Ensemble Learning:
About Ensemble Learning

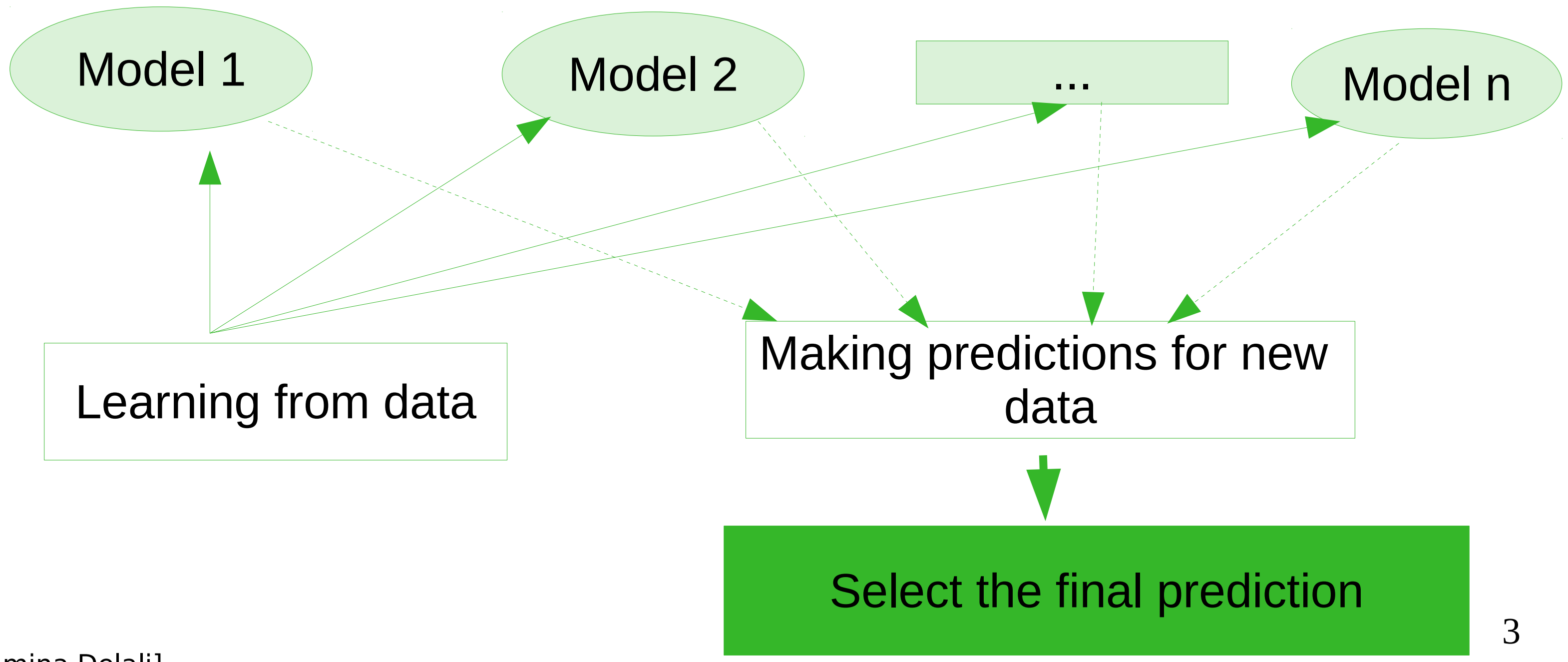
AAA-Python Edition

Plan

- 1- Ensemble Learning
- 2- Bagging & Pasting
- 3- Features sampling
- 4- Boosting : Adaptive Boosting
- 5- Boosting: Gradient boosting
- 6- Stacking or Blending

Concept

- In **Ensemble Learning**, we **combine** several models to build a **better** model. The algorithm used in **Ensemble learning** is called: an **Ensemble Method**.
- We can combine **classifiers** or **regressors.**
- The models can be **all the same type,** or **different.**



[By Amina Delali]

Voting

- Its about: how to select the **final prediction**.

- In Classification

  - **Hard Voting:** For each sample, a classifier will make a prediction : a class for that sample
    - Select the **most** predicted class by all the classifiers, for that sample .

  - **Soft Voting:** available when the classifiers can predict class probabilities.
    - Select the class with the **highest averaged** probability

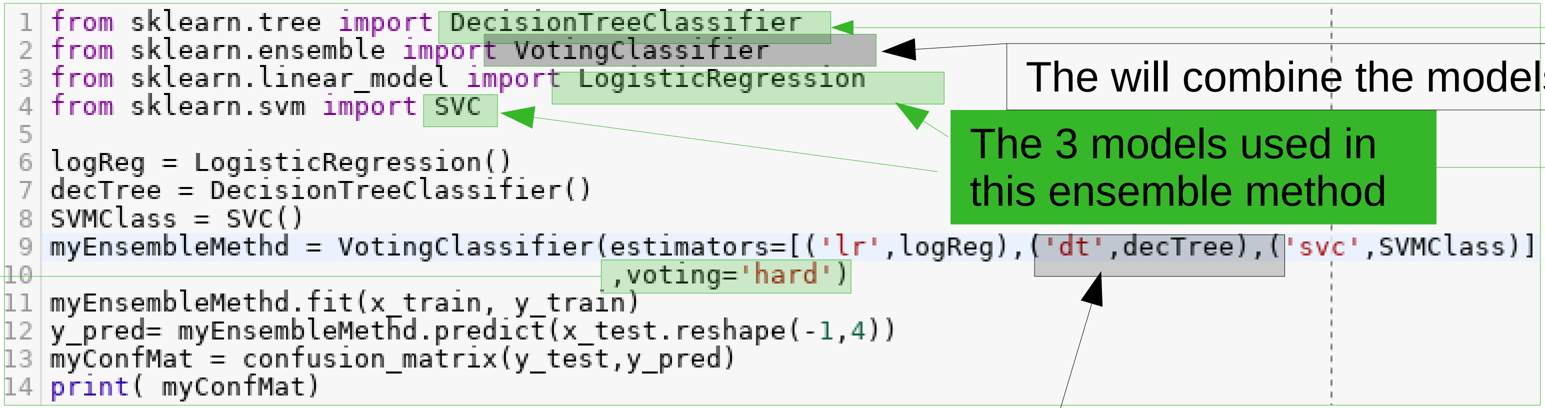
- In Regression

  - The **average** of the predicted values.

Ensemble Methods

- The ensemble methods can vary by:
  - **Varying** or not the types of **the models**: use the same or different models.
  - Select the **same sample** only **once** or **several times** in **the same model**.
  - Whether or not the model use all the **features** or only **a subset** of features.
  - The models learn in **parallel** or **sequentially**.
  - The type of **mechanism** used to make a **prediction**.

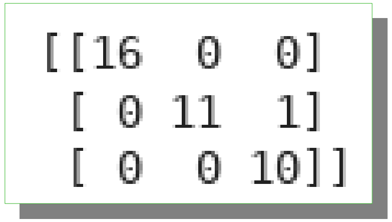
Example



```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

logReg = LogisticRegression()
decTree = DecisionTreeClassifier()
SVMClass = SVC()
myEnsembleMethd = VotingClassifier(estimators=[('lr',logReg),('dt',decTree),('svc',SVMClass)]
,voting='hard')
myEnsembleMethd.fit(x_train, y_train)
y_pred= myEnsembleMethd.predict(x_test.reshape(-1,4))
myConfMat = confusion_matrix(y_test,y_pred)
print( myConfMat)
```

The will combine the models

The 3 models used in this ensemble method



```
[[16  0  0]
 [ 0 11  1]
 [ 0  0 10]]
```

Hard voting was used to determine these classes

A string representing the type of the model: ‘dt’ for : decision tree

[By Amina Delali]

Definition

- **Bagging (**or **Bootstrap aggregating)** and **pasting** are both ensemble methods that combine **same type** of models. They both train the models on **different** random sub sets. All the models run in **parallel.** They use the **voting mechanism** for the final prediction
- Both **Bagging** and **Basting** apply **random sampling :**
    - The training set for each model is a subset of the original data randomly selected.
    - The same sample can be found in different models (different subsets).
    -
- The difference is:
    - **Bagging :** random sampling with **replacement** <==>
        - One sample can be found **several times** in the same model (same subset).
    - **Pasting:** random sampling without **replacement** <==>
        - One sample can be found **only once** in the same model (same subset).

2-Bagging & Pasting

Bagging example using scikit-learn

The data

```
# load the data
myIris = load_iris()
X = myIris.data
y = myIris.target
x_train,x_test,y_train,y_test= train_test_split(X, y, test_size=0.25)
```

```
from sklearn.ensemble import BaggingClassifier
from sklearn.svm import SVC
theModel2 = SVC()
myBagClass2 = BaggingClassifier(
theModel2, n_estimators=300,max_samples=90, bootstrap=True, n_jobs=-1)
myBagClass2.fit(x_train, y_train)
y_pred = myBagClass2.predict(x_test)
```

Bootstrap =True ==> Ensemble method : **bagging**
TheModel2 = SVC == > all the models are support vector machine classifiers
max_samples=90 ==> the size of the subsets (bags) == 90 sample
n_jobs=-1 ==> use all the available cores (to compute in parallel)
n_estimators= 300 ==> use 300 SVC

```
myConfMat = confusion_matrix(y_test,y_pred)
print( myConfMat)
myBagClass.score(x_test,y_test)
```

```
[[11  0  0]
 [ 0  8  3]
 [ 0  0 16]]
```

[By Amina Delali]

Pasting example using sckit-learn

```python
from sklearn.svm import SVC

myBagClass2P = BaggingClassifier(
theModel2, n_estimators=300,max_samples=90, bootstrap=False, n_jobs=-1)
myBagClass2P.fit(x_train, y_train)
y_pred = myBagClass2P.predict(x_test)
```

Bootstrap =False ==> Ensemble method : **pasting**
The remaining parameter are the same as the previous ones

```python
myConfMat2P = confusion_matrix(y_test,y_pred)
print( myConfMat2P)
```

```
[[10  0  0]
 [ 0 14  1]
 [ 0  0 13]]
```

```python
from sklearn.linear_model import LogisticRegression

theModel = LogisticRegression()
```

The model used is a **LogisticRegression classifier**

```
[[10  0  0]
 [ 0 13  2]
 [ 0  1 12]]
```

3- Features sampling

Definition

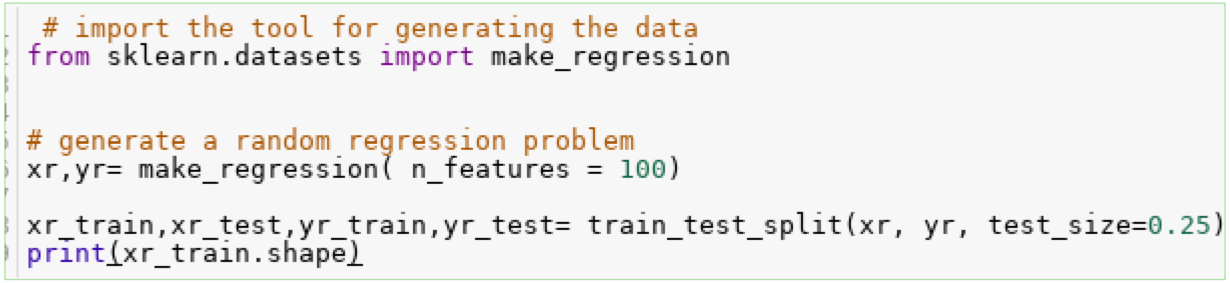
- All the following methods use **features sampling:** each **model** will be **trained** in **a random subset** of features.

- Sampling features can be **with** or **without replacement.**

- **Random patches method**
  - Sampling both “training instances” and “features”

- **Random subspaces method**
  - Keeping all “training” instances but “sampling” features

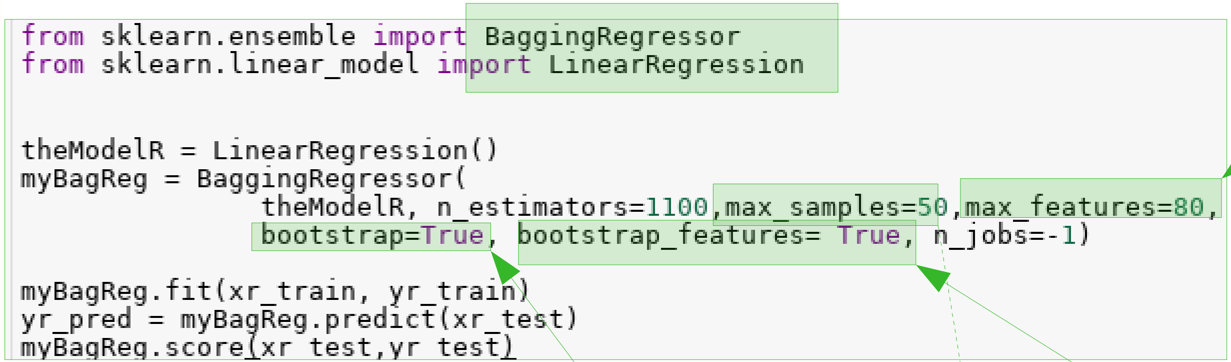
Random Patches method



```
 # import the tool for generating the data
from sklearn.datasets import make_regression


# generate a random regression problem
xr,yr= make_regression( n_features = 100)

xr_train,xr_test,yr_train,yr_test= train_test_split(xr, yr, test_size=0.25)
print(xr_train.shape)
```

The data: 75 samples (instance), with 100 features

```
(75, 100)
```



```
from sklearn.ensemble import BaggingRegressor
from sklearn.linear_model import LinearRegression


theModelR = LinearRegression()
myBagReg = BaggingRegressor(
               theModelR, n_estimators=1100,max_samples=50,max_features=80,
               bootstrap=True, bootstrap_features= True, n_jobs=-1)

myBagReg.fit(xr_train, yr_train)
yr_pred = myBagReg.predict(xr_test)
myBagReg.score(xr_test,yr_test)
```

```
0.5057404688880153
```

Number of selected features = 80 <100 features==> **features sampling**

Bagging

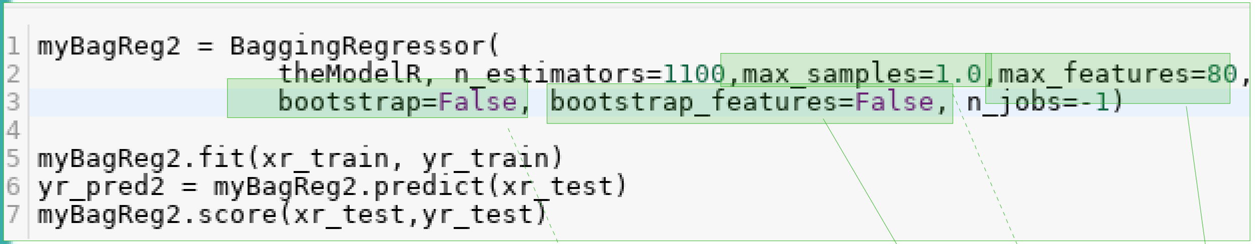
Number of selected samples = 50 < 75 ==> **instances sampling**

Sampling features with replacement

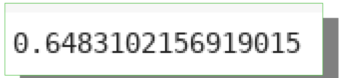
[By Amina Delali]

3- Features sampling

Random Subspaces method



```
myBagReg2 = BaggingRegressor(
                theModelR, n_estimators=1100,max_samples=1.0,max_features=80,
                bootstrap=False, bootstrap_features=False, n_jobs=-1)

myBagReg2.fit(xr_train, yr_train)
yr_pred2 = myBagReg2.predict(xr_test)
myBagReg2.score(xr_test,yr_test)
```



```
0.6483102156919015
```

Sampling features :
80 < 100

Features sampling without replacement

Pasting

Since all samples are used without replacement ==> the instances are **not sampled** ==> all the training data is used (75 samples)

Since max_samples = 1.0 (a float value)==> max_samples =100% of the training data (100% *75 = 75)

[By Amina Delali]

4- Boosting : Adaptive Boosting

Definition

- Boosting : Ensemble method, that combines several **weak** learners into a **stronger** learner.
- This is done by training the models **sequentially ==>** each **model correct (boost)** its **predecessor.**
- Uses the **same models** on the **same data** each time.

- The most known boosting methods are**: Adaptive Boosting** and **Gradient Boosting**.

  - **Adaptive Boosting:** each new predictor focus on the training samples that its **predecessor underfitted** (for example: misclassified in a classification problem) by **modifying** the **instances weight .**
  - **Gradient Boosting:** the new predictor tries to **fit to the residual errors** made by the **previous predictor**.

4- Boosting :
Adaptive Boosting

AdaBoost: training

- Weighting samples ==> each sample value will be multiplied by its weight.
- AdaBoost is Applicable in binary classification.

- The steps of the algorithm are as follow:
  - initialize the samples weight $\mathbf{w^i}$ (for the first predictor) by 1/m. m is the number of the training samples.
  - for each predictor j compute:
    - the weighted error rate : $\mathbf{r_j=\sum w^i}$ **(whre the prediction is wrong)** $\mathbf{/ \sum w^i}$
    - compute the j predictor's weight: $\mathbf{\alpha_j=\eta\log(1-r_{j)}/ r_j)}$. $\boldsymbol{\eta}$ is the learning rate parameter.
    - Compute the new weights (to be used by the following new predictor **j+1**) : $\mathbf{w^i=w^i, if, y_{true}(i)=y_{pred}(i)}$
      $\mathbf{w^i=w^i\exp(\alpha_j), if, y_{true}(i)\neq y_{pred}(i)}$
    - Normalize the new weights $\mathbf{w^i}$ by: $\mathbf{w^i=1/\sum w^i}$
    - The process is repeated until the perfect predictor is found, or the maximum number of predictors is reached.

AdaBoost: Predicting

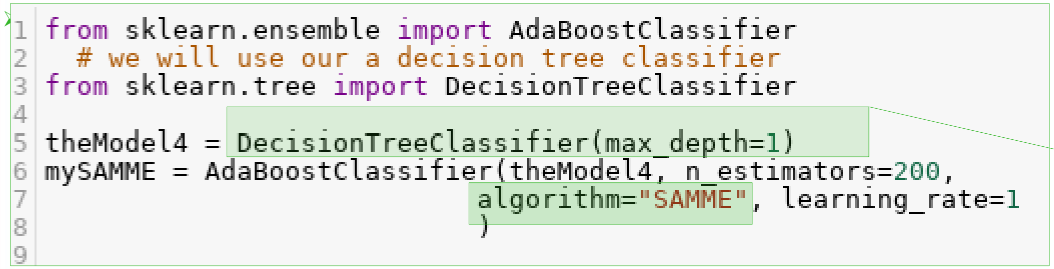
- To make a prediction:
  - make a prediction with each predictor **j** from the resulting **N** predictors.
  - attribute a weight to each prediction by the predictor's **j** weight $\boldsymbol{\alpha_j}$
  - for each sample **x** select the class **k** that receives the majority of weighted votes: for each predicted class **k** sum up the corresponding $\boldsymbol{\alpha_j}$ weights, then select the class **k** with the biggest sum.

AdaBoost: SAMME

- SAMME : **S**tagewise **A**dditive **M**odeling using a **M**ulti-class **E**xponential loss function
- Enhanced version of AdaBoost, applicable in multiclass classification.
- Same steps as AdaBoost, just the **α** weight is computed differently: $\boldsymbol{\alpha_j = \eta * (\log[(1-rj)/rj] + \log(K-1))}$. **K** is the number of classes.

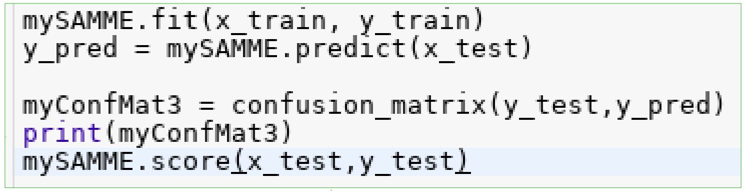
4- Boosting : Adaptive Boosting

Example



```
from sklearn.ensemble import AdaBoostClassifier
  # we will use our a decision tree classifier
from sklearn.tree import DecisionTreeClassifier

theModel4 = DecisionTreeClassifier(max_depth=1)
mySAMME = AdaBoostClassifier(theModel4, n_estimators=200,
                        algorithm="SAMME", learning_rate=1
                        )
```

A weak classifier : a decision tree with 1 level: the **2** leafs, the split of the root node. This Tree is called a : **Decision Stump**



```
mySAMME.fit(x_train, y_train)
y_pred = mySAMME.predict(x_test)

myConfMat3 = confusion_matrix(y_test,y_pred)
print(myConfMat3)
mySAMME.score(x_test,y_test)
```

```
[[12  0  0]
 [ 0 11  2]
 [ 0  1 12]]
```

```
0.9210526315789473
```

In sckit-learn to apply the adaptive boosting to regression, the weights are adjusted according to the error of the predictions

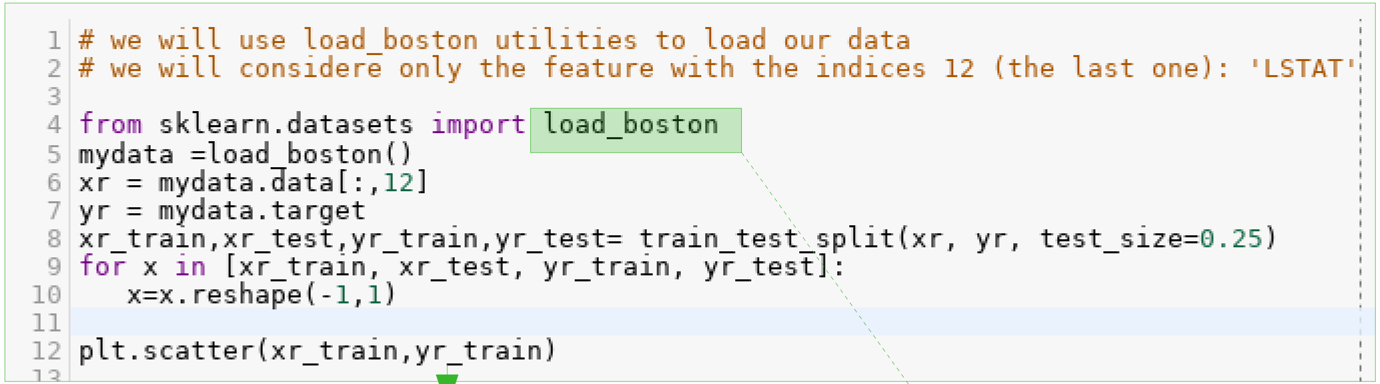
[By Amina Delali]

The concept

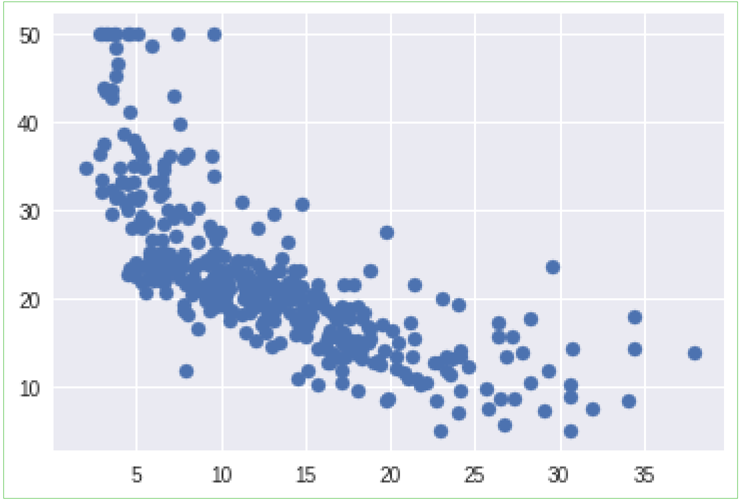
1. The predictor will be first trained on a set of data: x,y
2. The residual errors are computed from its prediction: $r = y - y_{pred}$
3. A new predictor will be trained with the new set of data: x,r
4. The residual errors are computed again as follow: $r2 = r - r_{pred}$
5. The steps 3 and 4 are repeated until: you predict using all the predefined number of predictors, or you determine the optimal consecutive predictors (the least generated error) and you select those predictors as your final model. Or, you continue adding predictors until the errors will not diminish
6. The final prediction will be the sum of all the predictions.

- Scikit learn implements gradient tree boosting: the models used are decision trees.

Example: the data



```
# we will use load_boston utilities to load our data
# we will considere only the feature with the indices 12 (the last one): 'LSTAT'

from sklearn.datasets import load_boston
mydata =load_boston()
xr = mydata.data[:,12]
yr = mydata.target
xr_train,xr_test,yr_train,yr_test= train_test_split(xr, yr, test_size=0.25)
for x in [xr_train, xr_test, yr_train, yr_test]:
    x=x.reshape(-1,1)

plt.scatter(xr_train,yr_train)
```



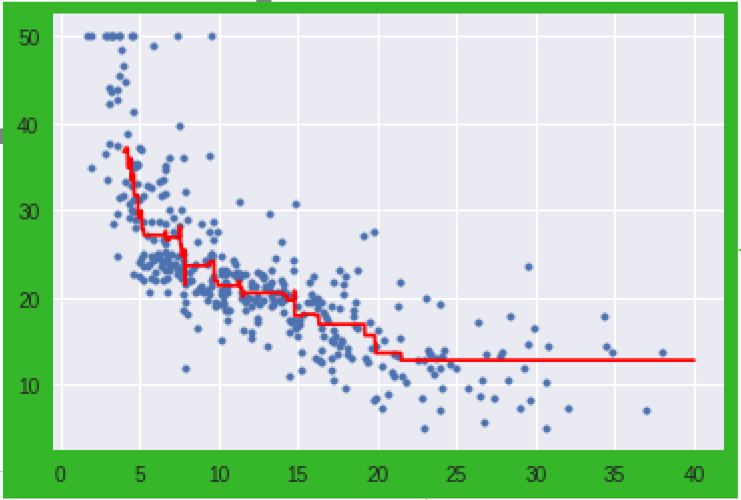
- Boston House Prices dataset.
- The chosen feature represents the: % lower status of the population

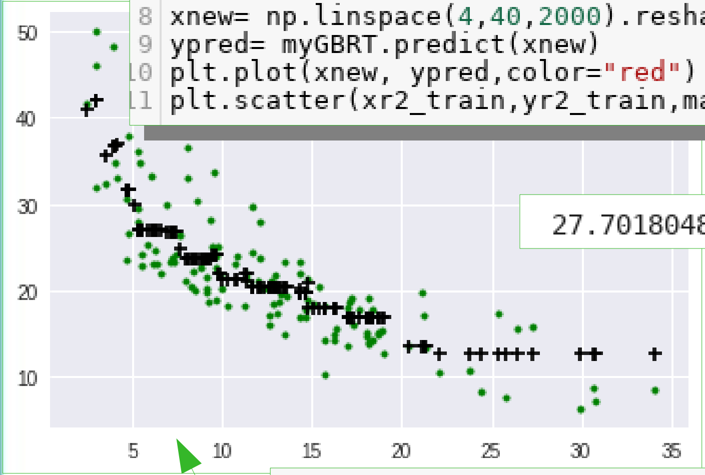
5- Boosting:
Gradient boosting

Example: using a GBRT

- **GBRT** for **G**radient **B**oosted **R**egression **T**rees

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split

myGBRT = GradientBoostingRegressor(max_depth=2, n_estimators=45)
myGBRT.fit(xr2_train, yr2_train)

# xnew to draw the model
xnew= np.linspace(4,40,2000).reshape(-1,1)
ypred= myGBRT.predict(xnew)
plt.plot(xnew, ypred,color="red")
plt.scatter(xr2_train,yr2_train,marker=".")
```





```
27.701804841167945
```

```
# testing the model
ypred= myGBRT.predict(xr2_test)

#plot the true values: in green and the predicted values: in black
plt.scatter(xr2_test,yr2_test,color="green",marker=".")
plt.scatter(xr2_test,ypred,color="black",marker="+")


from sklearn.metrics import mean_squared_error as MSE
print (MSE(yr2_test,ypred))
```

[By Amina Delali]

Concept

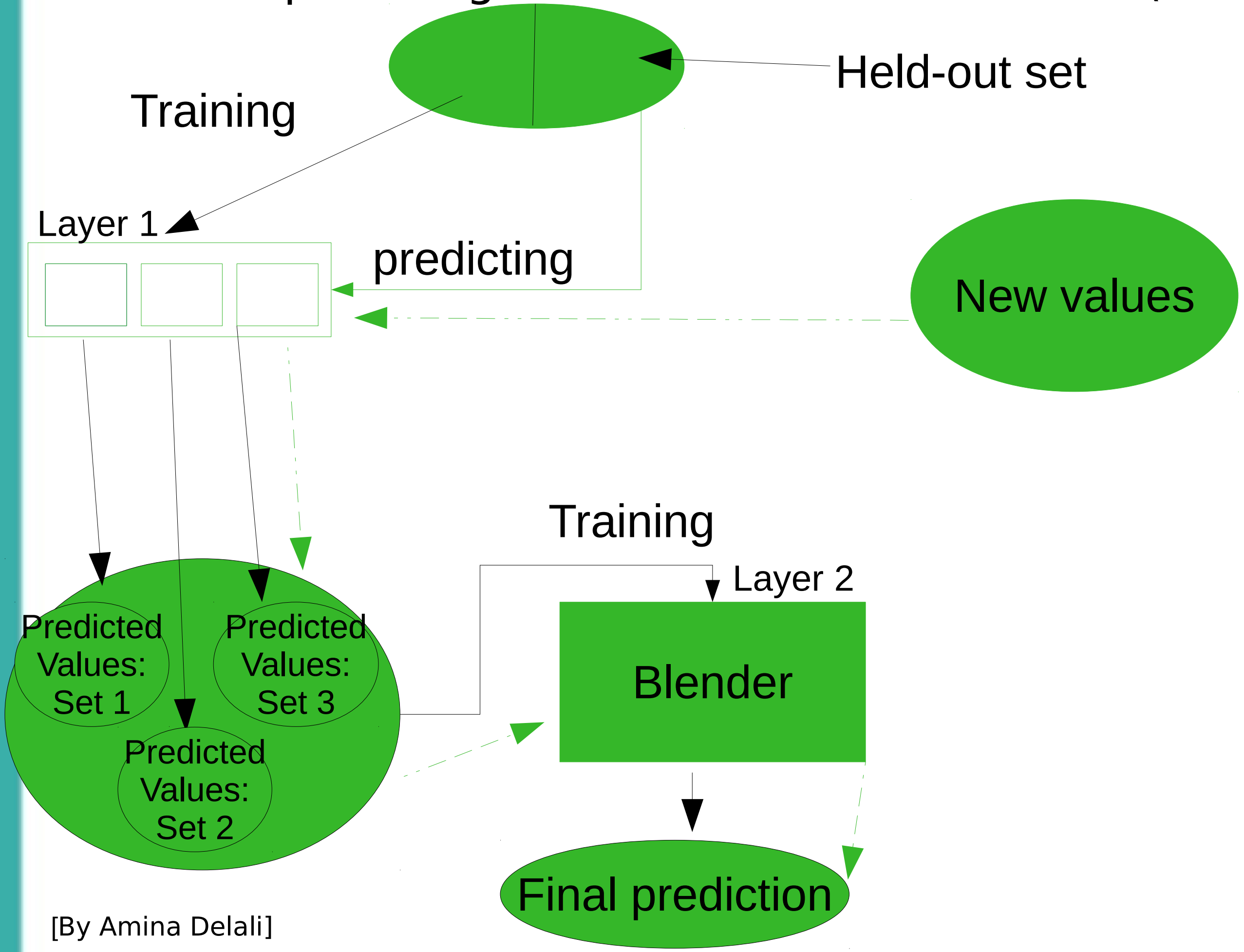
- The idea here, is to **train** a model to learn how to **aggregate** the ensemble models **predictions.**

- The method is composed of:
  - Learner models : that will fit to the data, and make the predictions.
  - Blender: the final model or meta learner, that will make the final prediction.

- There are different methods to train the blender:
  - Hold-out set: Blending
  - Out-of-fold: Stacking

Hold-out set: principle

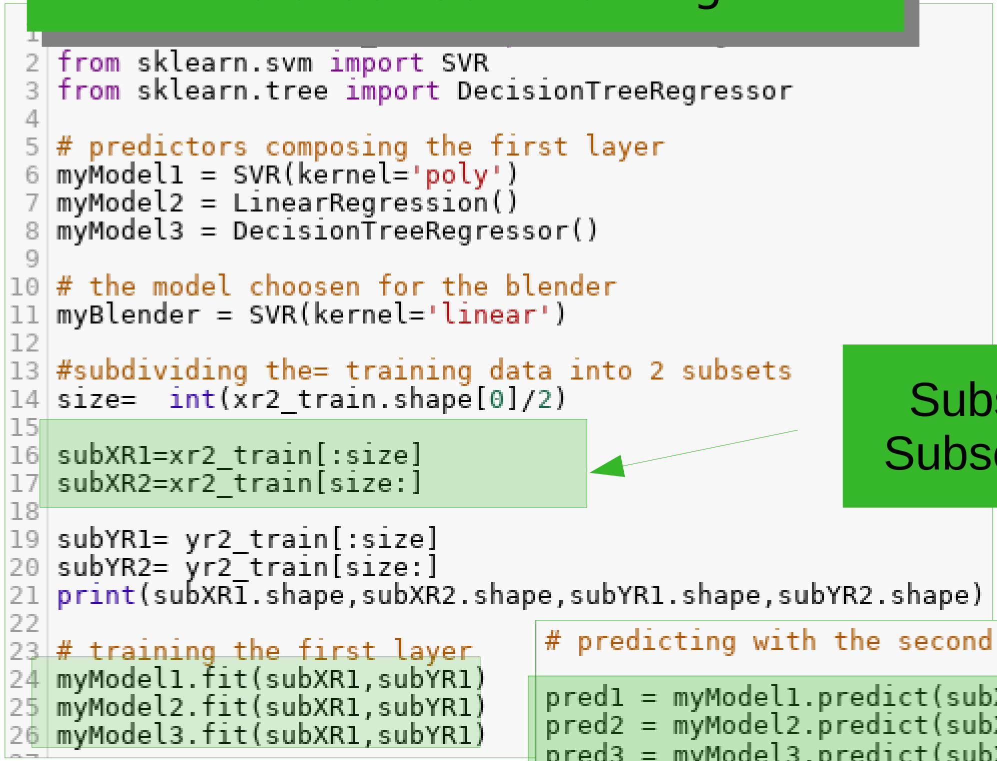
- An example using 3 Predictors and 1 blender (2 layers)



- To make a prediction, the new instance will go through the first layer.
- The resulting predictions will serve as input for the second layer.
- The prediction made by this later one is the final result.

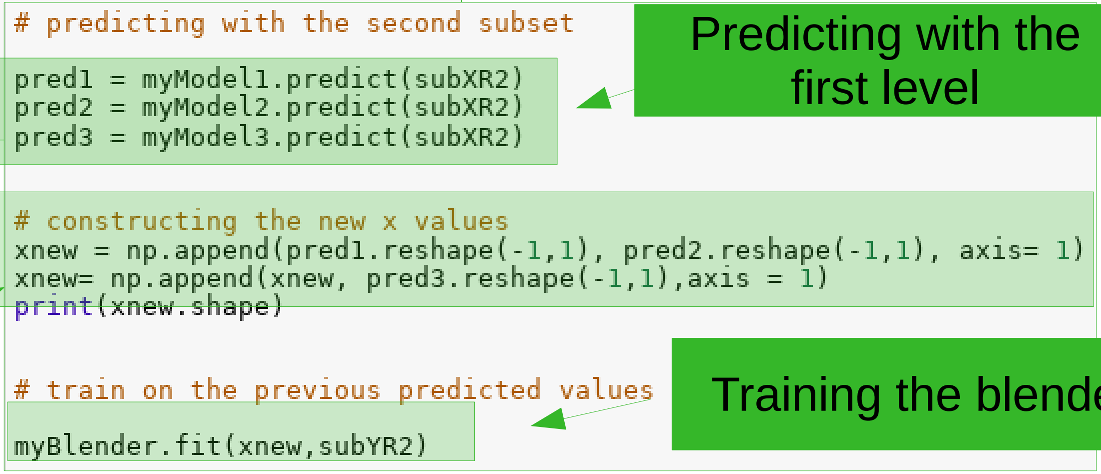
[By Amina Delali]

Hold-out set: Training



```
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor

# predictors composing the first layer
myModel1 = SVR(kernel='poly')
myModel2 = LinearRegression()
myModel3 = DecisionTreeRegressor()

# the model choosen for the blender
myBlender = SVR(kernel='linear')

#subdividing the= training data into 2 subsets
size=  int(xr2_train.shape[0]/2)

subXR1=xr2_train[:size]
subXR2=xr2_train[size:]

subYR1= yr2_train[:size]
subYR2= yr2_train[size:]
print(subXR1.shape,subXR2.shape,subYR1.shape,subYR2.shape)

# training the first layer
myModel1.fit(subXR1,subYR1)
myModel2.fit(subXR1,subYR1)
myModel3.fit(subXR1,subYR1)
```

Subset xr1 for training
Subset xr2 for predicting



```
# predicting with the second subset

pred1 = myModel1.predict(subXR2)
pred2 = myModel2.predict(subXR2)
pred3 = myModel3.predict(subXR2)

# constructing the new x values
xnew = np.append(pred1.reshape(-1,1), pred2.reshape(-1,1), axis= 1)
xnew= np.append(xnew, pred3.reshape(-1,1),axis = 1)
print(xnew.shape)


# train on the previous predicted values

myBlender.fit(xnew,subYR2)
```

Predicting with the first level

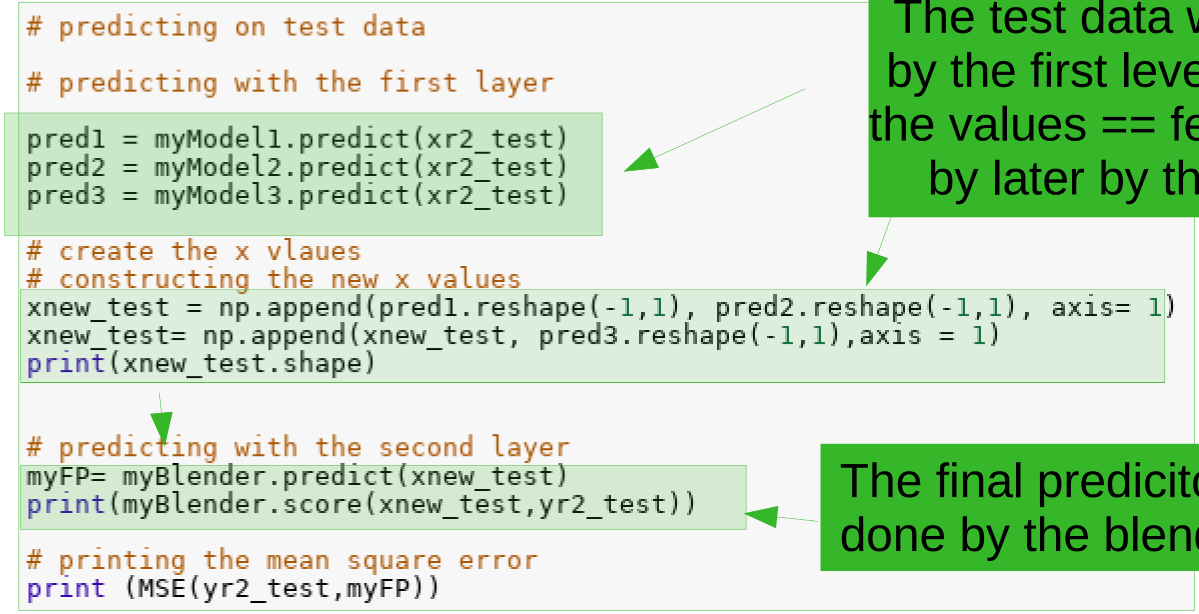
Training the first level

Generate the new features

Training the blender

[By Amina Delali]

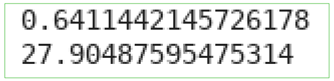
Hold-out set: Testing



```
# predicting on test data

# predicting with the first layer

pred1 = myModel1.predict(xr2_test)
pred2 = myModel2.predict(xr2_test)
pred3 = myModel3.predict(xr2_test)

# create the x vlaues
# constructing the new x values
xnew_test = np.append(pred1.reshape(-1,1), pred2.reshape(-1,1), axis= 1)
xnew_test= np.append(xnew_test, pred3.reshape(-1,1),axis = 1)
print(xnew_test.shape)


# predicting with the second layer
myFP= myBlender.predict(xnew_test)
print(myBlender.score(xnew_test,yr2_test))

# printing the mean square error
print (MSE(yr2_test,myFP))
```

The test data will be used by the first level to predict the values == features used by later by the blender

The final prediciton, will be done by the blender.



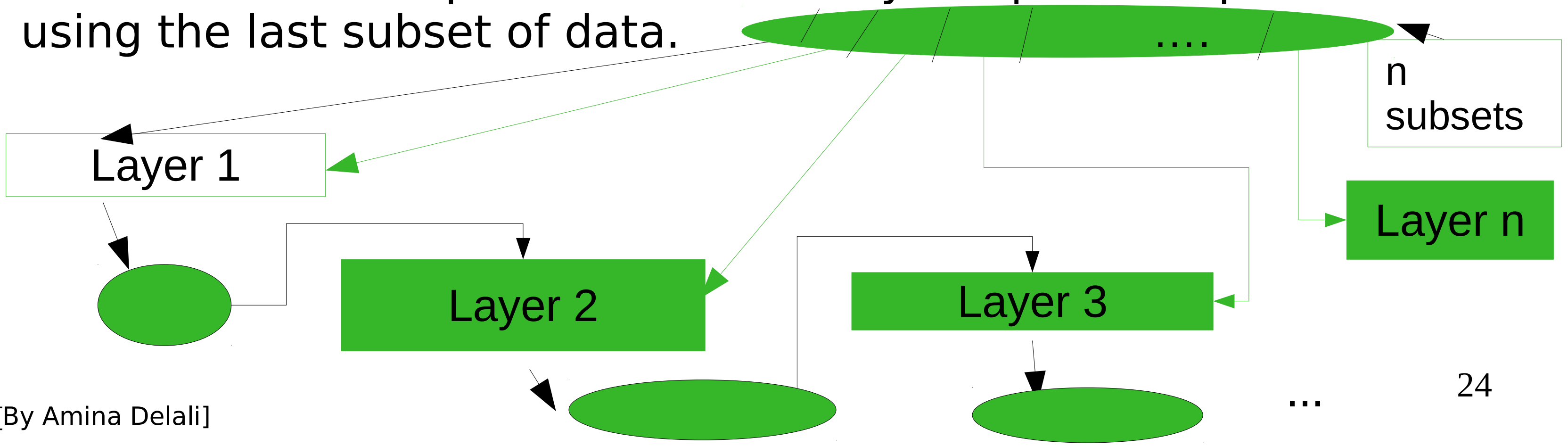
```
0.6411442145726178
27.90487595475314
```

[By Amina Delali]

6-Stacking or blending

Hold-out set: generalization

- Its possible to train several type of blenders. And , each one can be a set of models.
- The idea is to divide the original training set into several subsets: n subsets ==> n layers (n-1 blending phase)
- The first set of predictors will train from the first subset, and make prediction with the second one.
- The second set of predictors will train from the previous predictions. And then make new predictions using the third subset.
- The process is repeated until the last subset of predictors: it will train from the last predictions made by the previous predictors using the last subset of data.



[By Amina Delali]

References

- Aurélien Géron. Hands-on machine learning with Scikit-Learn and Tensor-Flow: concepts, tools, and techniques to build intelligent systems. O’Reilly Media, Inc, 2017.
- Scikit-learn.org. scikit-learn, machine learning in python. On-line at https://scikit-learn.org/stable/. Accessed on 03-11-2018.

Thank you!

FOR ALL YOUR TIME