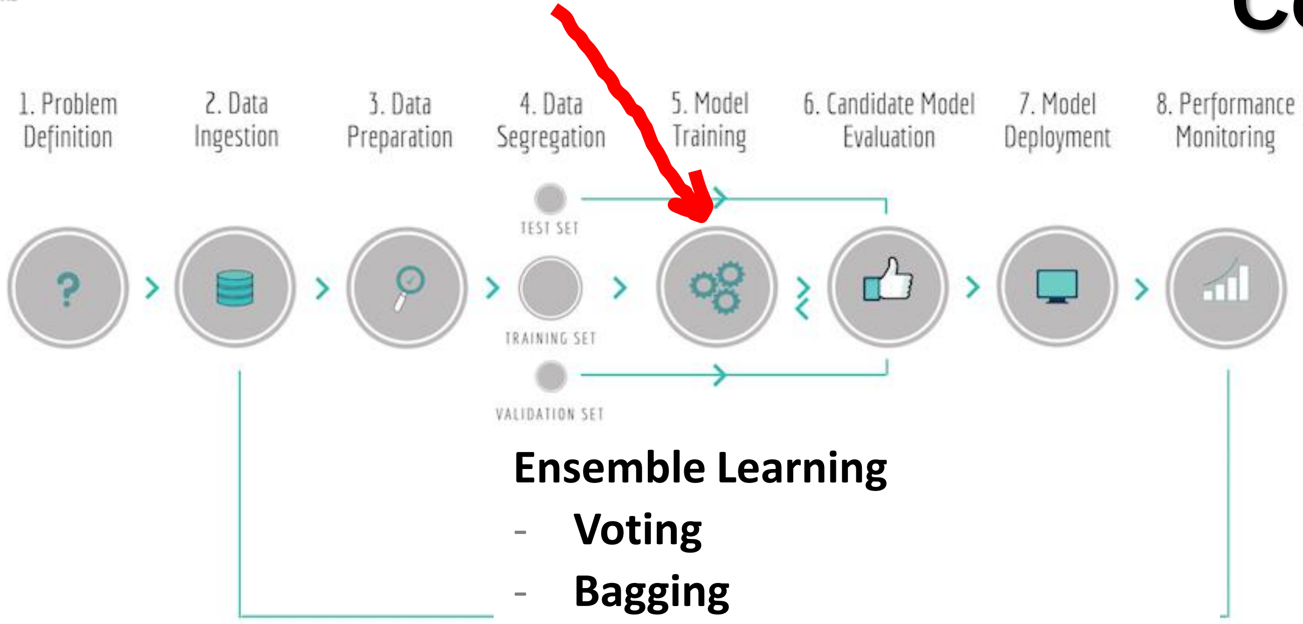
Synthetic Intelligence Lab
Universidade do Minho
Departamento de
Informática

DADOS e APRENDIZAGEM AUTOMÁTICA

Ensemble Learning

MESTRADO (integrado) EM ENGENHARIA INFORMÁTICA

Contents

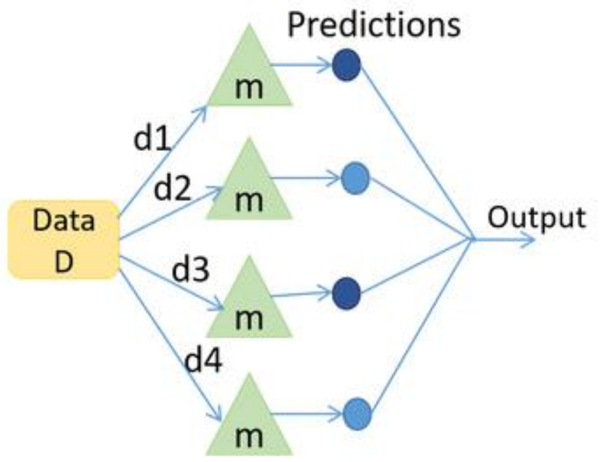


Ensemble Learning

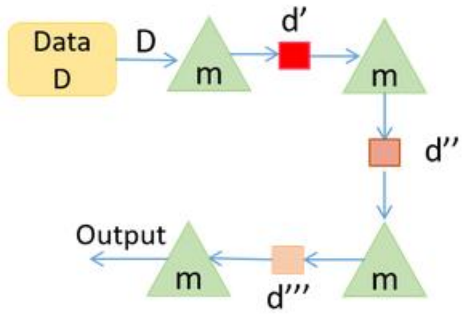
- **Voting**
- **Bagging**
- **Boosting**
- **Stacking**
- **Blending**

Main Ensemble Learning Classes

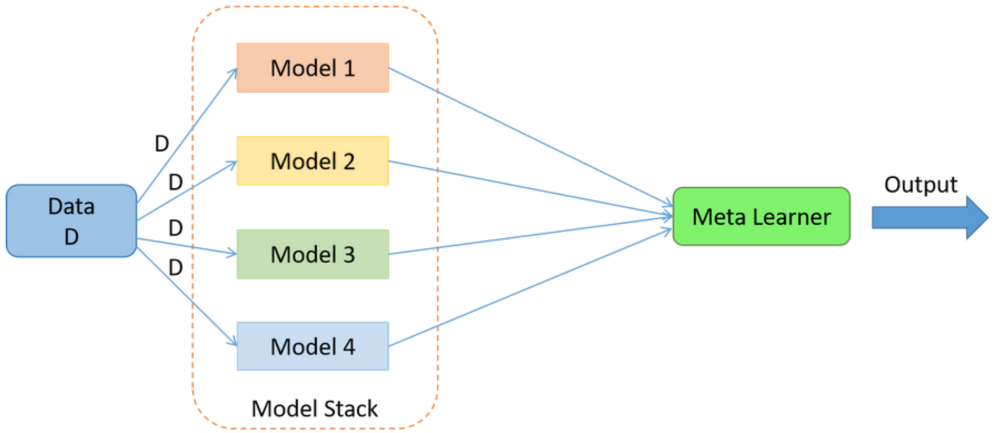
Bagging



Boosting



Stacking



Max Voting (classification)

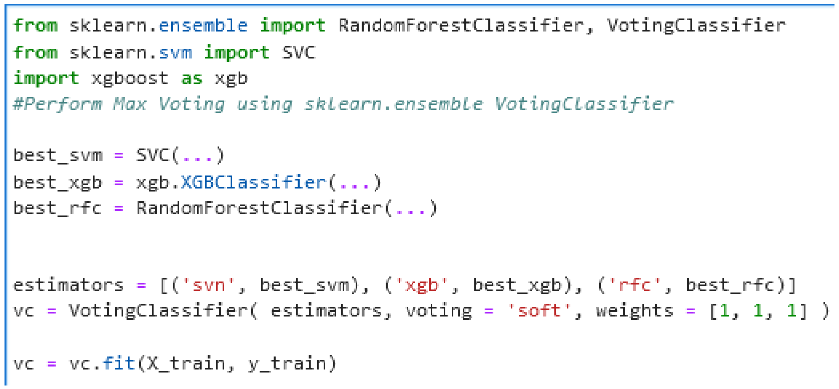
- Each machine learning model makes a vote, the **maximum voted option is the selected**. Simple, yet effective.
- There are two types of max voting:

Hard Voting
Classifier_1 predicts *Class **A***
Classifier_2 predicts *Class **B***
Classifier_3 predicts *Class **B***

2/3 classifier models predict class **B**.
Class B is the **ensemble decision**.

Soft Voting
Classifier_1 predicts *Class **A*** with the probability of 99%
Classifier_2 predicts *Class **A*** with the probability of 49%
Classifier_3 predicts *Class **A*** with the probability of 49%

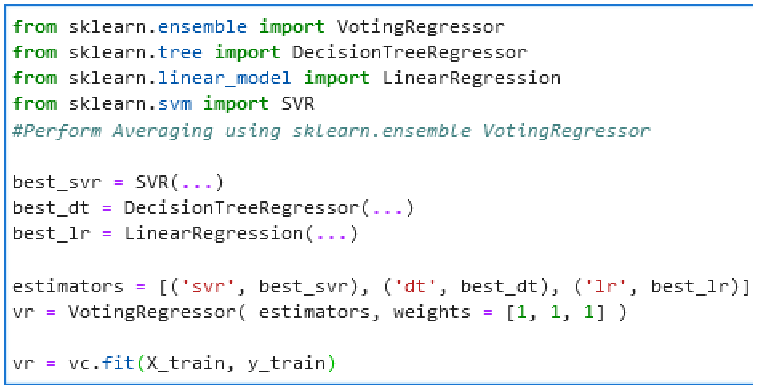
The average probability of belonging to Class A is (99 + 49 + 49) / 3 = 65.67%.
Thus, ***Class A*** is the **ensemble decision**.



```
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.svm import SVC
import xgboost as xgb
#Perform Max Voting using sklearn.ensemble VotingClassifier

best_svm = SVC(...)
best_xgb = xgb.XGBClassifier(...)
best_rfc = RandomForestClassifier(...)


estimators = [('svn', best_svm), ('xgb', best_xgb), ('rfc', best_rfc)]
vc = VotingClassifier( estimators, voting = 'soft', weights = [1, 1, 1] )

vc = vc.fit(X_train, y_train)
```

Averaging and Weighted Average

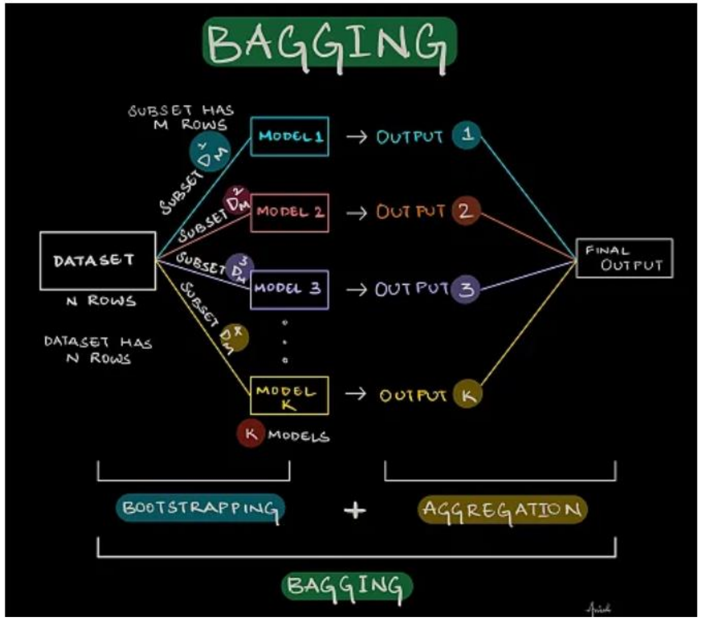
- The **average of the models predictions** is the final prediction that we consider.
- A weighted average ensemble model allows multiple models to contribute to the prediction based on how good the model is. If a model does better on the dataset in general, we will give it a higher weight. This generalization will help reduce bias and improve overall performance.



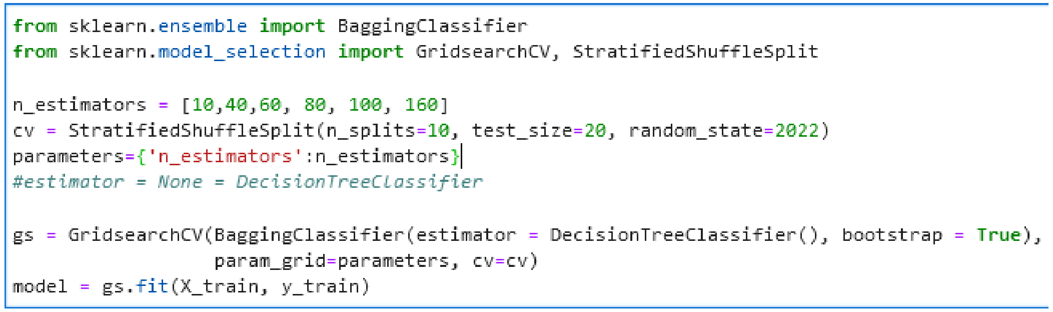
```
from sklearn.ensemble import VotingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
#Perform Averaging using sklearn.ensemble VotingRegressor

best_svr = SVR(...)
best_dt = DecisionTreeRegressor(...)
best_lr = LinearRegression(...)

estimators = [('svr', best_svr), ('dt', best_dt), ('lr', best_lr)]
vr = VotingRegressor( estimators, weights = [1, 1, 1] )

vr = vc.fit(X_train, y_train)
```

Bagging (Bootstrap Aggregation)

- **Bagging** stands for **Bootstrapped Aggregating**, it has two main steps:
  - Use existing train data and make multiple data instances (**bootstrapping**) – here, you can use the same training samples multiple times.
  - Make multiple models from this bootstrapped data and multiple model outputs. **Aggregate** the results of the model and get the final result.



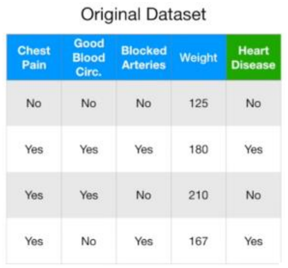
Suppose we have 25 students. We have 50 True or False questions that we will divide in 40(train)+10(test)
1- teach/ train all the students on the same 40 questions and test each one on 10 questions (they all learned the same way)
2 - teach/ train each students on a set of random less than 40 questions(e.g. 30)(bootstrapping) and test them all together on the 10 questions. To get the answer we take the max vote as final result (aggregation).



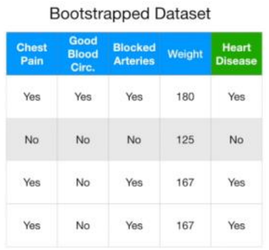
```
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridsearchCV, StratifiedShuffleSplit

n_estimators = [10,40,60, 80, 100, 160]
cv = StratifiedShuffleSplit(n_splits=10, test_size=20, random_state=2022)
parameters={'n_estimators':n_estimators}
#estimator = None = DecisionTreeClassifier

gs = GridsearchCV(BaggingClassifier(estimator = DecisionTreeClassifier(), bootstrap = True),
                  param_grid=parameters, cv=cv)
model = gs.fit(X_train, y_train)
```

Random Forest

Step 1. Create a bootstrapped dataset


Original Dataset

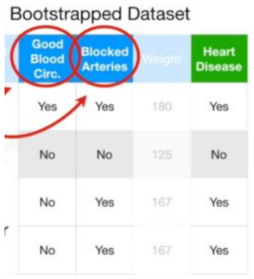
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes


Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Typically, about 1/3 of the original data does not end up in the bootstrapped dataset, This is called the Out-OF-Bag dataset.

Step 2. Build a Decision Tree
The decision tree is built:
1.using the bootstrapped data, and
2.considering a random subset of features (the number of features considered is usually the square root of *n*).


Bootstrapped Dataset

Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	180	Yes
No	No	125	No
No	Yes	167	Yes
No	Yes	167	Yes

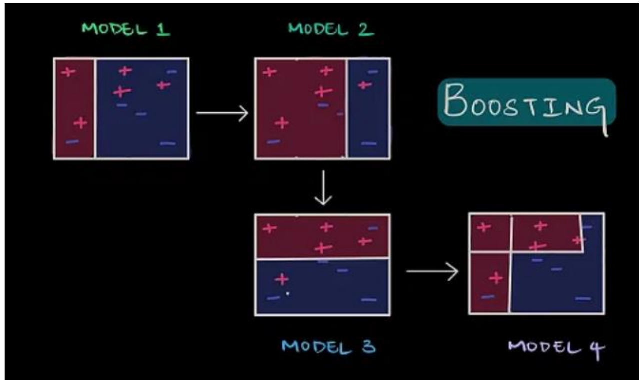
Step 3. Repeat
Step 1 and Step 2 are iterated many times (number of trees) .

Step 4. Estimate the accuracy of a Random Forrest
1.Using Out-Of-Bag samples and aggregation, calculate Out-Of-Bag Error.
2.Iterate Step 2 and Step 3, changing the number of features used (Typically a few settings above and below the square root are tried) minimizing the Error.

The proportion of Out-Of-Bag samples that were incorrectly classified is the “Out-Of-Bag Error”

Boosting

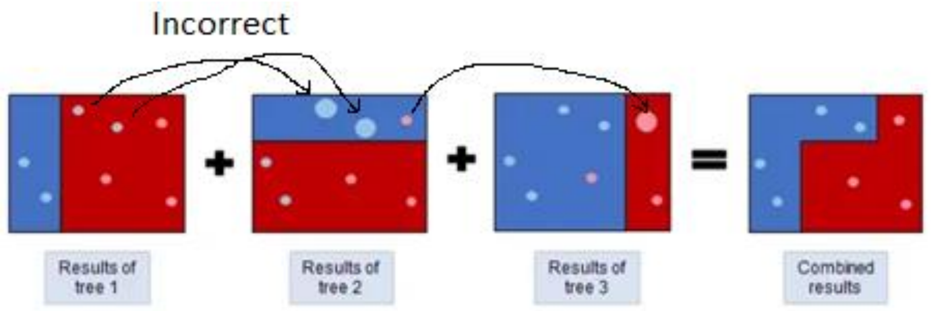
First, a **model is run**, and the results are obtained (Model 1), where specific points are classified correctly, and the rest incorrectly. We **give more weight** to the points that are incorrectly classified and rerun the model. Since there are specific points with a higher weight, the model is more likely to include them. We keep **repeating this process**, and multiple models are created, where **each one corrects the errors of the previous one**.



Suppose we are **teaching a student ML**. He makes mistakes as he starts, and **with each session, we highlight his errors** so that he does not repeat them. After many sessions, it is clear to the student how best to differentiate between right and wrong. This gradual improvement was made by highlighting (boosting) his mistakes.

AdaBoost vs XGBoost

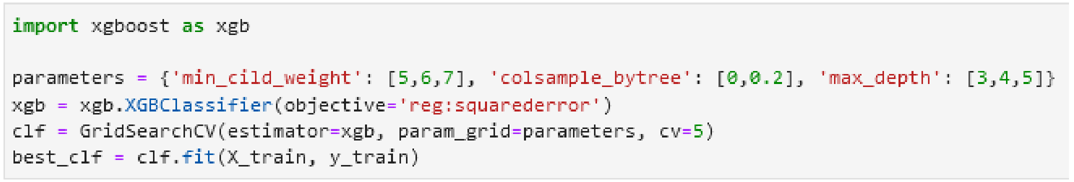
AdaBoost (adaptive boosting):
- Combining decision trees (weak learners)
- Assigning weights to incorrect values
- Sequential tree growing considering past mistakes
- Rebust to overfitting
- Few hyperparameters



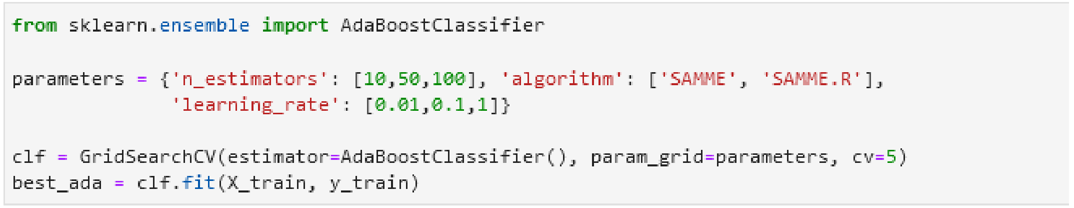
Xgboost – gradient boosting - The idea with gradient boosting is to use gradient descent to optimise the parameters of new estimators as estimators are added in boosting.

Taking the best parts of AdaBoost and random forests, adding aditional features:

- Sequential tree growing
- Minimize loss function using gradient descent
- Parallel processing
- Regularization parameter

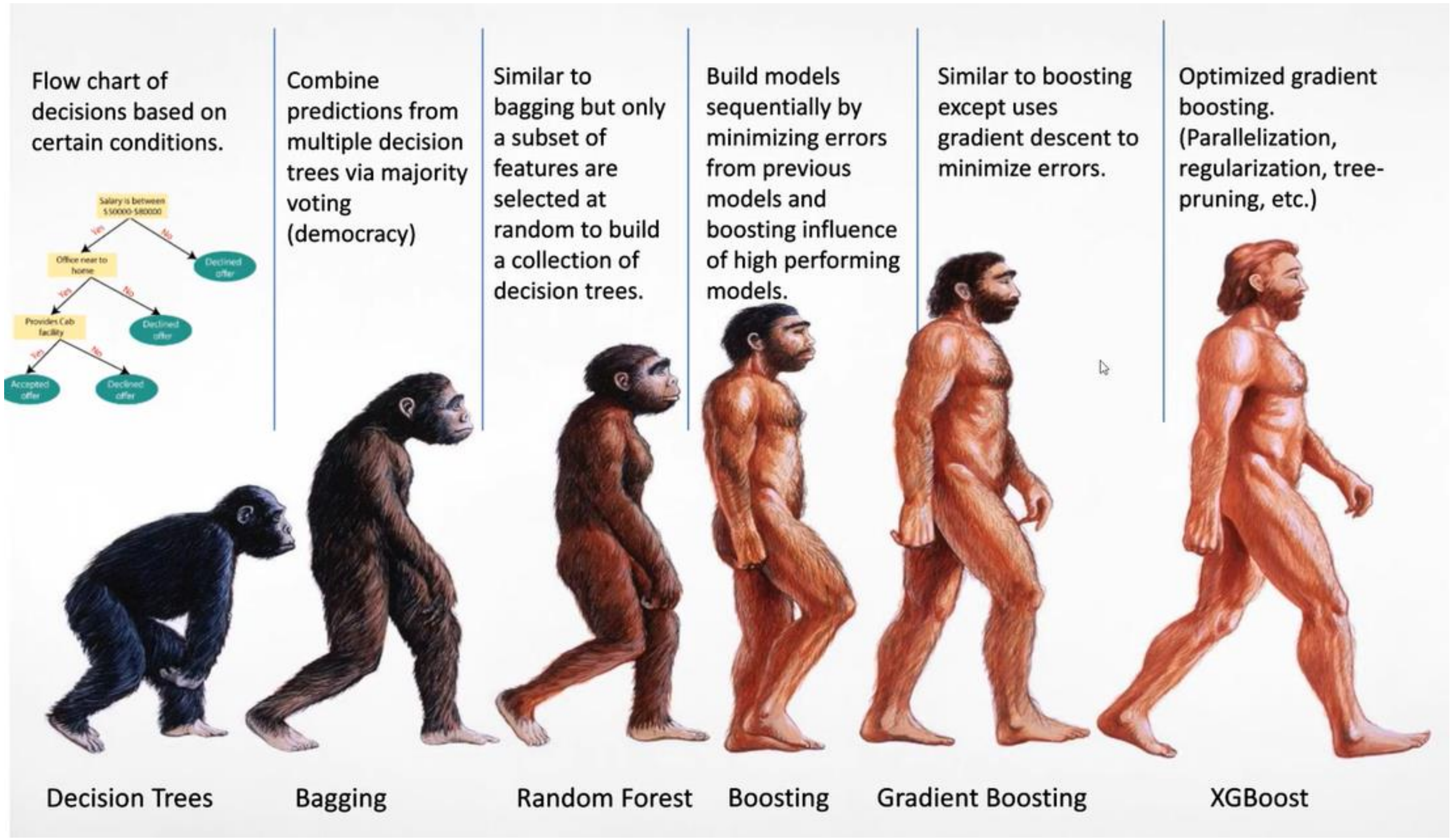


```python
import xgboost as xgb

parameters = {'min_cild_weight': [5,6,7], 'colsample_bytree': [0,0.2], 'max_depth': [3,4,5]}
xgb = xgb.XGBClassifier(objective='reg:squarederror')
clf = GridSearchCV(estimator=xgb, param_grid=parameters, cv=5)
best_clf = clf.fit(X_train, y_train)
```



```python
from sklearn.ensemble import AdaBoostClassifier

parameters = {'n_estimators': [10,50,100], 'algorithm': ['SAMME', 'SAMME.R'],
              'learning_rate': [0.01,0.1,1]}

clf = GridSearchCV(estimator=AdaBoostClassifier(), param_grid=parameters, cv=5)
best_ada = clf.fit(X_train, y_train)
```

From Decision Trees to XGBoost



Decision Trees: Flow chart of decisions based on certain conditions.

Bagging: Combine predictions from multiple decision trees via majority voting (democracy)

Random Forest: Similar to bagging but only a subset of features are selected at random to build a collection of decision trees.

Boosting: Build models sequentially by minimizing errors from previous models and boosting influence of high performing models.

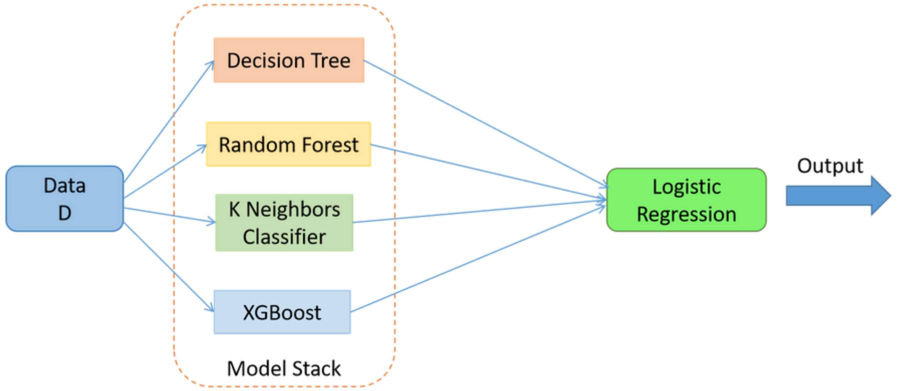
Gradient Boosting: Similar to boosting except uses gradient descent to minimize errors.

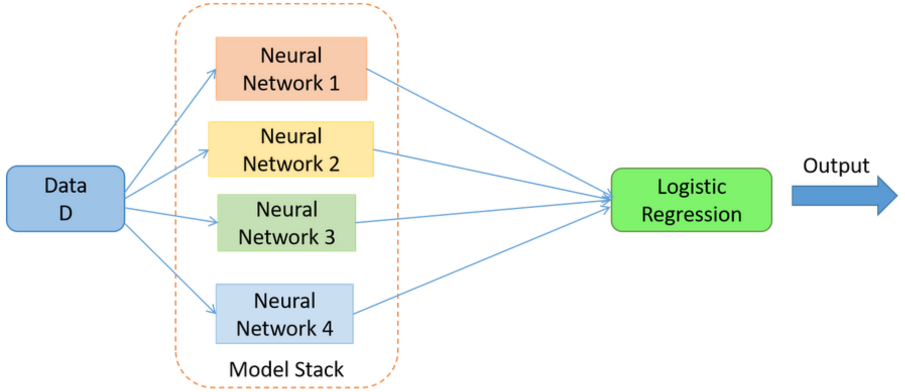
XGBoost: Optimized gradient boosting. (Parallelization, regularization, tree-pruning, etc.)

Retrieved from DigitalSreeni

Stacking

The idea of stacking is that we **perform predictions on the train and test dataset with a few models**. This is done on the train and test dataset. We now do not consider the original train and test data. Instead, we consider the new models outputs on the train data as the base train model. The new test data is the models outputs on the test data



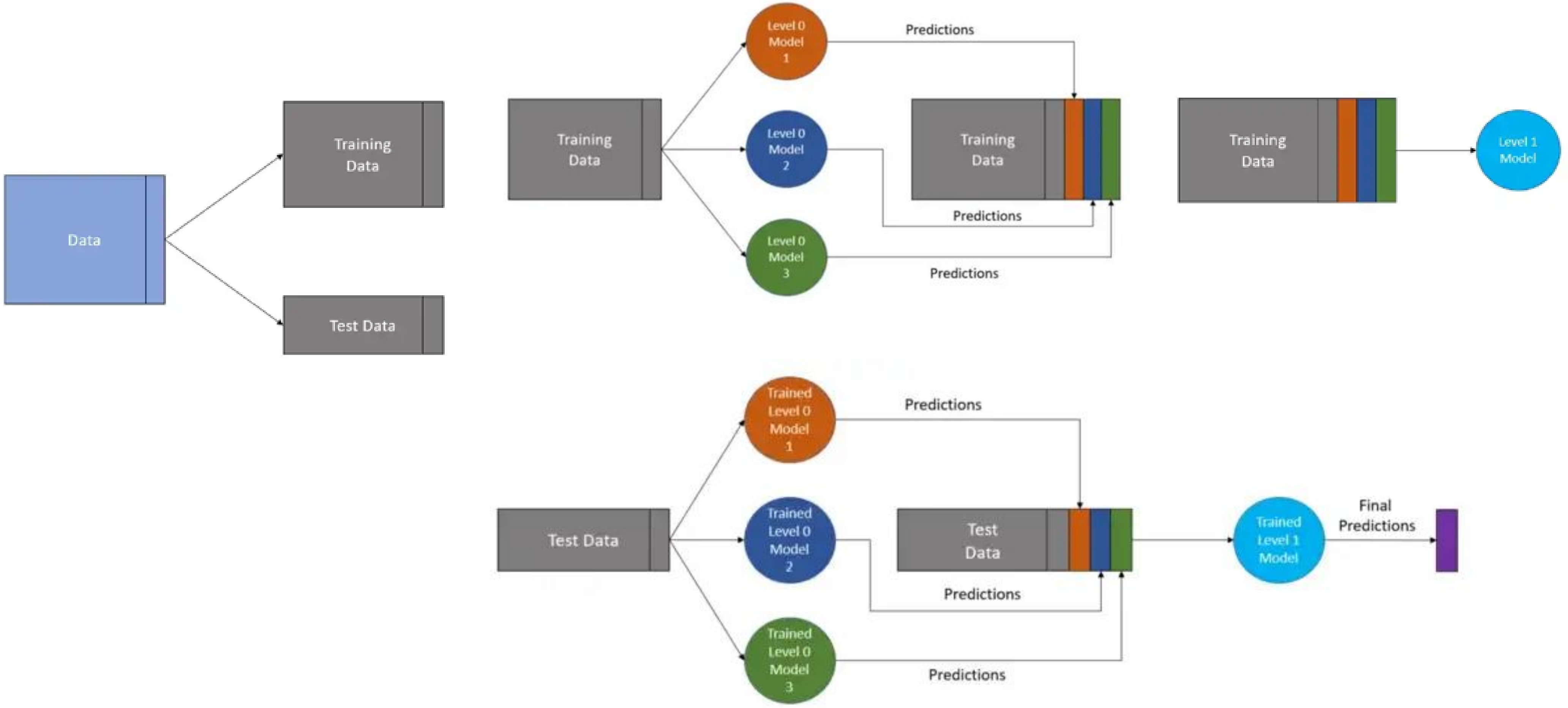


```
estimators = [('rf', RandomForestClassifier(n_estimators=10)),
              ('svm',make_pipeline(StandardScaler(), LinearSVC()))]
#stacking
clf = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())

resultStacking= clf.fit(X_train, y_train).score(X_test, y_test)
```

Universidade do Minho
Departamento de Informática

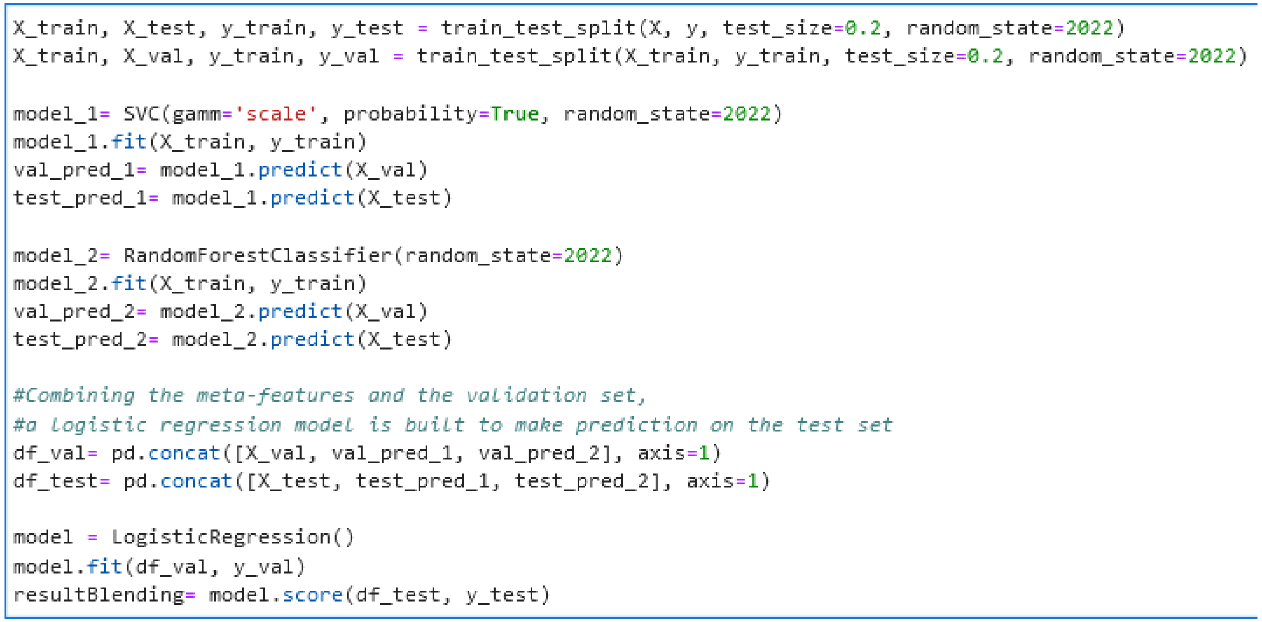
Synthetic Intelligence Lab

Stacking



Blending

You can think of blending as a type of stacking, where the meta-model is trained on predictions made by the base model on the hold-out validation set.



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2022)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=2022)

model_1= SVC(gamm='scale', probability=True, random_state=2022)
model_1.fit(X_train, y_train)
val_pred_1= model_1.predict(X_val)
test_pred_1= model_1.predict(X_test)

model_2= RandomForestClassifier(random_state=2022)
model_2.fit(X_train, y_train)
val_pred_2= model_2.predict(X_val)
test_pred_2= model_2.predict(X_test)

#Combining the meta-features and the validation set,
#a logistic regression model is built to make prediction on the test set
df_val= pd.concat([X_val, val_pred_1, val_pred_2], axis=1)
df_test= pd.concat([X_test, test_pred_1, test_pred_2], axis=1)

model = LogisticRegression()
model.fit(df_val, y_val)
resultBlending= model.score(df_test, y_test)
```

Ensemble Learning

Ensemble models work **by combining multiple base learners** into a single strong learner. This helps by decreasing the bias, variance, or improving predictions.

There are two groups of ensemble models :

- **Parallel ensemble models** – The logic employed is to leverage the independence between the base learners. Thus, the mistakes in predictions made by one model are different from those from another independent model. This lets the ensemble model average out the errors. (e.g. Random Forest with independent Decision Trees)
- **Sequential ensemble models** – The logic employed is to leverage the dependence between the base learners. Thus, the mistakes made by the first model are sequentially corrected by the second model and so on. This helps get the most accurate ensemble possible. (e.g. ADABoost is sequencial)

? How about **Stacking** and **Blending** ?

OBS: Combining similar or good learners does not make much of a difference. Instead, combining a variety of **heterogeneous/weak machine learning techniques** helps improve generalization.

References

- R. Bonab, Hamed; Can, Fazli (2019). *Less Is More: A Comprehensive Framework for the Number of Components of Ensemble Classifiers.* TNNLS. USA: IEEE. arXiv:1709.02925
- Kuncheva, L. and Whitaker, C., *Measures of diversity in classifier ensembles*, *Machine Learning*, 51, pp. 181-207, 2003
- Opitz, D.; Maclin, R. (1999). "*Popular ensemble methods: An empirical study*". Journal of Artificial Intelligence Research. **11**: 169–198. doi:10.1613/jair.614