MODEL LIFECYCLE

BATALHA DE VIZINHOS I

DIEGO RODRIGUES DSC

INFNET

MODEL LIFECYCLE : BATALHA DE VIZINHOS I

PARTE 1 : TEORIA

- MODELING
  - MODELOS MULTICLASSE
  - ENSEMBLE
- EVALUATION
  - LEAVE ONE OUT

PARTE 2 : PRÁTICA

- NOTEBOOK
  CLASSIFICAÇÃO IRIS LOO

Produzir Ação

CICLO DE VIDA DO MODELO

Baseado em Dados

AMBIENTE PYTHON

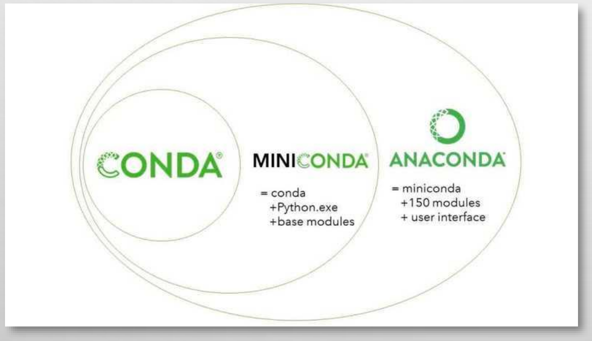
4. Variáveis Aleatórias

5. Visualização

6. Estimação e Inferência

7. Machine Learning

1. Editor de Código

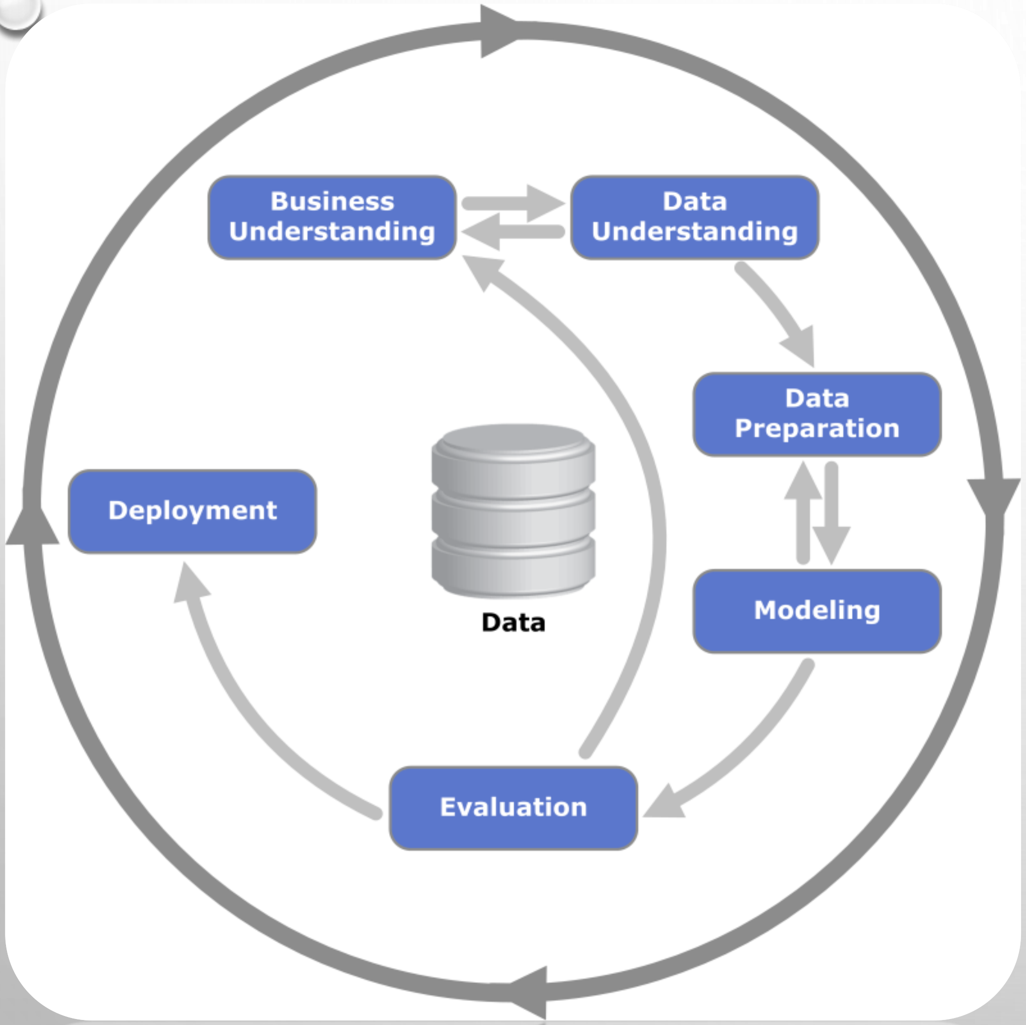


2. Gestor de Ambiente

3. Ambiente Python do Projeto

3. Notebook Dinâmico

Cross Industry Standard Process for Data Mining - IBM



1) Requerimentos e Análise de Negócio

Entendimento do problema decisório, dados relacionados & revisão bibliográfica.

2) Preparação dos Dados

Entendimento das fontes de dados, dos tipos e elaboração da representação.

3) Modelagem

Análise Exploratória, Seleção de atributos e treinamento.

4) Avaliação

Seleção do melhor modelo.

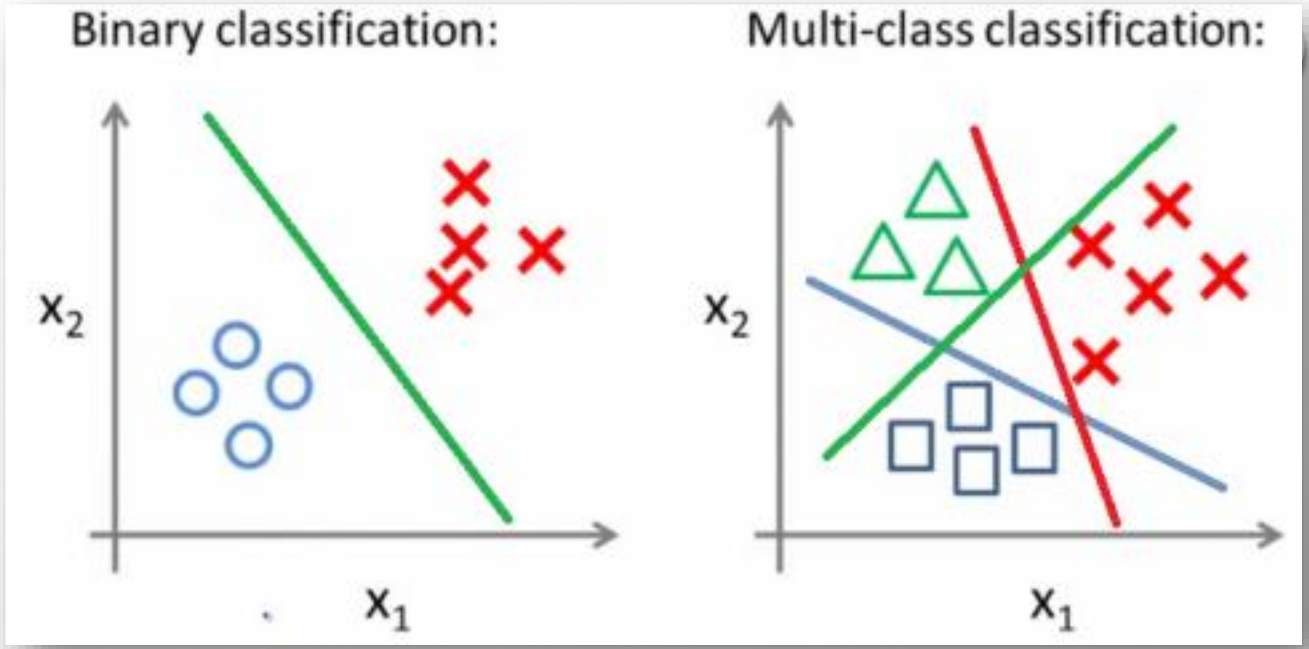
5) Liberação

Liberação do modelo no ambiente de produção.

MODELING

MODELOS MULTICLASSE

A **classificação multiclasse** é uma tarefa de aprendizado supervisionado onde o objetivo é prever **uma entre três ou mais classes distintas.**

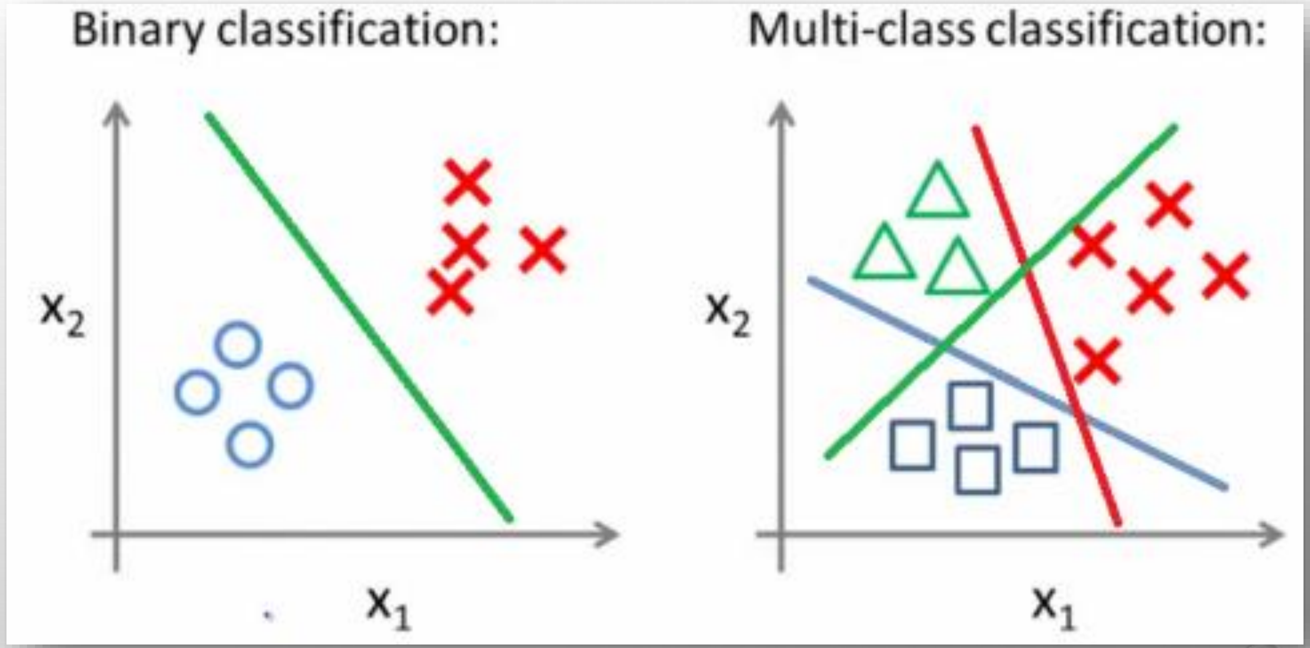


Diferente da **classificação binária** (apenas duas classes), aqui o modelo precisa aprender a **diferenciar múltiplas categorias.**

Desafios
Desequilíbrio entre classes
Confusão entre classes semelhantes
Avaliação mais complexa (precisão, recall, F1 por classe)

MODELOS MULTICLASSE

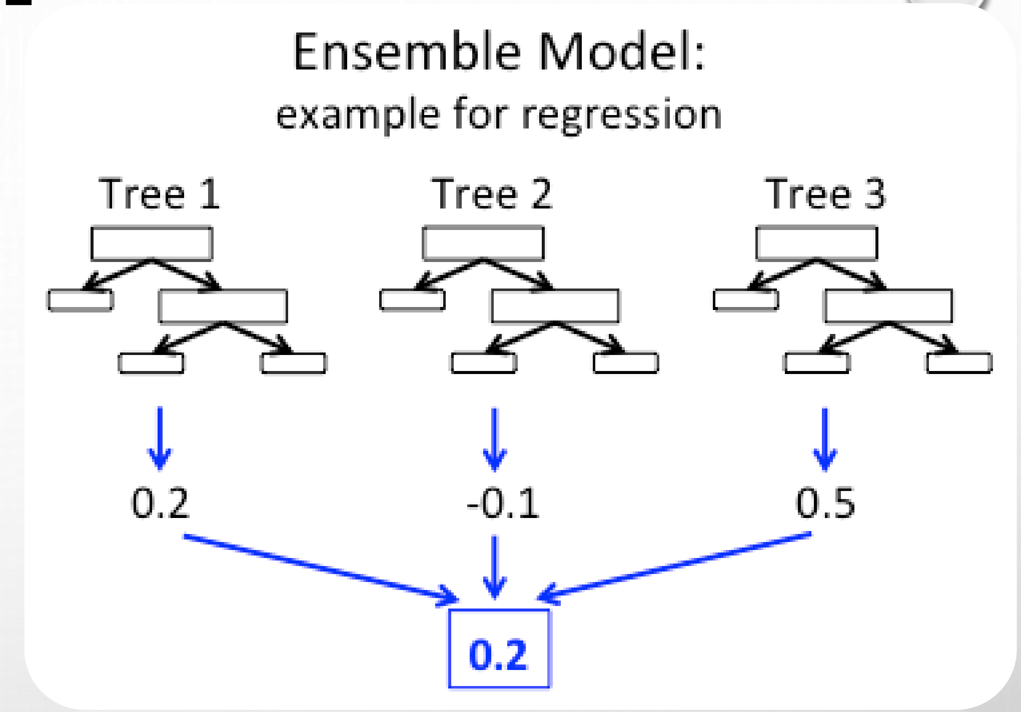
- Árvores de Decisão
- Redes Neurais
- Naive Bayes
- **K Vizinhos Mais Próximos**
- Regressão Logística Multinomial
- **Máquina de Comitê (Ensemble)**



ALGORITMOS BASEADOS EM ENSEMBLE

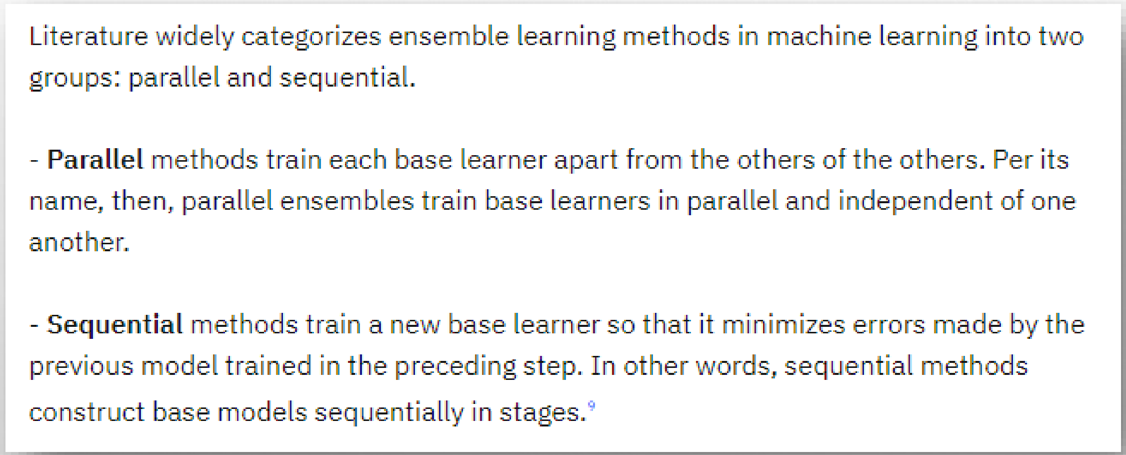
Algoritmos que **combinam modelos simples,** usualmente através de **votação ou ponderação,** para atingir maiores taxas de classificação.

1) Random Forest
2) Boosting



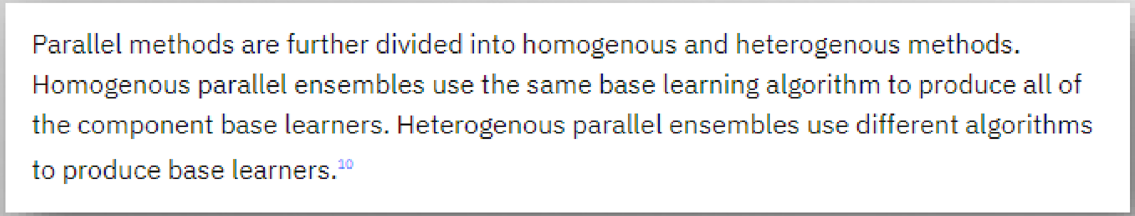
Boa **capacidade de generalização** gerado através de **arranjos complexos** de múltiplos modelos simples de machine learning.

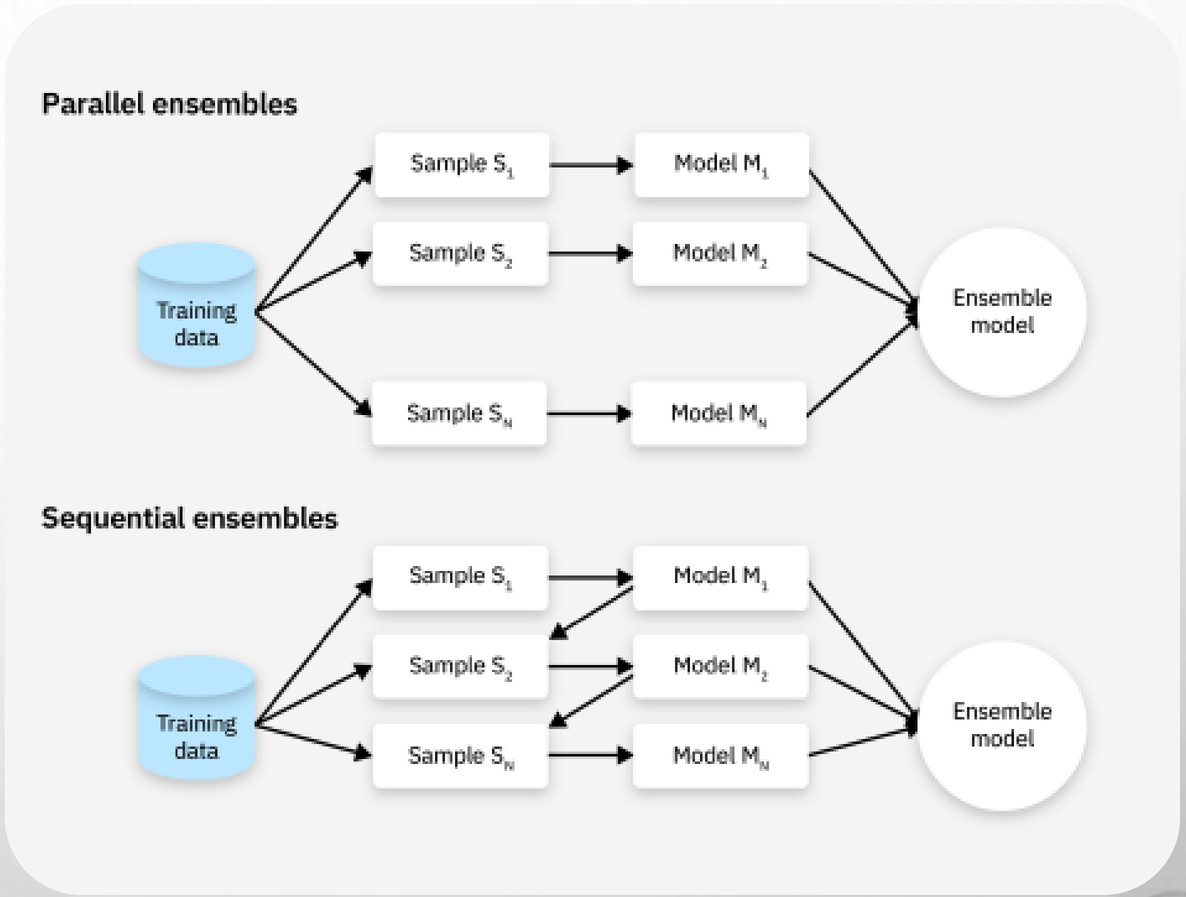
TIPOS DE COMITÊ


Literature widely categorizes ensemble learning methods in machine learning into two groups: parallel and sequential.

- **Parallel** methods train each base learner apart from the others of the others. Per its name, then, parallel ensembles train base learners in parallel and independent of one another.

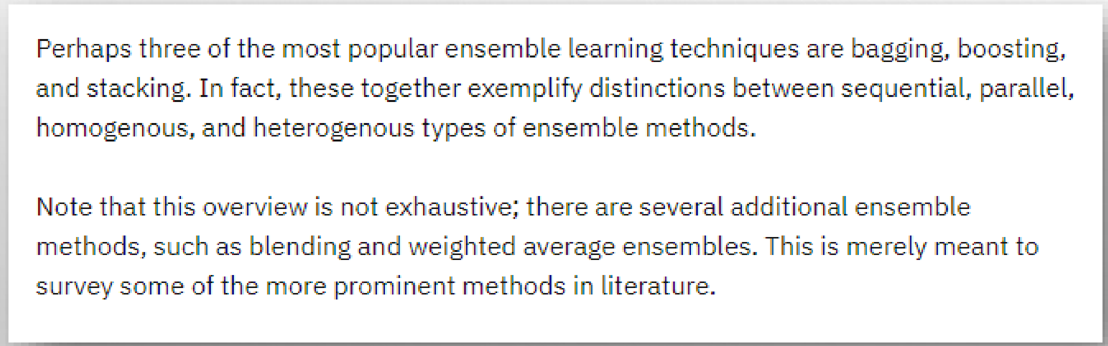
- **Sequential** methods train a new base learner so that it minimizes errors made by the previous model trained in the preceding step. In other words, sequential methods construct base models sequentially in stages.[9]


Parallel methods are further divided into homogenous and heterogenous methods. Homogenous parallel ensembles use the same base learning algorithm to produce all of the component base learners. Heterogenous parallel ensembles use different algorithms to produce base learners.[10]

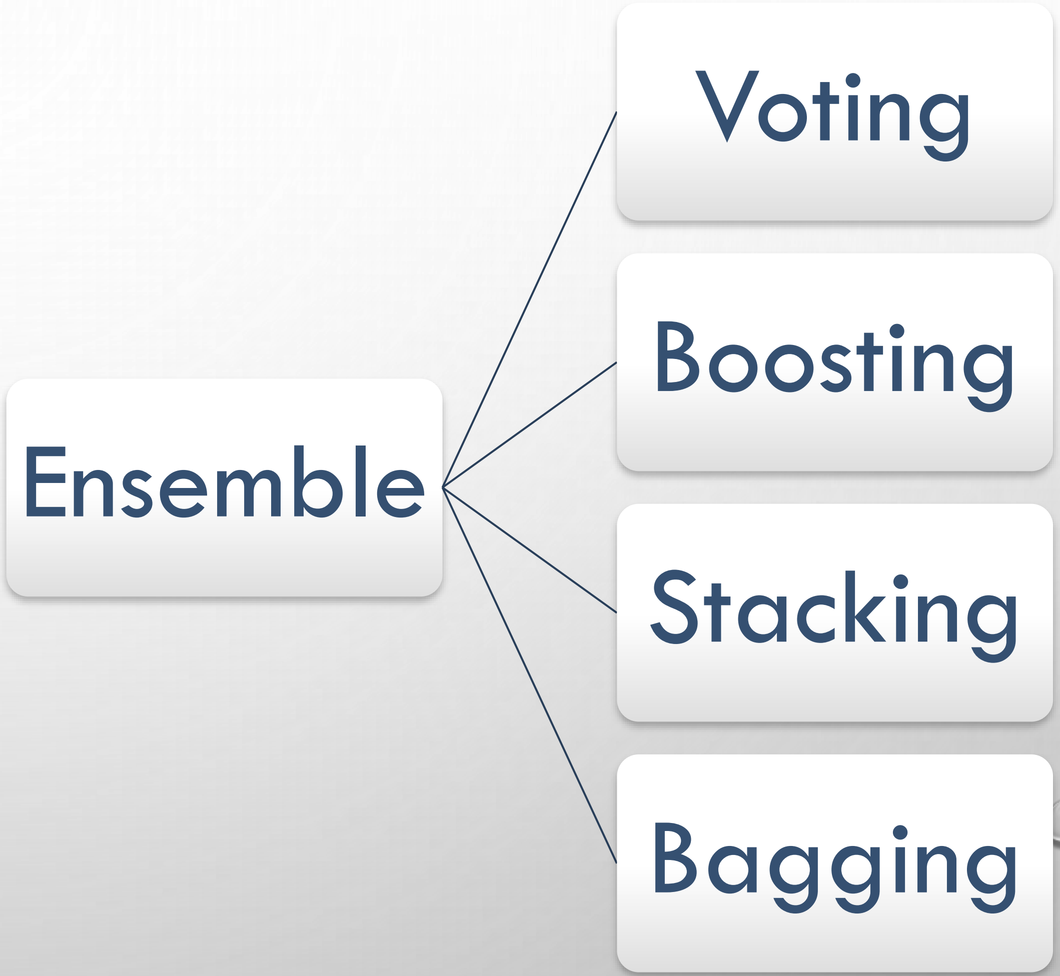


https://www.ibm.com/topics/ensemble-learning

TÉCNICAS DE TREINAMENTO


Perhaps three of the most popular ensemble learning techniques are bagging, boosting, and stacking. In fact, these together exemplify distinctions between sequential, parallel, homogenous, and heterogenous types of ensemble methods.

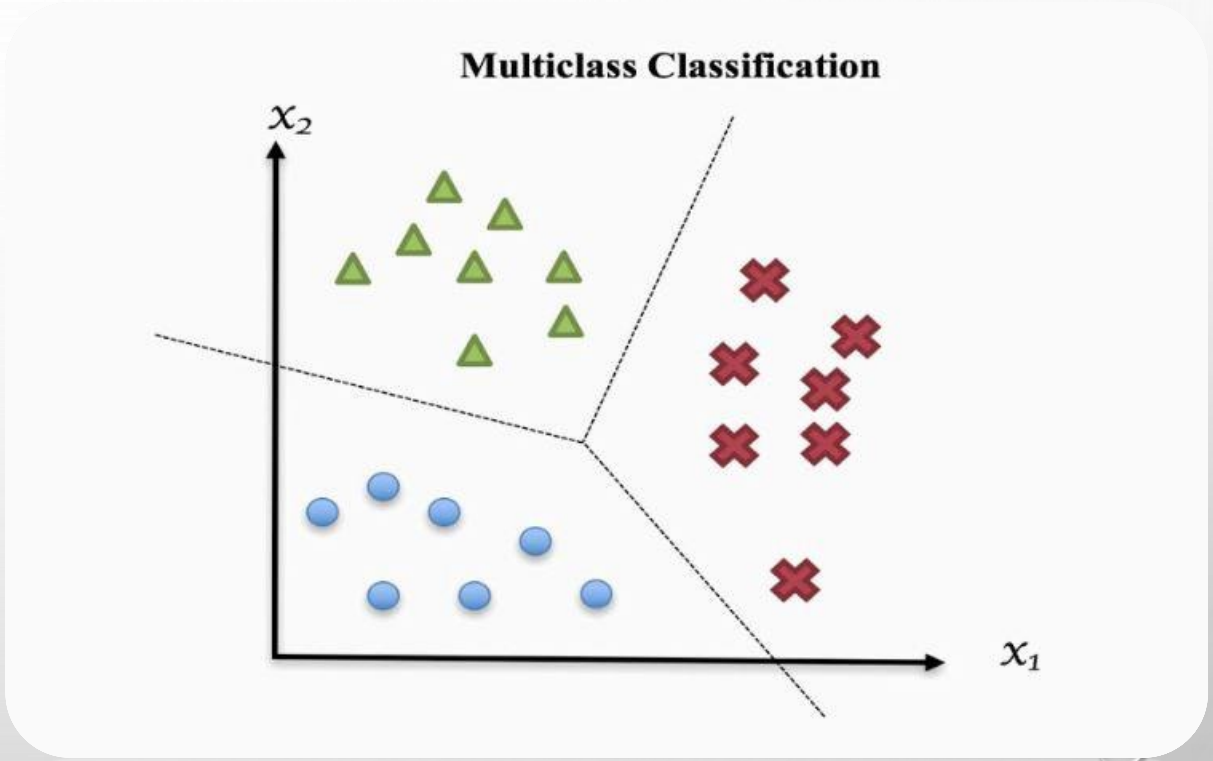
Note that this overview is not exhaustive; there are several additional ensemble methods, such as blending and weighted average ensembles. This is merely meant to survey some of the more prominent methods in literature.



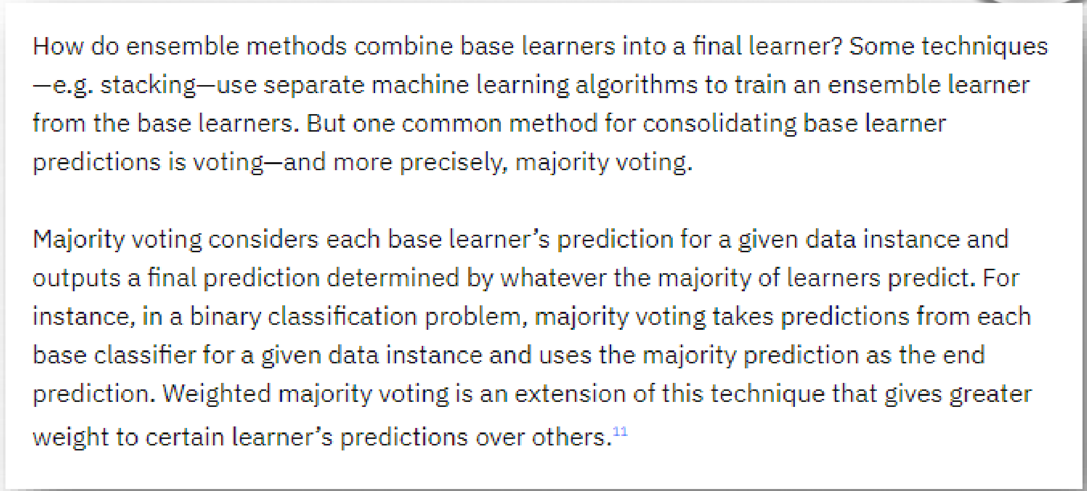
MÁQUINA DE COMITÊ - VOTAÇÃO

Modelos de Votação

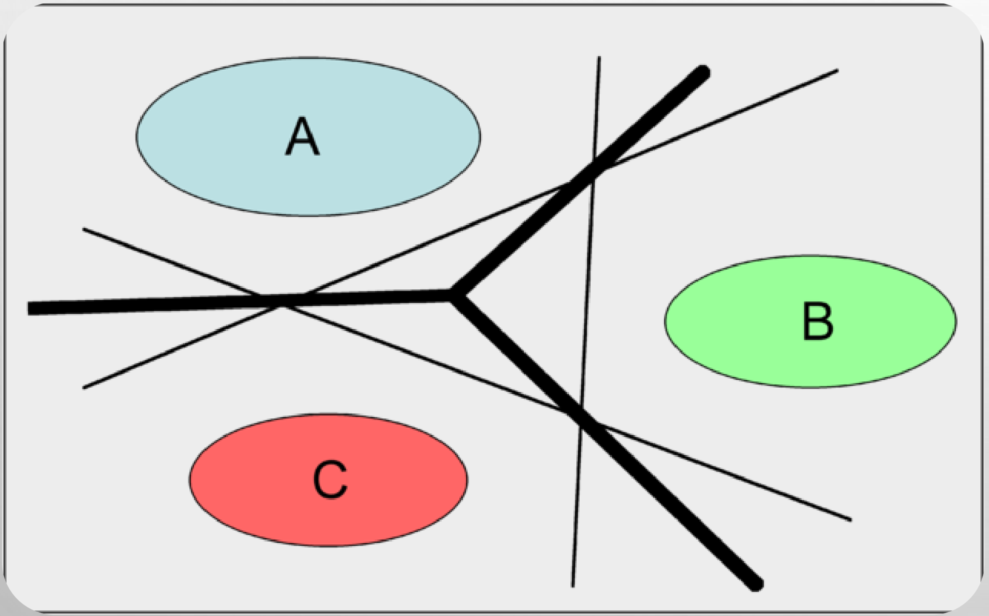
- Discriminar múltiplos objetos em paralelo.
- Quaisquer modelos podem ser combinados.
- Extensão para o caso multiclasse de maneira simples e intuitiva.



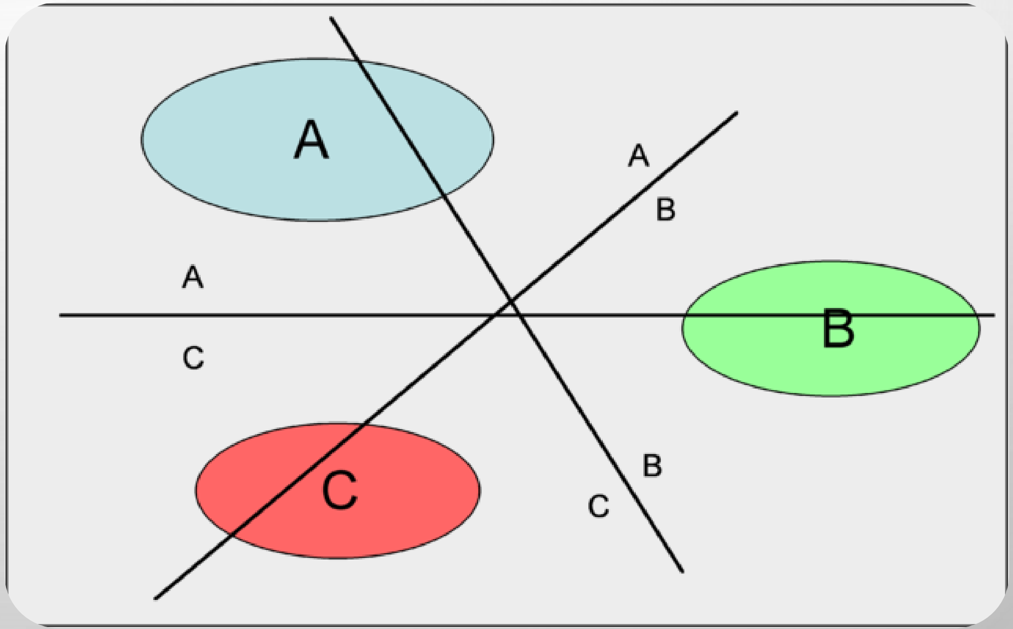
VOTAÇÃO


How do ensemble methods combine base learners into a final learner? Some techniques—e.g. stacking—use separate machine learning algorithms to train an ensemble learner from the base learners. But one common method for consolidating base learner predictions is voting—and more precisely, majority voting.

Majority voting considers each base learner's prediction for a given data instance and outputs a final prediction determined by whatever the majority of learners predict. For instance, in a binary classification problem, majority voting takes predictions from each base classifier for a given data instance and uses the majority prediction as the end prediction. Weighted majority voting is an extension of this technique that gives greater weight to certain learner's predictions over others.[11]

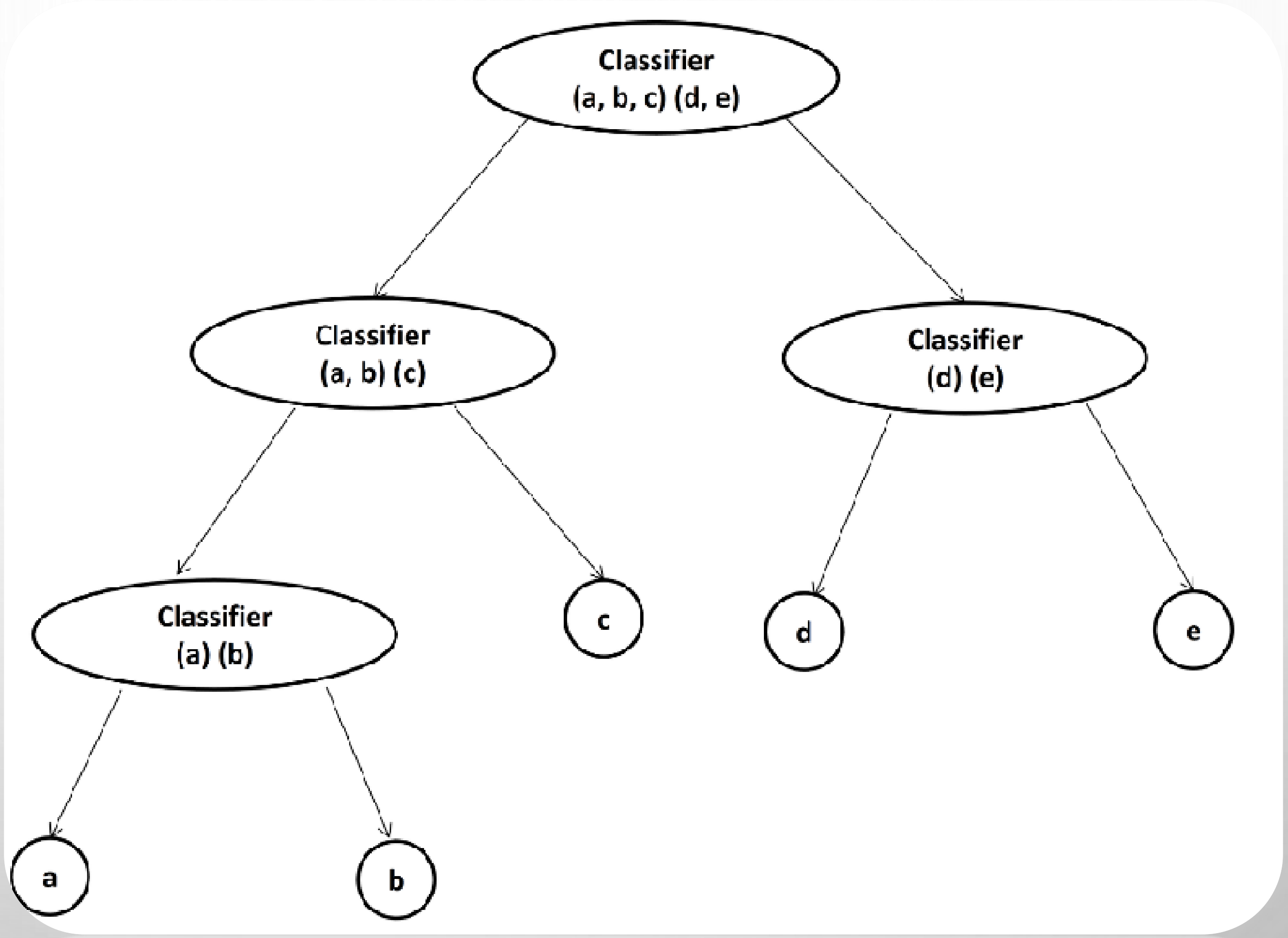


ONE AGAINST ALL



ONE AGAINST ONE

VOTAÇÃO: DIRECTED ACYCLIC GRAPH (DAG)

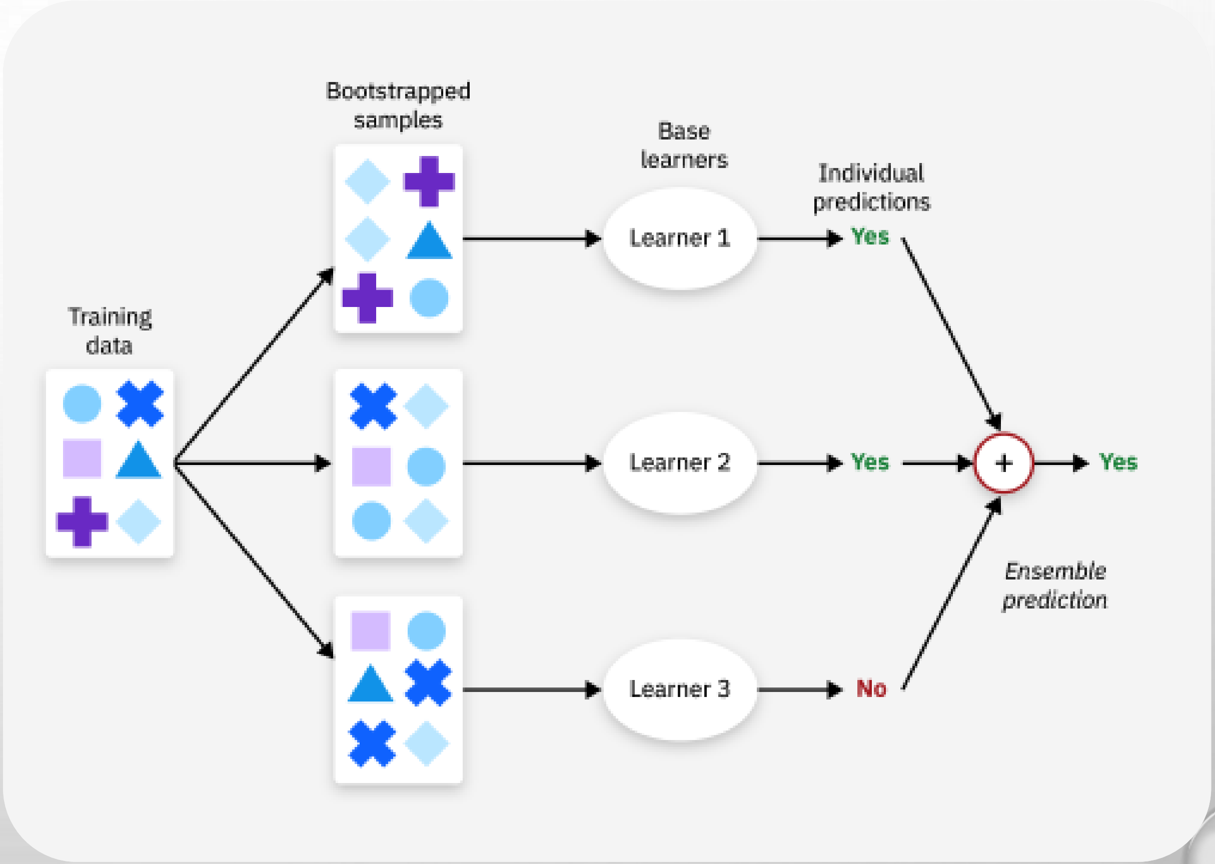


BAGGING

Bagging is a homogenous parallel method sometimes called *bootstrap aggregating*. It uses modified replicates of a given training data set to train multiple base learners with the same training algorithm.[12] Scikit-learn's ensemble module in Python contains functions for implementing bagging, such as BaggingClassifier.

More specifically, bagging uses a technique called bootstrap resampling to derive multiple new datasets from one initial training dataset in order to train multiple base learners. How does this work? Say a training dataset contains *n* training examples. Bootstrap resampling copies *n* data instances from that set into a new subsample dataset, with some initial instances appearing more than once and others excluded entirely. These are bootstrap samples. Repeating this process *x* times produces *x* iterations of the original dataset, each containing *n* samples from the initial set. Each iteration of the initial set is then used to train a separate base learner with the same learning algorithm.[13]

Random forest is an extension of bagging that specifically denotes the use of bagging to construct ensembles of randomized decision trees. This differs from standard decision trees in that the latter samples every feature to identify the best for splitting. By contrast, random forests iteratively sample random subsets of features to create a decision node.[14]

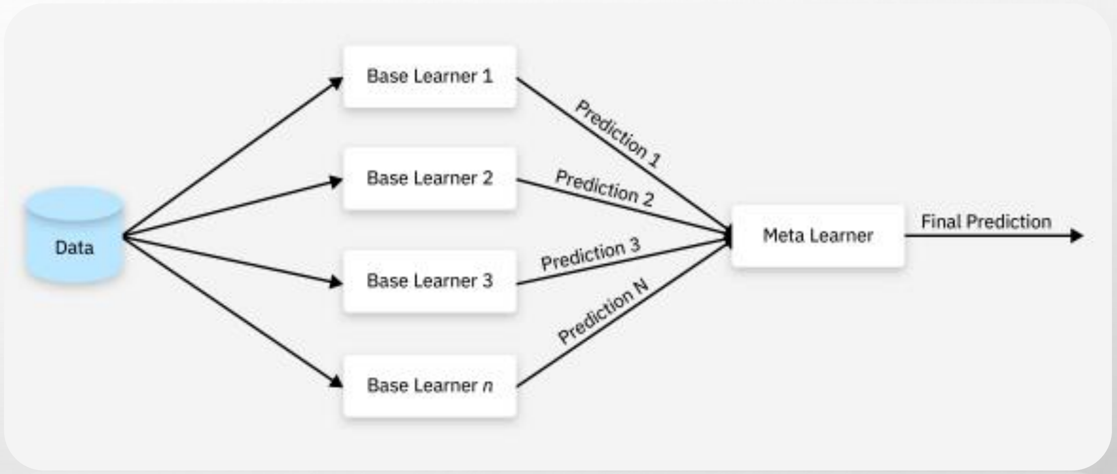


STACKING

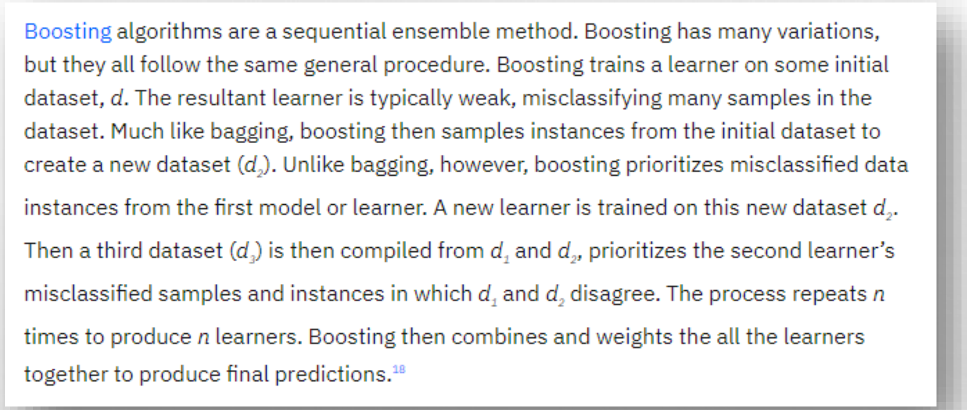
Stacking, or stacked generalization,[15] is a heterogenous parallel method that exemplifies what is known as meta-learning. Meta-learning consists of training a meta-learner from the output of multiple base learners. Stacking specifically trains several base learners from the same dataset using a different training algorithm for each learner. Each base learner makes predictions on an unseen dataset. These first model predictions are then compiled and used to train a final model, being the meta-model.[16]

Note the importance of using a different dataset from that used to train the base learners in order to train the meta-learner. Using the same dataset to train the base learners and the meta-learner can result in overfitting. This can require excluding data instances from the base learner training data to serve as its test set data, which in turn becomes training data for the meta-learner. Literature often recommends techniques such as cross-validation to ensure these datasets do not overlap.[17]

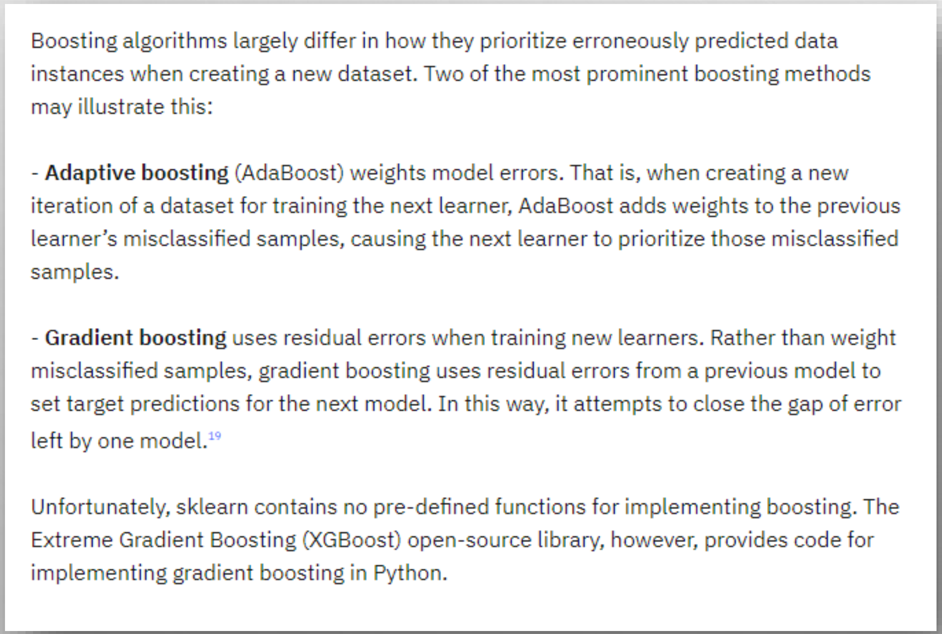
Much as bagging, the sklearn.ensemble module in Python provides various functions for implementing stacking techniques.



BOOSTING



Boosting algorithms are a sequential ensemble method. Boosting has many variations, but they all follow the same general procedure. Boosting trains a learner on some initial dataset, *d*. The resultant learner is typically weak, misclassifying many samples in the dataset. Much like bagging, boosting then samples instances from the initial dataset to create a new dataset (d_2). Unlike bagging, however, boosting prioritizes misclassified data instances from the first model or learner. A new learner is trained on this new dataset d_2. Then a third dataset (d_3) is then compiled from d_1 and d_2, prioritizes the second learner's misclassified samples and instances in which d_1 and d_2 disagree. The process repeats *n* times to produce *n* learners. Boosting then combines and weights the all the learners together to produce final predictions.[18]

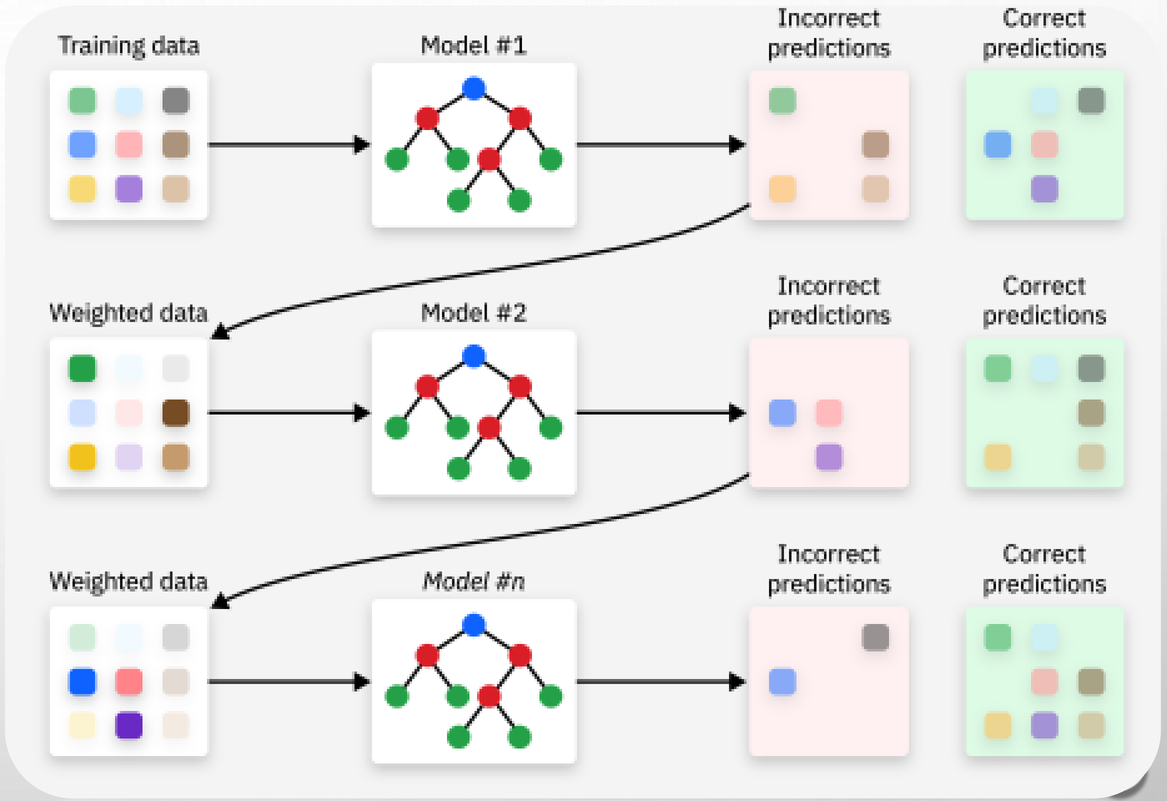


Boosting algorithms largely differ in how they prioritize erroneously predicted data instances when creating a new dataset. Two of the most prominent boosting methods may illustrate this:

- **Adaptive boosting** (AdaBoost) weights model errors. That is, when creating a new iteration of a dataset for training the next learner, AdaBoost adds weights to the previous learner's misclassified samples, causing the next learner to prioritize those misclassified samples.

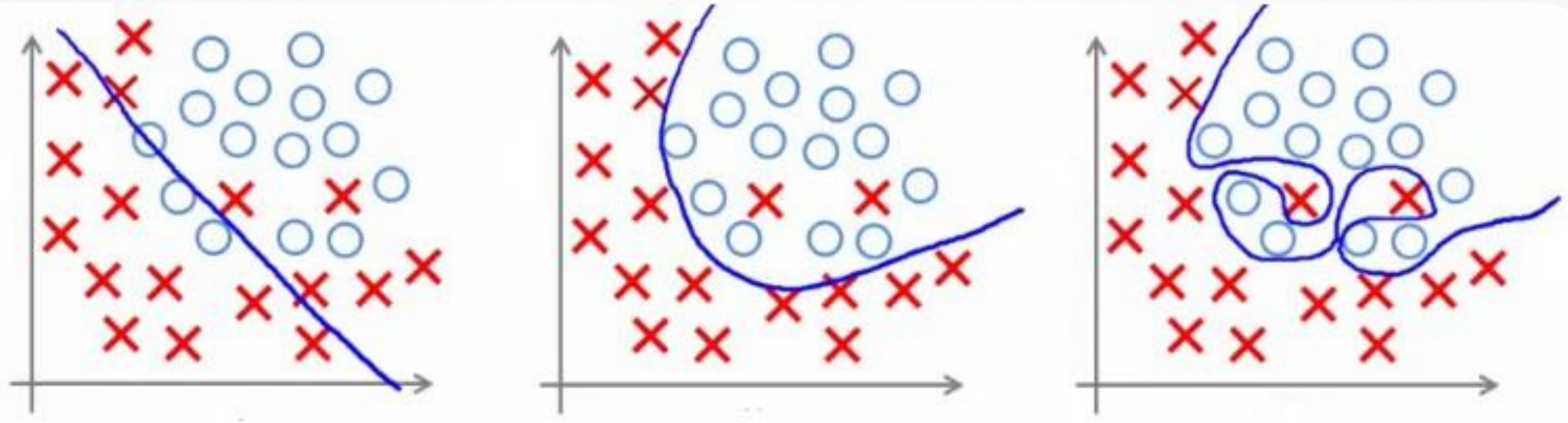
- **Gradient boosting** uses residual errors when training new learners. Rather than weight misclassified samples, gradient boosting uses residual errors from a previous model to set target predictions for the next model. In this way, it attempts to close the gap of error left by one model.[19]

Unfortunately, sklearn contains no pre-defined functions for implementing boosting. The Extreme Gradient Boosting (XGBoost) open-source library, however, provides code for implementing gradient boosting in Python.



EVALUATION

CAPACIDADE E GENERALIZAÇÃO



Under-fitting

(too simple to explain the variance)

Appropriate-fitting

Over-fitting

(forcefitting -- too good to be true)

ESTIMANDO O ERRO DE GENERALIZAÇÃO

LEAVE ONE OUT

- Uma única observação é deixada de fora a cada treinamento. N treinamentos são realizados para calcular a estatística de erro.

SINGLE SPLIT (GRUPO DE CONTROLE)

- Amostra é dividida entre treino e teste, mantendo um percentual das observações como grupo de teste externo ao treinamento.

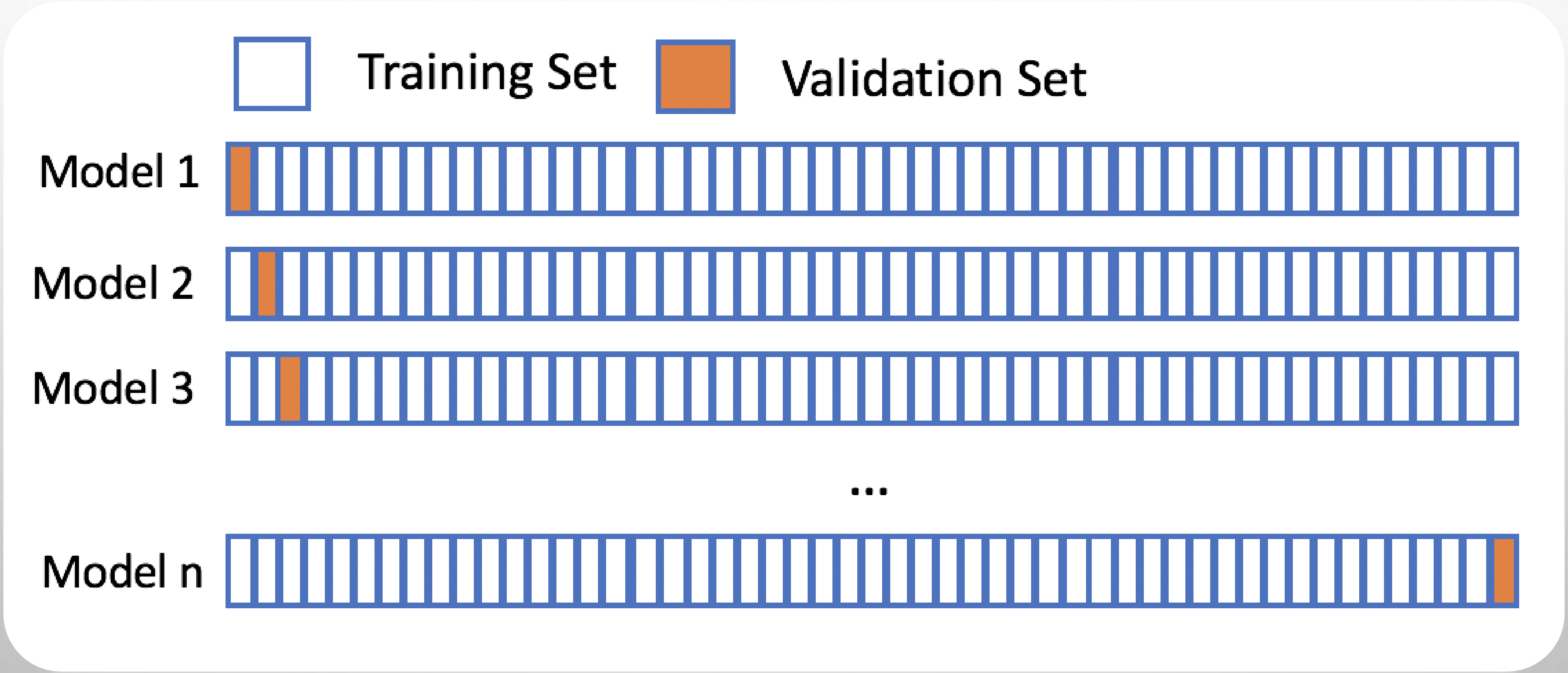
K FOLDS

- Amostra é dividida em K conjuntos. K treinamentos são realizados, mantendo um conjunto como fora-da-amostra.

BOOTSTRAPPING

- O algoritmo itera, amostrando aleatoriamente M observações, para a quantidade Q desejada de treinamentos.

LEAVE ONE OUT



BATALHA DE VIZINHOS II