an accuracy of 0.76

Random Forests: Python Example



```python
# Print feature importance
for name, score in zip(feature_set, rnd_clf.feature_importances_):
    print(name, score)
```

```
pregnant 0.0596791694937236
insulin 0.0679218158097347
bmi 0.18599011296302081
age 0.17153427598815596
glucose 0.3828602471876601
bp 0.05382292519949445
pedigree 0.0781914533582104
```



- Explanation:
  - It seems that the most important features are the glucose level (38%), followed by the BMI (18%) and age (17%)
  - The other features are relatively unimportant

Random Forests: Explainable AI

- Why is feature importance so relevant?

- Explainablity
  - The results help to explain the results of the random forest
  - We get higher interpretability of the results, e.g., which features are most important and which are not important
  - Discover new insights about our data

- Feature selection
  - The results can be used to select the top N most important features (“feature selection mechanism”)
  - The reduced set of features can be used to train a new model
  - This results in various benefits:
    - faster training, because of fewer features
    - noise reduction

You can learn all this in an interactive online course!

https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/bagging-and-random-forests?ex=1

Bagging

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

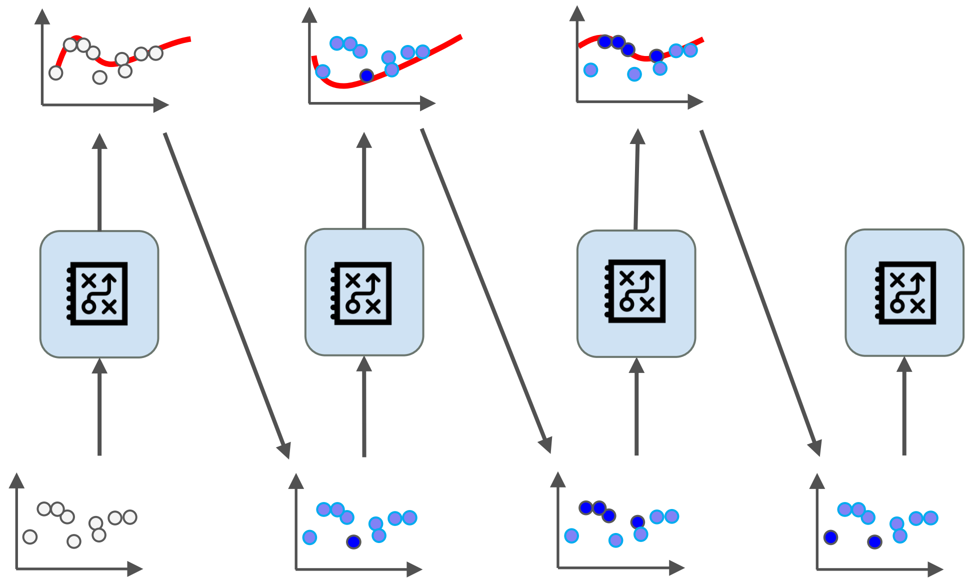
datacamp

Ensemble Learning and Random Forests

AdaBoost

- General concept:
  - One way for a new predictor to correct its predecessor is to pay more attention to the training instances that the predecessor underfitted.
  - This results in new predictors focusing more and more on the hard cases. This is the technique used by AdaBoost.
  - Once all predictors are trained, the ensemble makes predictions very much like bagging or pasting, except that predictors have different weights depending on their overall accuracy on the weighted training set.

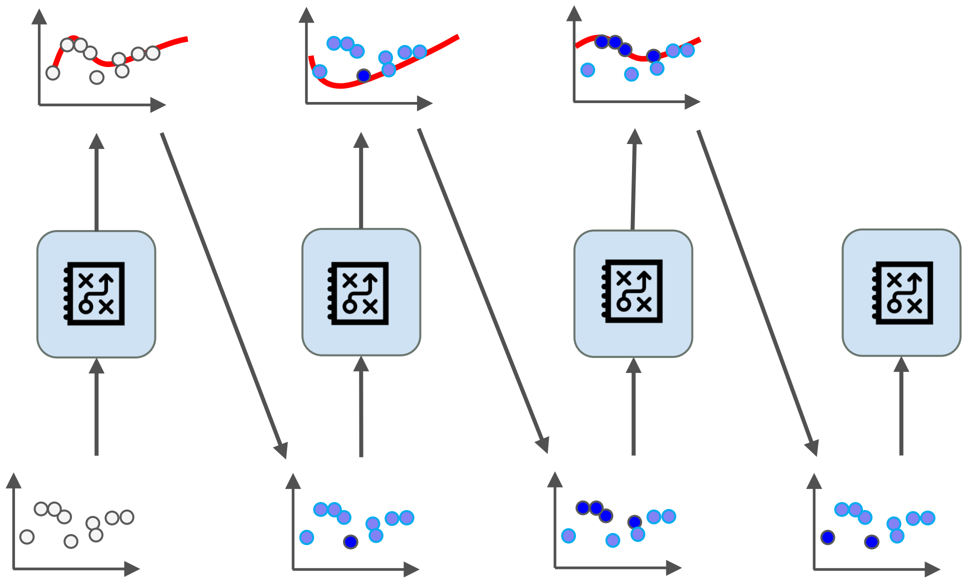
- Example:
  - When training an AdaBoost classifier, the algorithm first trains a base classifier (such as a decision tree) and uses it to make predictions on the training set.
  - The algorithm then increases the relative weight of misclassified training instances.
  - Then it trains a second classifier, using the updated weights, and again makes predictions on the training set, updates the instance weights, and so on



Note:
● well fitted training instance
● Underfitted / misclassified training instances
○ Initial training instances

AdaBoost

- Drawback of sequential learning, such as AdaBoost
  - A sequential learning technique cannot be parallelized (or only partially),
  - Each predictor can only be trained after the previous predictor has been trained and evaluated.
  - As a result, it does not scale as well as bagging or pasting.



Note:

- ● well fitted training instance
- ● Underfitted / misclassified training instances
- ○ Initial training instances

AdaBoost: Algorithm

- The AdaBoost algorithm
  - Each instance weight $w^{(i)}$ is initially set to $1/m$
  - A first predictor is trained, and its weighted error rate r_1 is computed on the training set with the following equation

$$r_j = \frac{\sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$$

where $\hat{y}_j^{(i)}$ is the j^{th} predictor's prediction for the i^{th} instance.

  - The predictor's weight α_j is then computed with the following equation

$$a_j = \eta \log \frac{1 - r_j}{r_j}$$

    – is the learning rate hyperparameter (defaults to 1)
    – The more accurate the predictor is, the higher its weight will be.
    – If it is just guessing randomly, its weight will be close to zero.
    – However, if it is most often wrong (i.e., less accurate than random guessing), its weight will be negative.

AdaBoost: Algoithm

- The AdaBoost algorithm
  - Next, the AdaBoost algorithm updates the instance weights with the following equation

$$\text{for } i = 1,2,\dots,m$$
$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & if\ \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp(a_j) & if\ \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

  - Then all the instance weights are normalized $\sum_{i=1}^{m} w^{(i)}$

  - Finally, a new predictor is trained using the updated weights,
    - the whole process is repeated until the desired number of predictors is reached

AdaBoost: Algorithm

- Making predictions with AdaBoost
  - To make predictions, AdaBoost computes the predictions of all the predictors and weighs them using the predictor weights α_j
  - The predicted class is the one that receives the majority of weighted votes

$$\hat{y}(\boldsymbol{x}) = \underset{k}{argmax} \sum_{\substack{j=1 \\ \hat{y}(\boldsymbol{x})=k}}^{N} a_j$$ where N is the number of predictors

AdaBoost in Scikit-Learn

- AdaBoost in Scikit-Learn
  - Scikit-Learn uses a multiclass version of AdaBoost called SAMME (*Stagewise Additive Modeling using a Multiclass Exponential loss function)*
  - This version is equivalent to AdaBoost if we only use two classes that need to be learned

- AdaBoost with class probabilities
  - If the machine learning model can estimate class probabilities (i.e., if they have a predict_proba() method), Scikit-Learn can use a variant of SAMME called SAMME.R (the R stands for “Real”), which relies on class probabilities rather than predictions

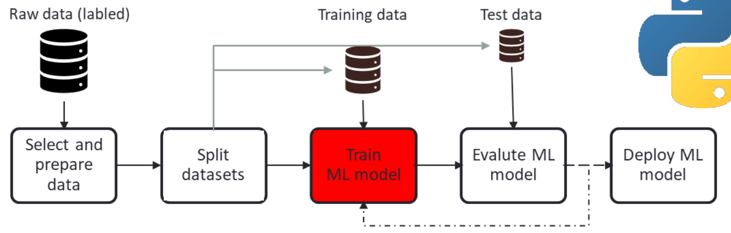
- AdaBoost for classification and regression
  - AdaBoost can be used for classification with the AdaBoostClassifier class
  - AdaBoost can also be used for regression with the AdaBoostRegressor class

AdaBoost: Python Example

INFORMATION SYSTEMS AND SERVICES
UNIVERSITY OF BAMBERG

AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SaMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train,y_train)
```

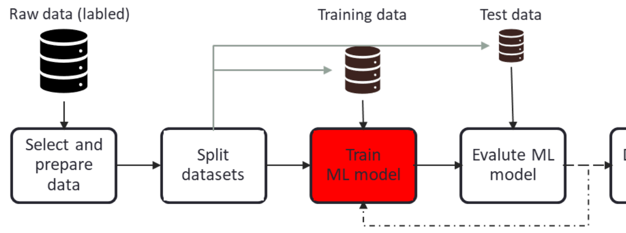


- Explanation:
  - The code trains an AdaBoost classifier based on 200 Decision Stumps using Scikit-Learn's AdaBoostClassifier class.
  - A Decision Stump is a Decision Tree with max_depth=1—in other words, a tree composed of a single decision node plus two leaf nodes.
  - This is the default base estimator for the AdaBoostClassifier class

AdaBoost: Python Example

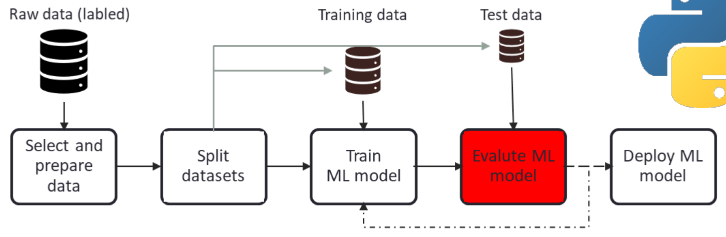
```python
ada_clf.get_params()
```

```
{'algorithm': 'SAMME.R',
 'base_estimator__ccp_alpha': 0.0,
 'base_estimator__class_weight': None,
 'base_estimator__criterion': 'gini',
 'base_estimator__max_depth': 1,
 'base_estimator__max_features': None,
 'base_estimator__max_leaf_nodes': None,
 'base_estimator__min_impurity_decrease': 0.0,
 'base_estimator__min_impurity_split': None,
 'base_estimator__min_samples_leaf': 1,
 'base_estimator__min_samples_split': 2,
 'base_estimator__min_weight_fraction_leaf': 0.0,
 'base_estimator__random_state': None,
 'base_estimator__splitter': 'best',
 'base_estimator': DecisionTreeClassifier(max_depth=1),
 'learning_rate': 0.5,
 'n_estimators': 200,
 'random_state': 42}
```



- Explanation:
  - There are many more hyperparameters that can be tuned

AdaBoost: Python Example



```
y_pred = ada_clf.predict(X_test)
print(accuracy_score(y_test, y_pred))
```

```
0.896
```

The model has an accuracy of 0.896

You can learn all this in an interactive online course!

https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/boosting?ex=1

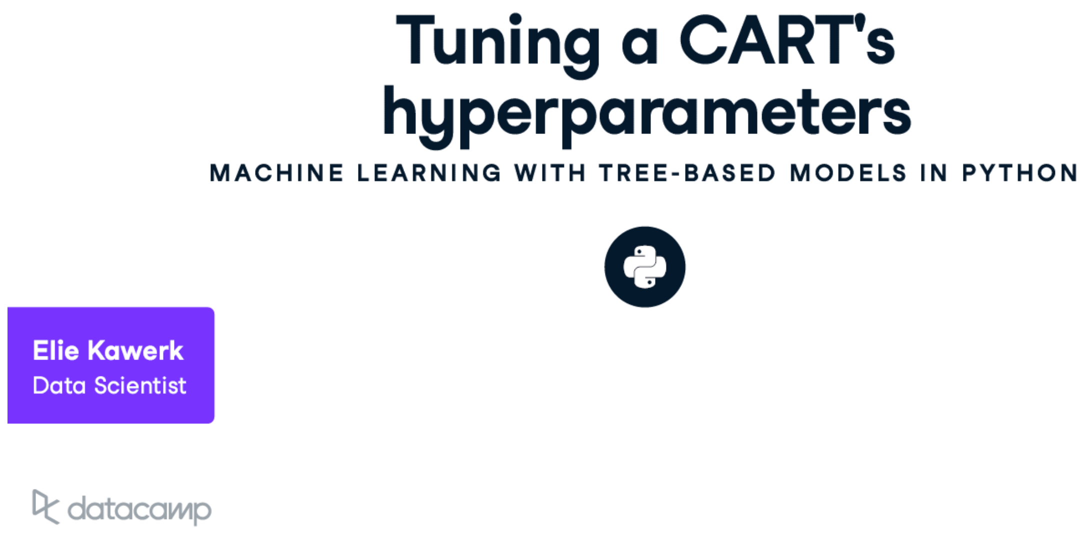
AdaBoost

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

You can learn all this in an interactive online course!

https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/model-tuning?ex=1



Tuning a CART's hyperparameters

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

datacamp

Ensemble Learning and Random Forests

Gradient Boosting

- Gradient boosting
  - Just like AdaBoost, gradient boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor.
  - Instead of tweaking the instance weights at every iteration as AdaBoost does, this method tries to fit the new predictor to the *residual errors* made by the previous predictor

- Gradient boosting for classification and regression
  - Gradient boosting can be used for regression, e.g., with regression trees
    $\rightarrow$ Gradient boosted regression trees (GBRT)
  - Gradient Boosting can also be used for classification, e.g., with decision trees
    $\rightarrow$ Gradient boosted decision tree (BGDT)

- Example
  - To demonstrate the fundamental concept of gradient boosting, we use a simple regression example and build a gradient boosting manually
  - Regression example is particularly suited because we are well-familiar with the role of residuals in regressions
  - The same principle can be transferred to classification tasks

Gradient Boosting: Manually Procedure

- First, let's fit a DecisionTreeRegressor to the training set

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)
```

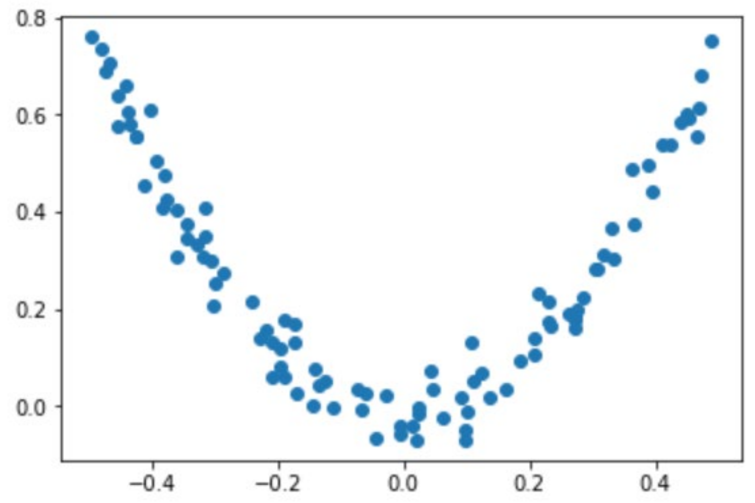
- Next, we will train a second DecisionTreeRegressor on the residual errors made by the first predictor:

```
y2 = y - tree_reg1.predict(X)

tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X, y2)
```

- Then we train a third regressor on the residual errors made by the second predictor:

```
y3 = y2 - tree_reg2.predict(X)

tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg3.fit(X, y3)
```

- Now we have an ensemble containing three trees. It can make predictions on a new instance simply by adding up the predictions of all the trees:



Example data set

Gradient Boosting: Manually Procedure

- Now we have an ensemble containing three trees.

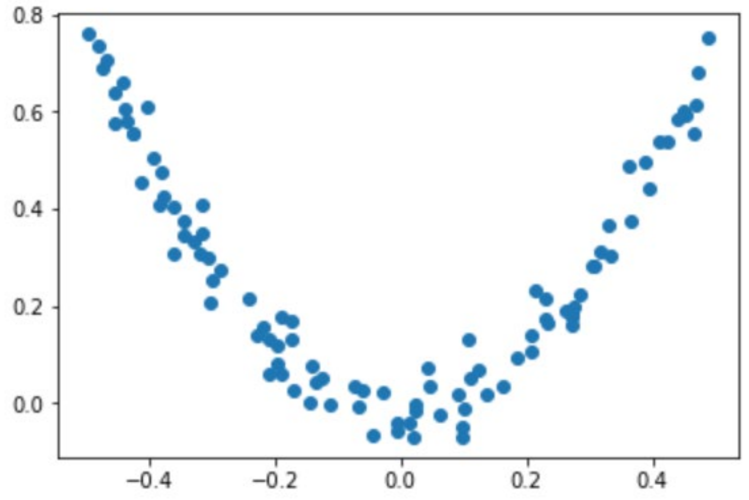
- It can make predictions on a new instance simply by adding up the predictions of all the trees:

```
X_new = np.array([[0.8]])
```

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

```
y_pred
```

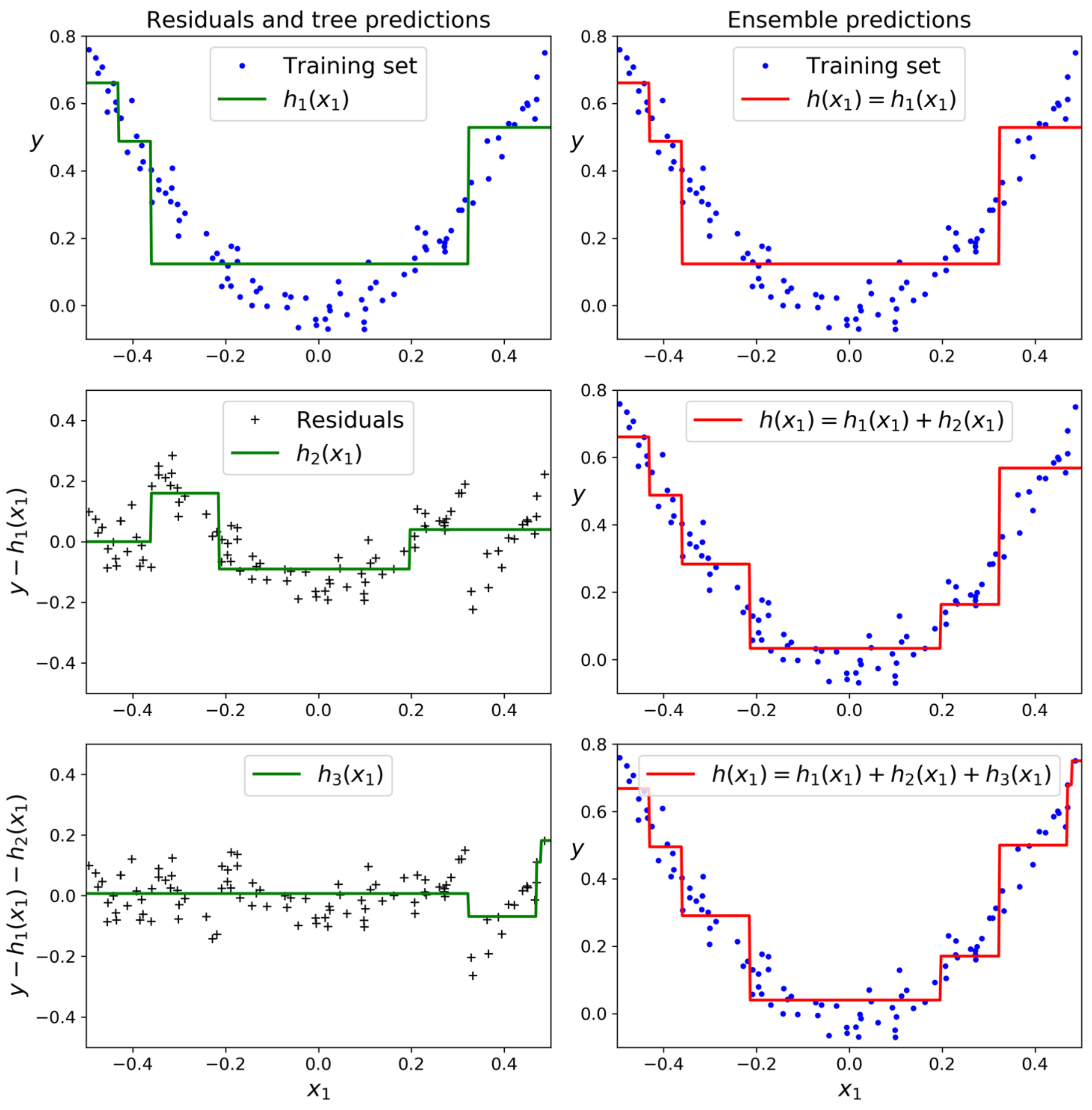
```
array([0.75026781])
```



Example data set

Gradient Boosting: Manually Procedure

- The figure represents the predictions of these three trees in the left column, and the ensemble's predictions in the right column.
- In the first row:
  - the ensemble has just one tree,
  - The ensemble's predictions are exactly the same as the first tree's predictions
- In the second row, a new tree is trained on the residual errors of the first tree
  - the ensemble's predictions are equal to the sum of the predictions of the first two trees.
- In the third row, a third tree is trained on the residual errors of the second tree
  - The ensemble's predictions gradually get better as trees are added to the ensemble.



Gradient Boosting in Scikit-Learn

- The previous slides demonstrated the general principle of gradient boosting.

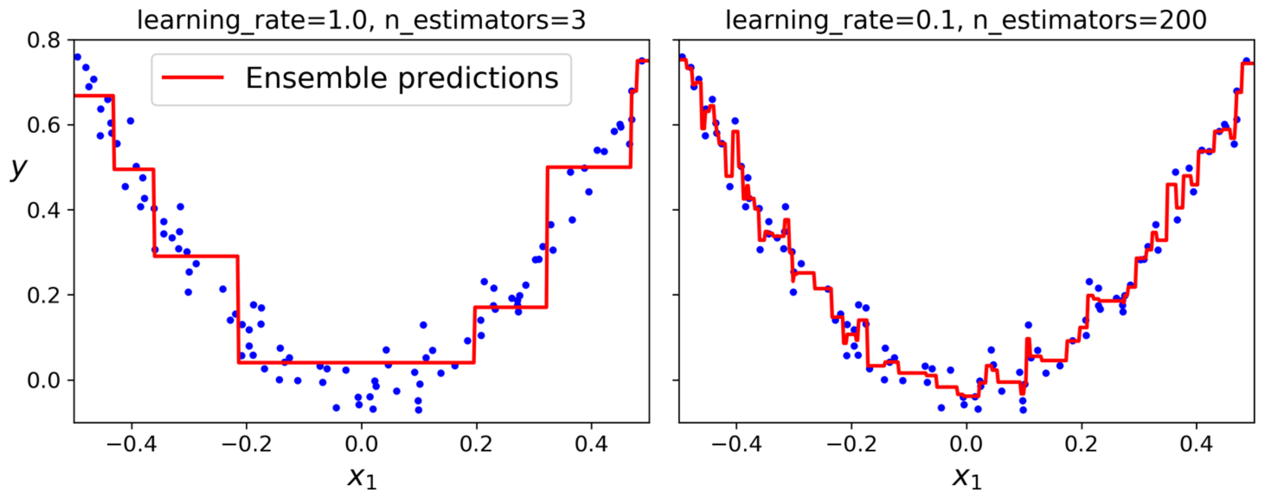
- A more straightforward way to train GBRT ensembles is to use Scikit-Learn's GradientBoostingRegressor class
  - Much like the RandomForestRegressor class, it has hyperparameters to control the growth of Decision Trees (e.g., max_depth, min_samples_leaf),
  - Additionally, the hyperparameters to control the ensemble training, such as the number of trees (n_estimators), are available

```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0, random_state=42)
gbrt.fit(X, y)
```

```
GradientBoostingRegressor(learning_rate=1.0, max_depth=2, n_estimators=3,
                          random_state=42)
```

Gradient Boosting: Learning Rate

- Hyperparamter: learning rate
  - The learning_rate hyperparameter scales the contribution of each tree.
  - If you set it to a low value, such as 0.1, you will need more trees in the ensemble to fit the training set, but the predictions will usually generalize better.
  - This is a regularization technique called shrinkage.
  - The Figure shows two GBRT ensembles trained with a low learning rate:
    - the one on the left does not have enough trees to fit the training set,
    - the one on the right has too many trees and overfits the training set.



Gradient Boosting: Early Stopping

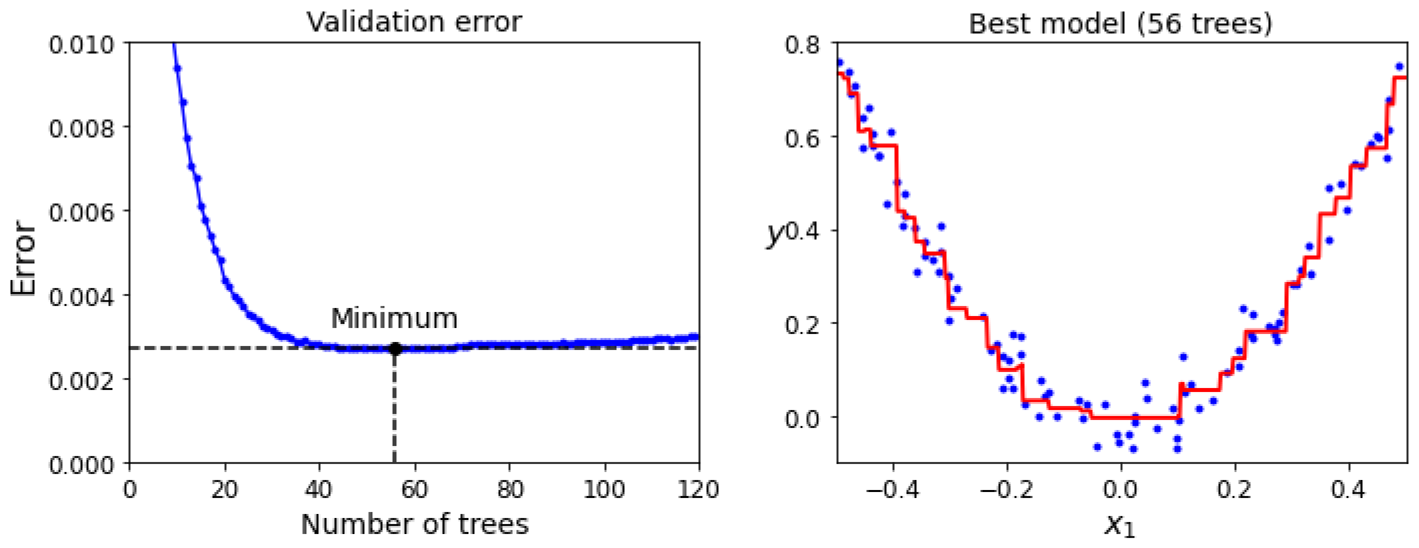
- How to avoid overfitting and how to find the optimal number of trees?

- Solution: early stopping
  - A simple way to implement this is to use the staged_predict() method:
    - it returns an iterator over the predictions made by the ensemble at each stage of training (with one tree, two trees, etc.).
  - The following code trains a GBRT ensemble with 120 trees, then measures the validation error at each stage of training to find the optimal number of trees, and finally trains another GBRT ensemble using the optimal number of trees:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=49)

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=120, random_state=42)
gbrt.fit(X_train, y_train)

errors =[mean_squared_error(y_val, y_pred)
         for y_pred in gbrt.staged_predict(X_val)]
bst_n_estimators = np.argmin(errors) + 1

gbrt_best = GradientBoostingRegressor(max_depth=2, n_estimators=bst_n_estimators, random_state=42)
gbrt_best.fit(X_train, y_train)
```

Gradient Boosting

- The validation errors are represented on the left and the best model’s predictions are represented on the right.



Gradient Boosting: Advanced Concepts

- Advanced concepts:
  - There are various hyperparameters for GradientBoostingRegressor
    - E.g. Other cost functions
    - Using only a subsample of the training data for each tree
      → **Stochastic Gradient Boosting**
  - A complete list of hyperparameters can be found in the Scikit-Learn's documentation
    - Regression: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html
    - Classification: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

You can learn all this in an interactive online course!

https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/boosting?ex=5

Gradient Boosting (GB)

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

datacamp

You can learn all this in an interactive online course!

https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/boosting?ex=9

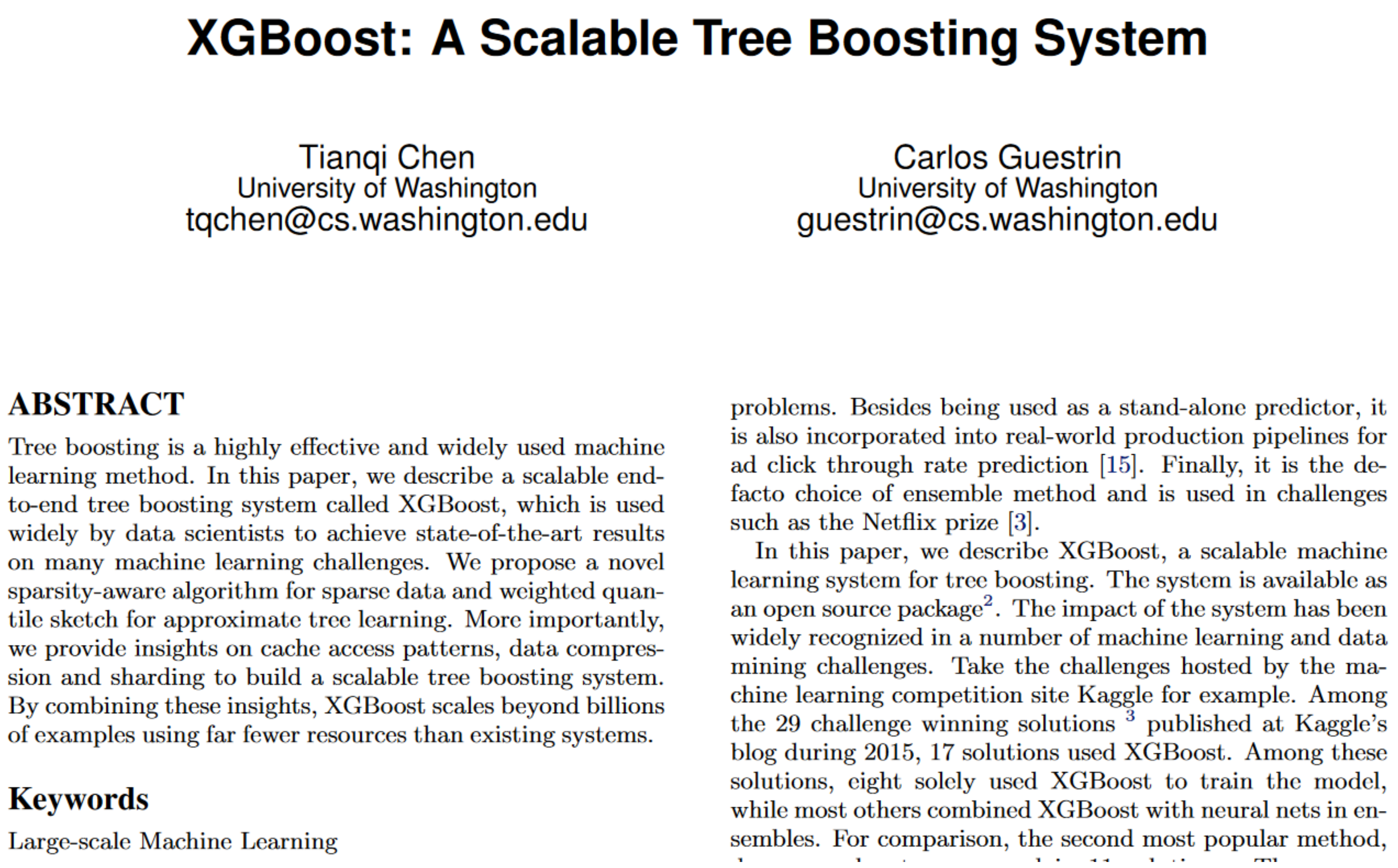
Stochastic Gradient Boosting (SGB)

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

Ensemble Learning and Random Forests

Extreme Gradient Boosting (XGBoost)



XGBoost: A Scalable Tree Boosting System

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

Carlos Guestrin
University of Washington
guestrin@cs.washington.edu

ABSTRACT

Tree boosting is a highly effective and widely used machine learning method. In this paper, we describe a scalable end-to-end tree boosting system called XGBoost, which is used widely by data scientists to achieve state-of-the-art results on many machine learning challenges. We propose a novel sparsity-aware algorithm for sparse data and weighted quantile sketch for approximate tree learning. More importantly, we provide insights on cache access patterns, data compression and sharding to build a scalable tree boosting system. By combining these insights, XGBoost scales beyond billions of examples using far fewer resources than existing systems.

Keywords

Large-scale Machine Learning

problems. Besides being used as a stand-alone predictor, it is also incorporated into real-world production pipelines for ad click through rate prediction [15]. Finally, it is the de-facto choice of ensemble method and is used in challenges such as the Netflix prize [3].

In this paper, we describe XGBoost, a scalable machine learning system for tree boosting. The system is available as an open source package[2]. The impact of the system has been widely recognized in a number of machine learning and data mining challenges. Take the challenges hosted by the machine learning competition site Kaggle for example. Among the 29 challenge winning solutions [3] published at Kaggle's blog during 2015, 17 solutions used XGBoost. Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles. For comparison, the second most popular method,

Extreme Gradient Boosting

- XGBoost is currently one of the most widely used supervised learning algorithms
  - Based on ensemble learning
  - Based on boosting
  - Mostly based on decision or regression trees

- XGBoost is an optimized implementation of gradient boosting (see slides before)

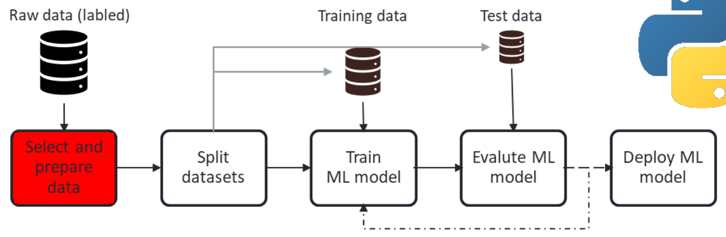
- XGBoost was originally implemented in C++ and has gained popularity
  - XGBoost is fast due to parallelization
    - Can easily be used on GPU
    - Possible to train on even huge data
  - XGBoost has a high performance

- XGBoost in Scikit-Learn offers several nice features, such as
  - automatically taking care of early stopping
  - performant cross-validation

Remember the Running Example

- As a healthcare data analyst, your job is to identify patients or sufferers that have a higher chance of a particular disease, for example, diabetes
- These predictions will help you to treat patients before the disease occurs
- As an analyst, you have first to define the problem that you want to solve using classification and then identify the potential features that predict the labels accurately
  - Features are the columns or attributes that are responsible for prediction.
  - In diabetes prediction problems, health analysts will collect patient information, such as the number of pregnancies the patient has had, their BMI, insulin level, age.
- Terminology:
  - Classification is the process of categorizing customers into two or more categories.
  - The classification model predicts the categorical class label, such as whether the customer is potential or not.
  - In the classification process, the model is trained on available data, makes predictions, and evaluates the model performance.
  - Developed models are called classifiers.

XGBoost: Python Example



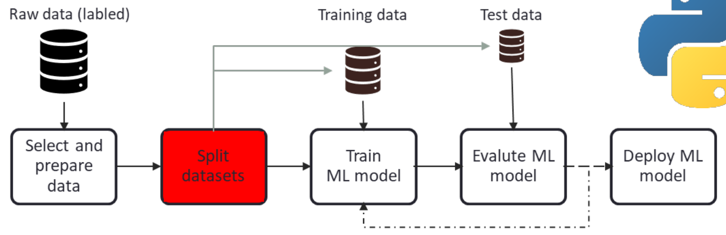
Diabetes data: Select and prepare data

```
from sklearn.datasets import load_iris
import pandas as pd
diabetes = pd.read_csv("diabetes.csv")
diabetes.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# Splitting dataset in two parts: feature set and target label
feature_set = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
features = diabetes[feature_set]
target = diabetes.label
```

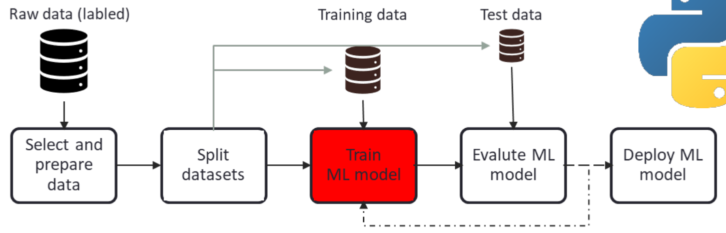
XGBoost: Python Example



Diabetes Data: Split dataset

```
# Partition data into training and testing set
from sklearn.model_selection import train_test_split
feature_train,feature_test, target_train, target_test = train_test_split(features, target, test_size=0.3,
random_state=42)
```

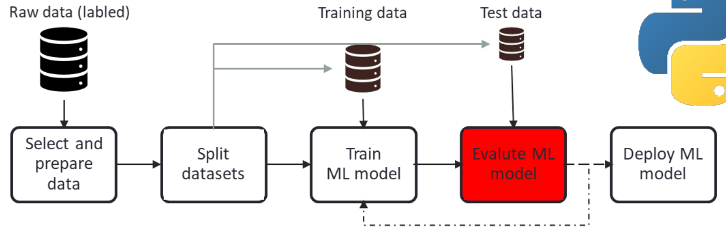
XGBoost: Python Example



Diabetes Data: train XGBoost

```
# Instantiate the XGBClassifier with 25 boosting rounds and error evaluation metric
xgb_clf = xgb.XGBClassifier(n_estimators=100, eval_metric='error', random_state=42)

# Fit the classifier to the  training set
xgb_clf.fit(feature_train,target_train)
```

XGBoost: Python Example



Diabetes Data: evaluate XGBoost model

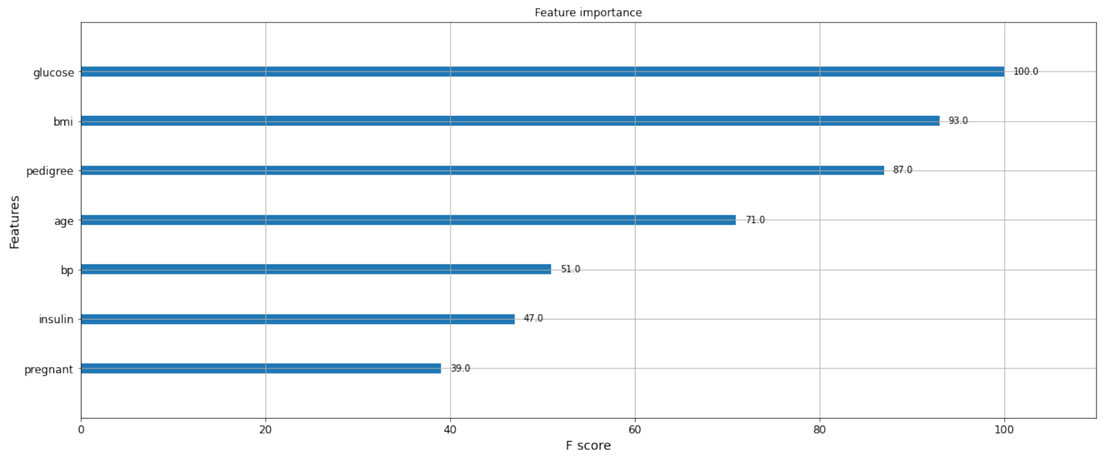
```
target_pred = xgb_clf.predict(feature_test)
print(accuracy_score(target_test, target_pred))
```

```
0.7662337662337663
```

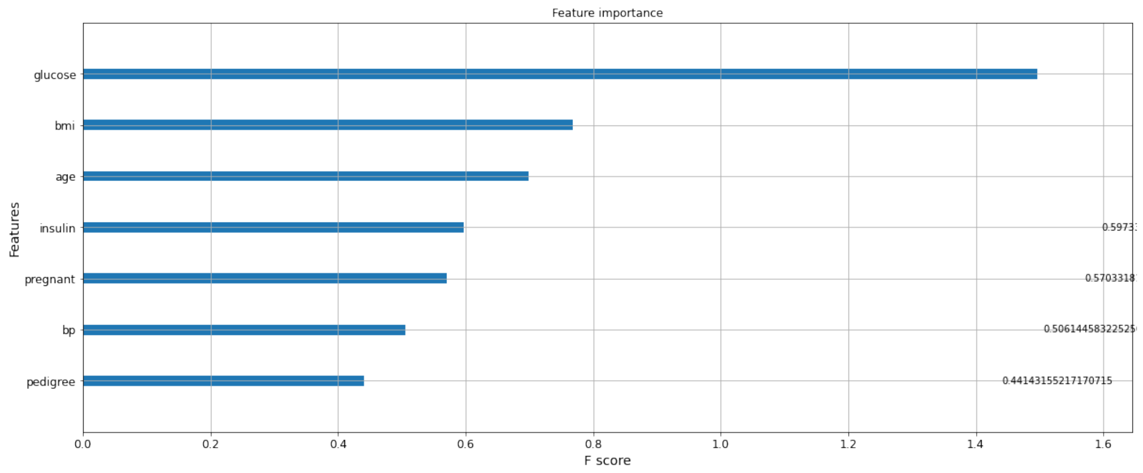
The model has an accuracy of 0.77

Extreme Gradient Boosting: Feature Importance

xgboost.plot_importance(xgb_clf, importance_**type="weight"**)



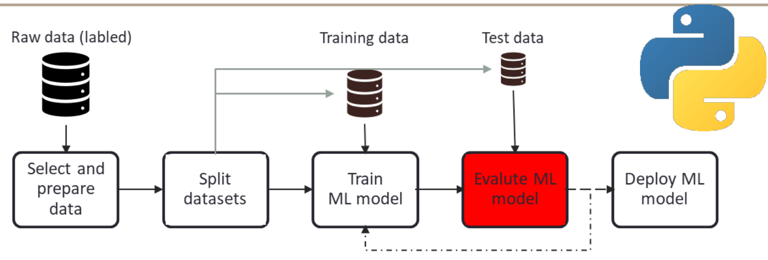
xgboost.plot_importance(xgb_clf, importance_**type="gain"**)

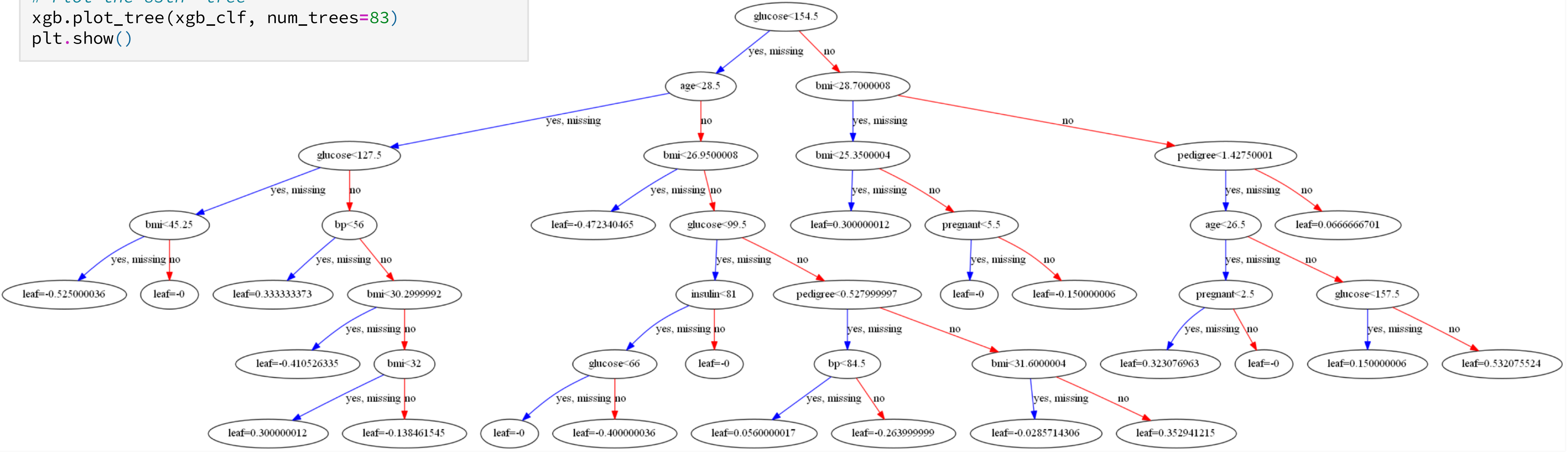


- Here we see the ordering of features differs quite a bit between gain and weight!
- This implies that
  - (1) feature importance can be subjective, and
  - (2) the number of appearances a feature has on a tree is not necessarily correlated to how much gain it brings.
- For example, a binary variable has less chance to appear as many times as a continuous variable on a tree since there are only two outputs.
- However, it can still be a powerful feature

XGBoost: Python Example

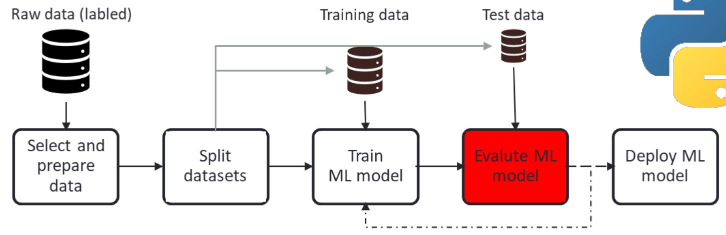
```python
##set up the parameters
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 80,50

# Plot the first tree
xgb.plot_tree(xgb_clf, num_trees=0)
plt.show()

# Plot the 50th tree
xgb.plot_tree(xgb_clf, num_trees=50)
plt.show()

# Plot the 83th  tree
xgb.plot_tree(xgb_clf, num_trees=83)
plt.show()
```

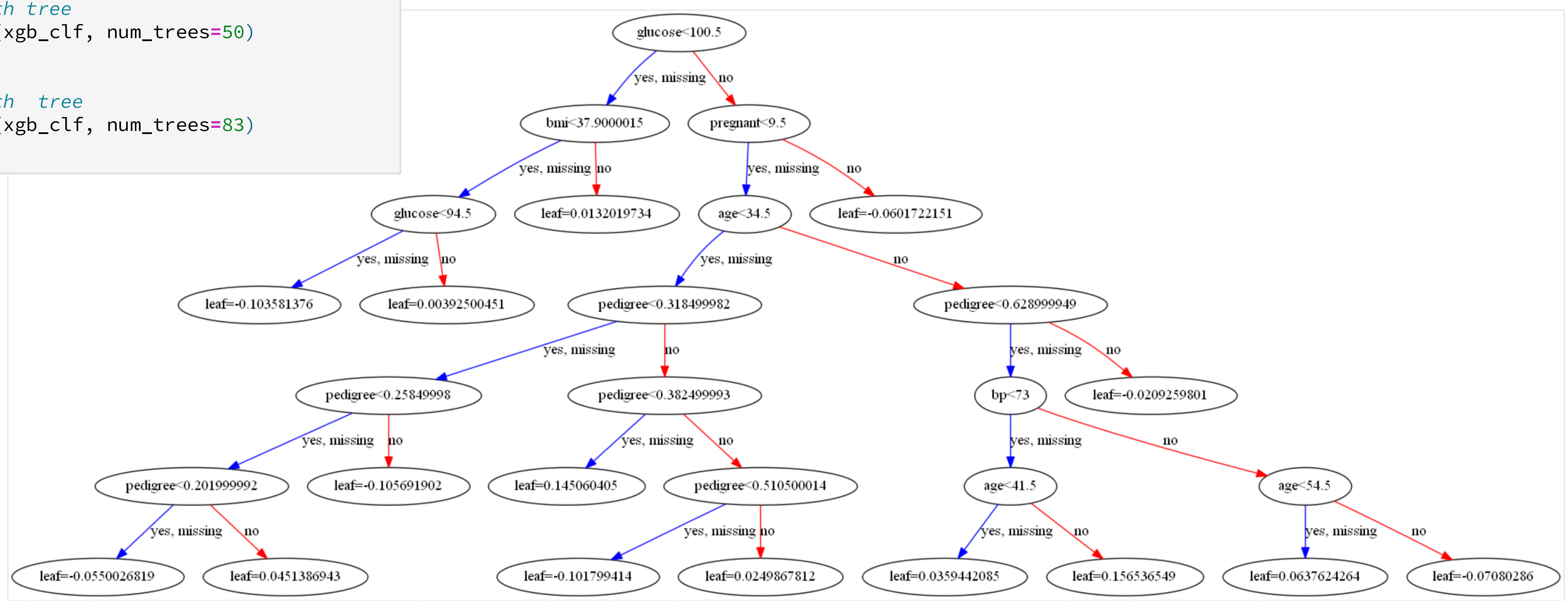




XGBoost: Python Example

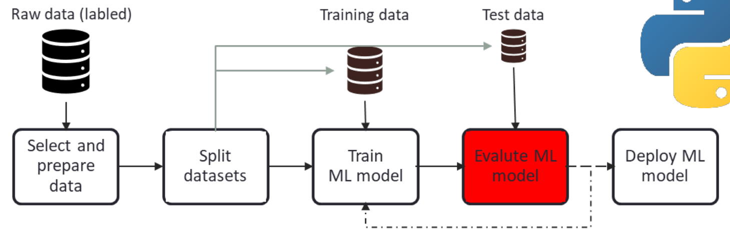
```
##set up the parameters
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 80,50

# Plot the first tree
xgb.plot_tree(xgb_clf, num_trees=0)
plt.show()

# Plot the 50th tree
xgb.plot_tree(xgb_clf, num_trees=50)
plt.show()

# Plot the 83th  tree
xgb.plot_tree(xgb_clf, num_trees=83)
plt.show()
```

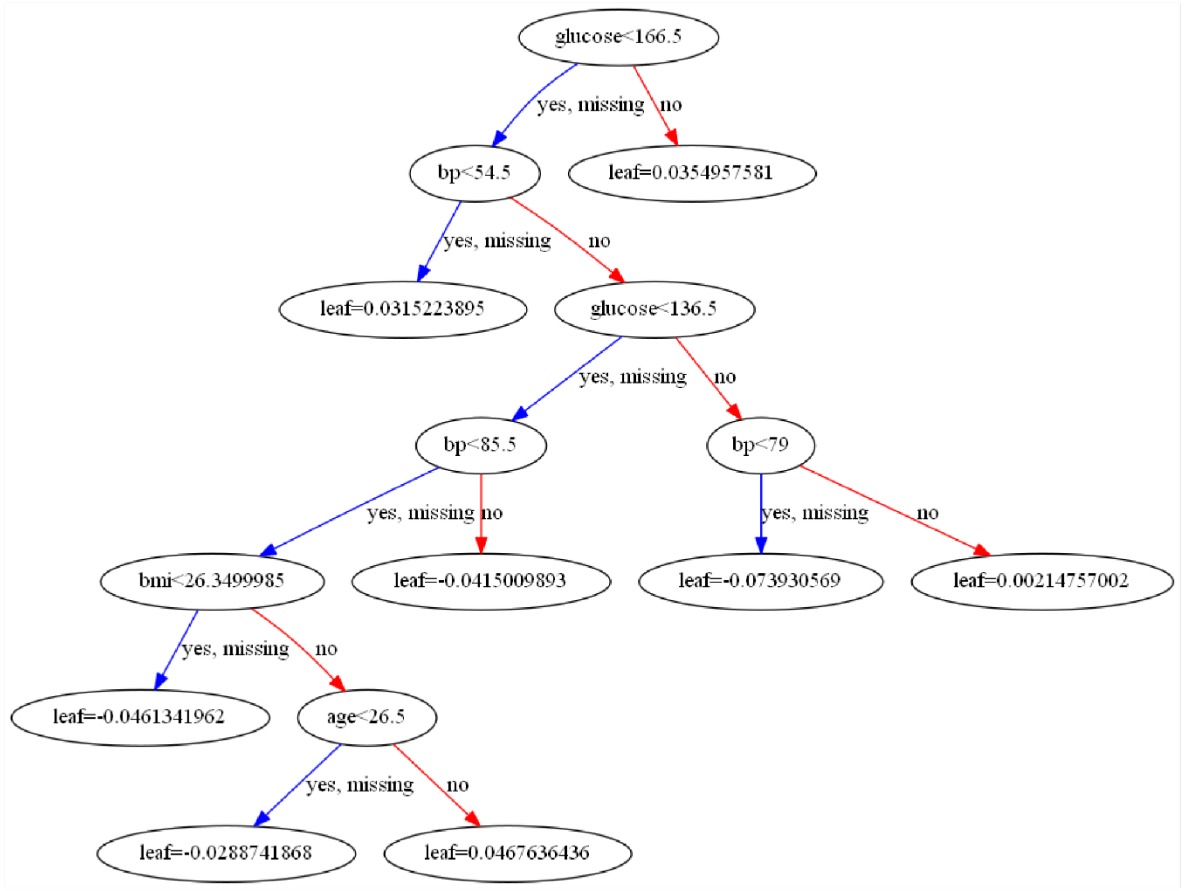




XGBoost: Python Example

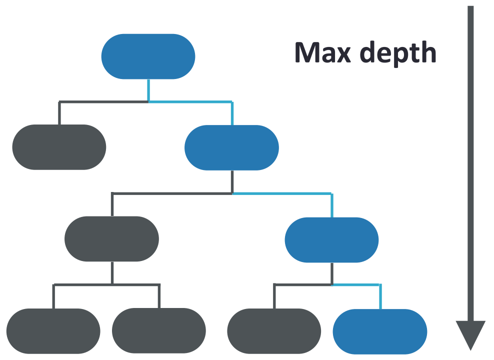
```
##set up the parameters
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 80,50

# Plot the first tree
xgb.plot_tree(xgb_clf, num_trees=0)
plt.show()

# Plot the 50th tree
xgb.plot_tree(xgb_clf, num_trees=50)
plt.show()

# Plot the 83th  tree
xgb.plot_tree(xgb_clf, num_trees=83)
plt.show()
```





Extreme Gradient Boosting: Hyperparameter

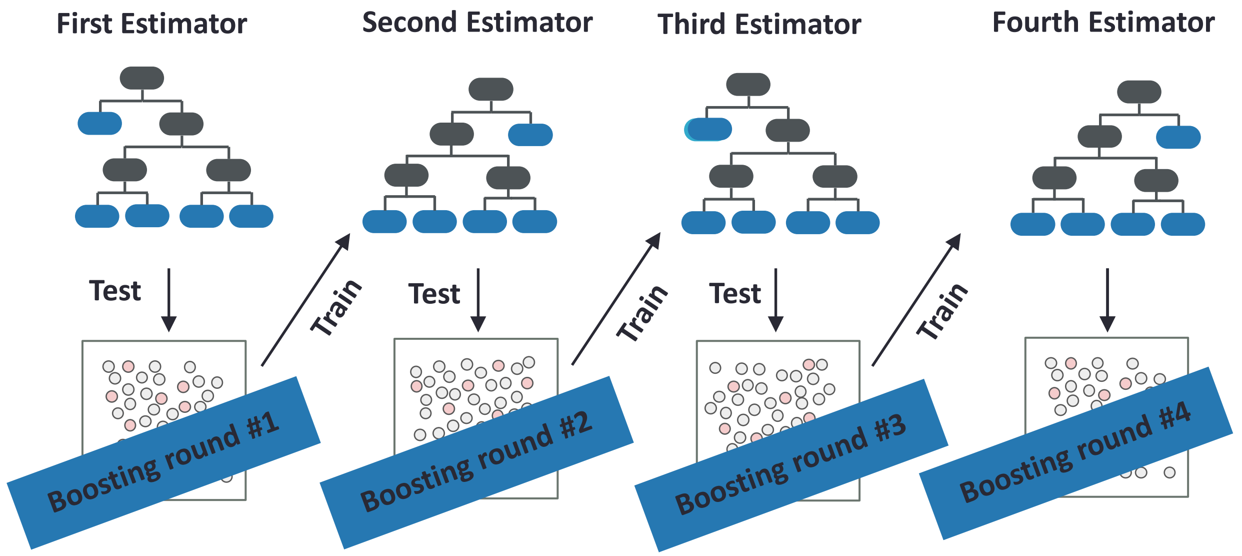
- Hyperparamter
  - There are many parameters for our tree booster
  - Full list: https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-booster
- Most important hyperparameter:
  - Max depth
    - Maximum depth of a tree.
    - Increasing this value will make the model more complex and more likely to overfit.



Extreme Gradient Boosting: Hyperparameter

- Hyperparamter
  - There are many parameters for our tree booster
  - Full list: https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-booster

- Most important hyperparameter:
  - n_estimators
    - the number of gradients boosted trees we want in our model.
    - It's equivalent to the number of boosting rounds.



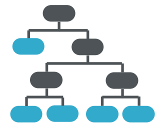
Extreme Gradient Boosting: Hyperparameter

- Hyperparamter
  - There are many parameters for our tree booster
  - Full list: https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-booster

- Most important hyperparameter:
  - colsample_bytree:
    - The subsample ratio of columns when constructing each tree.
    - Subsampling occurs once for every tree constructed.
    - Essentially, this lets us limit the number of columns used when constructing each tree.
    - This adds randomness, making the model more robust to noise.
    - The default is 1 (i.e., all the columns)

`colsample_bytree = 0.5`

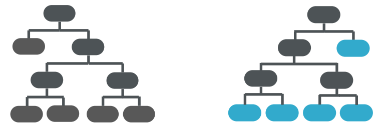
col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9	col_10
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…



For the first tree five out of ten columns (=0.5) are randomly selected.

`colsample_bytree = 0.5`

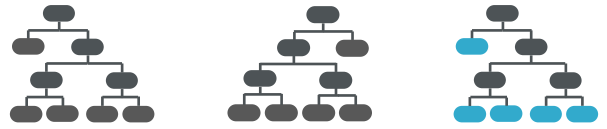
col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9	col_10
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…



For the second tree again five out of ten columns are randomly selected….

`colsample_bytree = 0.5`

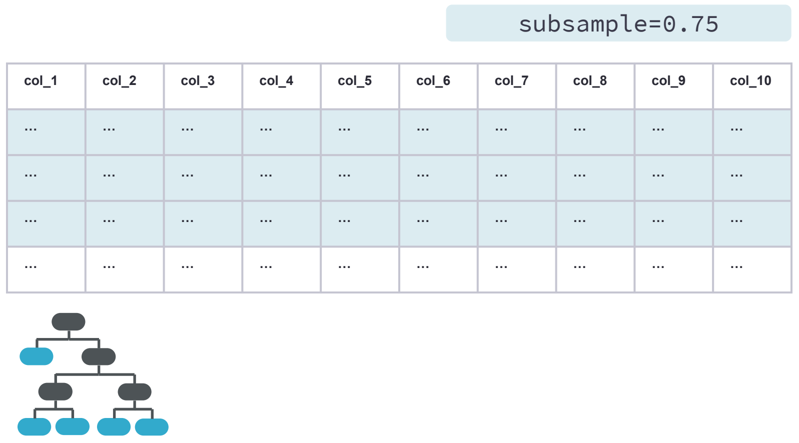
col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9	col_10
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…



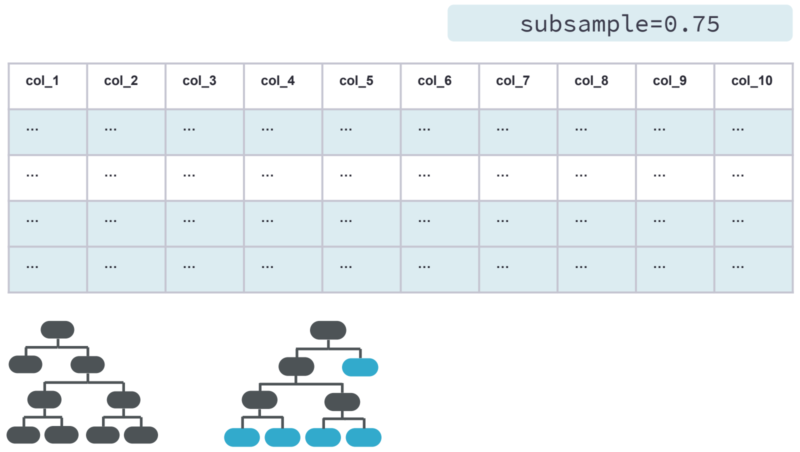
Extreme Gradient Boosting: Hyperparameter

- Hyperparamter
  - There are many parameters for our tree booster
  - Full list: https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-booster

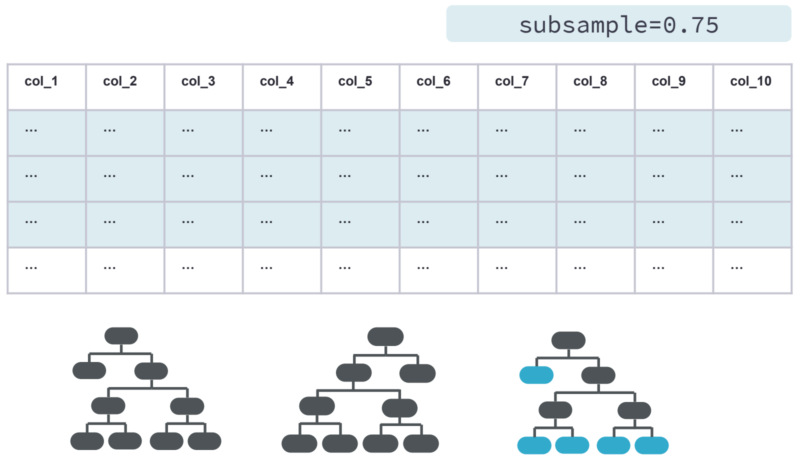
- Most important hyperparameter:
  - subsample
    - Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees preventing overfitting.
    - Subsampling will occur once in every boosting iteration.
    - range: (0,1]


subsample=0.75
col_1 col_2 col_3 col_4 col_5 col_6 col_7 col_8 col_9 col_10
...

For the first tree three out of four (=0.75) rows are randomly selected.


subsample=0.75
col_1 col_2 col_3 col_4 col_5 col_6 col_7 col_8 col_9 col_10
...

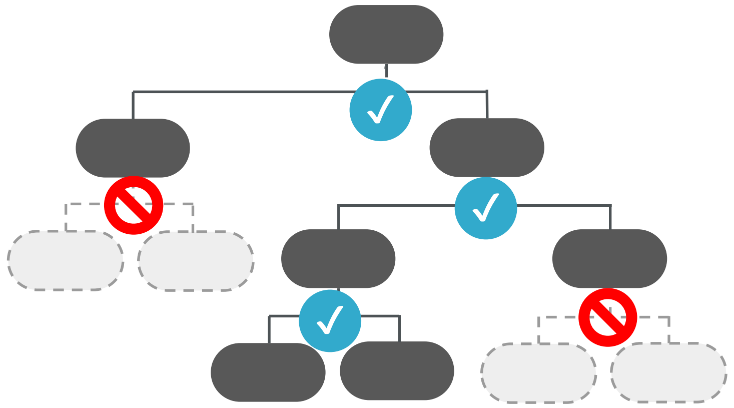
For the second tree three out of four (=0.75) rows are randomly selected....


subsample=0.75
col_1 col_2 col_3 col_4 col_5 col_6 col_7 col_8 col_9 col_10
...

Extreme Gradient Boosting: Hyperparameter

- Hyperparamter
  - There are many parameters for our tree booster
  - Full list: https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-booster

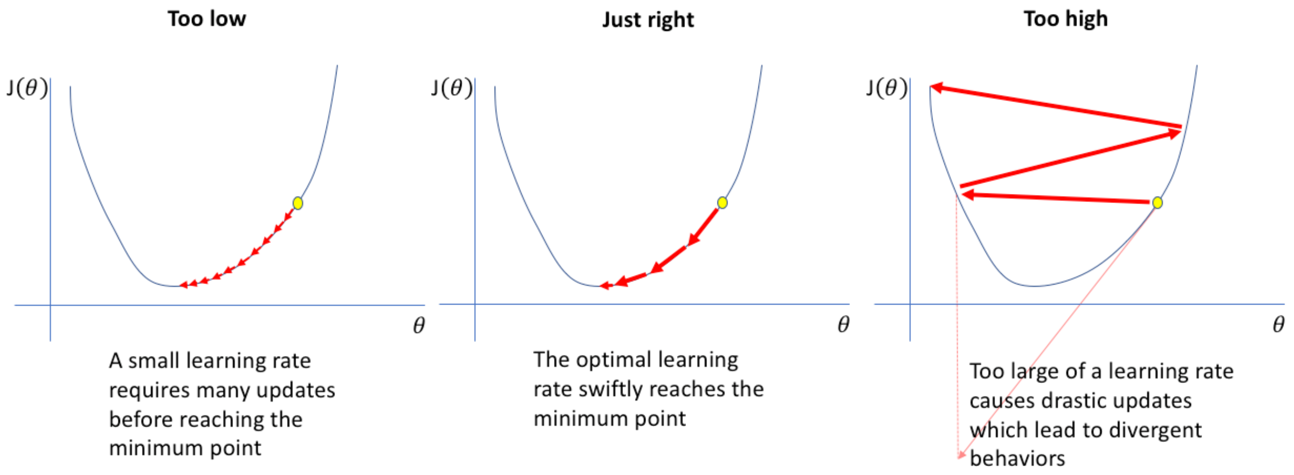
- Most important hyperparameter:
  - gamma
    - Minimum loss reduction is required to make a further partition on a leaf node of the tree.
    - The larger gamma is, the more conservative the algorithm will be.
    - range: [0,∞]
    - This decides whether a node will split based on the expected loss reduction after the split.
    - Increasing gamma = less splits = less complexity



Extreme Gradient Boosting: Hyperparameter

- Hyperparamter
  - There are many parameters for our tree booster
  - Full list: https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-booster

- Most important hyperparameter:
  - Learning rate
    - Step size shrinkage was used in the update to prevent overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative.
    - range: [0,1]
    - Gradient boosting works by sequentially adding weak learners to the model.
    - Each new weak learner attempts to correct the residual errors from the preceding trees.
    - This makes the model very susceptible to overfitting. Learning rate can help slow down learning by shrinking the resulting weights of the current tree before passing them on to the next tree.


Too low
Just right
Too high
$J(\theta)$
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

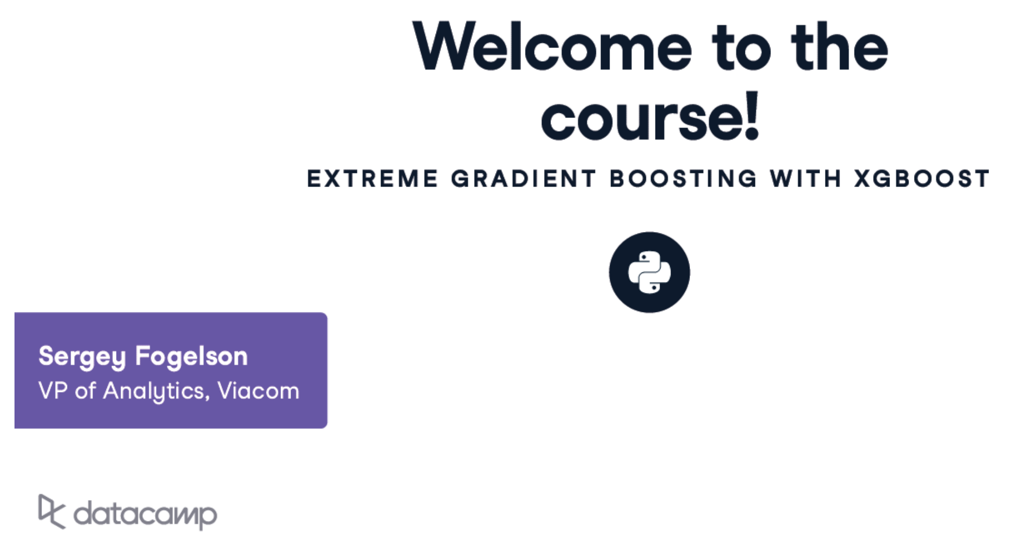
Extreme Gradient Boosting: Hyperparameter

- Hyperparamter
  - There are many parameters for our tree booster
  - Full list: https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-booster

- Most important hyperparameter:
  - reg_alpha
    - L1 regularization term on weights. Increasing this value will make model a more conservative.
    - L1 is often referred to as lasso regression.
    - It is a foundational regularization technique, meaning it aims to reduce overfitting by discouraging complex models.
    - In the case of gradient boosting, L1 does this by adding penalties on leaf weights. Increasing reg_alpha drives base learners' leaf weights towards 0.
    - Default is 0, meaning there is no alpha regularization in our model currently. Let us activate L1 with a value of 0.01.

You can learn all this in an interactive online course!

https://app.datacamp.com/learn/courses/extreme-gradient-boosting-with-xgboost



Welcome to the course!

EXTREME GRADIENT BOOSTING WITH XGBOOST

Sergey Fogelson
VP of Analytics, Viacom

datacamp

Ensemble Learning: Learning objectives

- After this lecture, students shall be able to…
  - explain how voting classifiers work,
  - how bagging and pasting ensemble learning works,
  - how boosting works, and
  - to differentiate the different ensemble learning methods.