INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Ceará

HUAWEI

Ensemble Learning

Douglas Chielle
douglas.chielle@ifce.edu.br

Objectives

Upon completion of this lecture, you will:

- understand what is the idea of ensemble learning;
- know the main concepts about the most popular Ensemble methods:
  - Bagging;
  - Boosting; and
  - Stacking.
- know how to use these methods with scikit-learn.

Content

Ensemble Learning

- If you ask a complex question to thousands of random people and aggregate their answers, this aggregated answer will be better than an expert’s answer in most cases.
- This is known as *wisdom of the crowds*.
- Similarly, when multiple predictors are used, the integrated generalization capability can be much stronger than that of a single predictor.
- ***Ensemble learning*** is a machine learning paradigm in which multiple predictors are trained and combined to solve the same problem.

Voting Classifiers

Voting classifiers

- Suppose you have trained a few classifiers.
- We can create an even better classifier by aggregating the predictions of each classifier.
- If this aggregation is done by predicting the class that gets the most votes, then the classifier is called a ***hard voting classifier***.
- If the aggregation is done by predicting the class with the highest probability, averaged over all the individual classifiers, then it is known as ***soft-voting classifier***.

Example: Classification

```python
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier

bc = load_breast_cancer()
X, y = bc.data, bc.target
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
                                                    random_state=7)
logistic = LogisticRegression(max_iter=10000,
                              random_state=7)
gNB = GaussianNB()
tree = DecisionTreeClassifier(max_leaf_nodes=16, random_state=7)
voting = VotingClassifier(
    estimators=[('lr', logistic), ('gnb', gNB), ('tree', tree)],
    voting='hard')

for clf in (logistic, gNB, tree, voting):
  clf.fit(X_train, y_train)
  print(clf.__class__.__name__, clf.score(X_test, y_test))
```

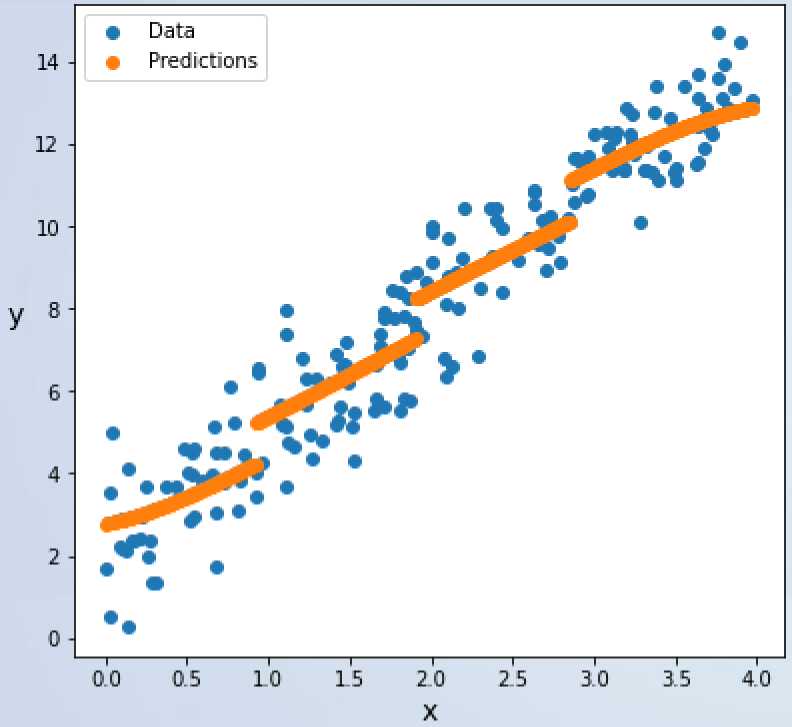
The output of this code is:

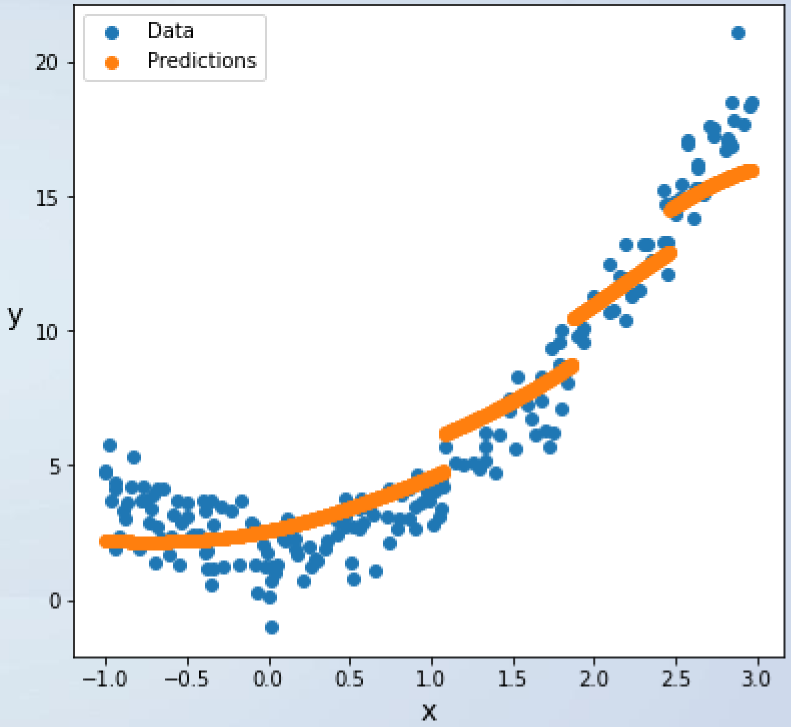
```
LogisticRegression 0.947
GaussianNB 0.956
DecisionTree  0.930
VotingClassifier 0.982
```

Example: Regression

```
from sklearn.ensemble import VotingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor

LR = LinearRegression()
sv = SVR(kernel='rbf')
tree = DecisionTreeRegressor(max_depth=2)

voting = VotingRegressor(
    estimators=[('lr', LR), ('svr', sv),
                ('tree', tree)])
```

Example: Regression



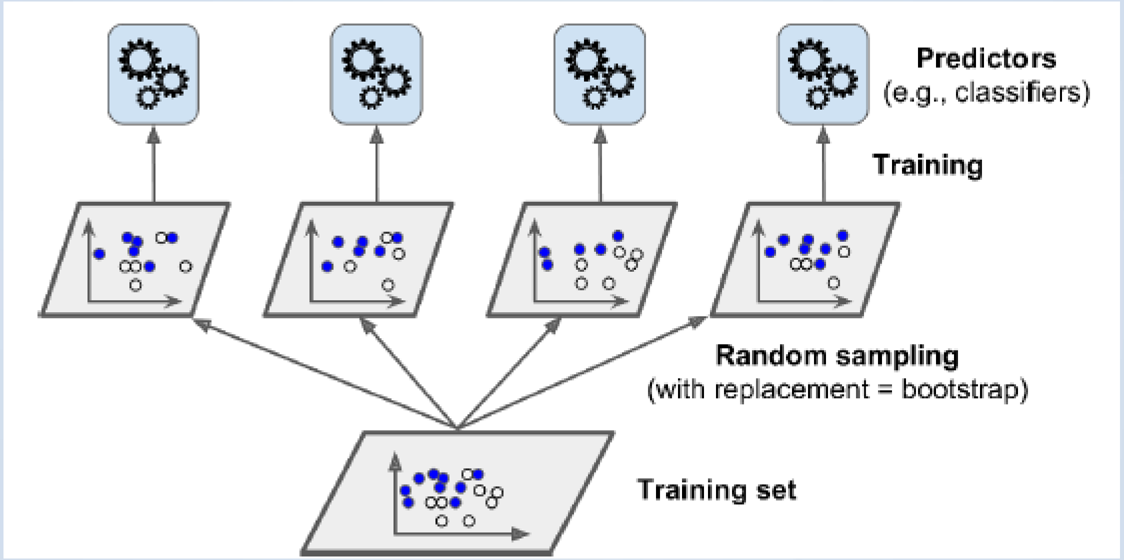


Bagging and Pasting

Bagging and Pasting

- Another approach to get a diverse set of classifiers is to use the same training algorithm for every predictor and train them on different random subsets of the training set.
- When sampling is performed *with replacement*, this method is called ***bagging*** (bootstrap aggregating).
- When sampling is performed *without replacement*, it is called ***pasting***.
- Once all predictors are trained, the ensemble can make a prediction for a new instance by simply aggregating the predictions of all predictors.

Bagging and Pasting

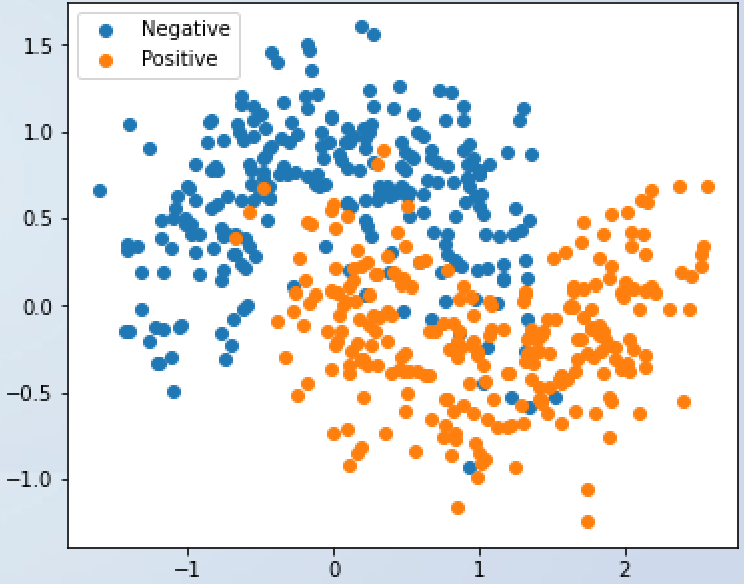


(Géron, 2019)

Example: Classification

- For this bagging example, lets use a dataset built with the make_moons function from sklearn:

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30,
random_state=7)
X_train, X_test, y_train, y_test = train_test
_split(X, y, random_state=7)
```



Example: Classification

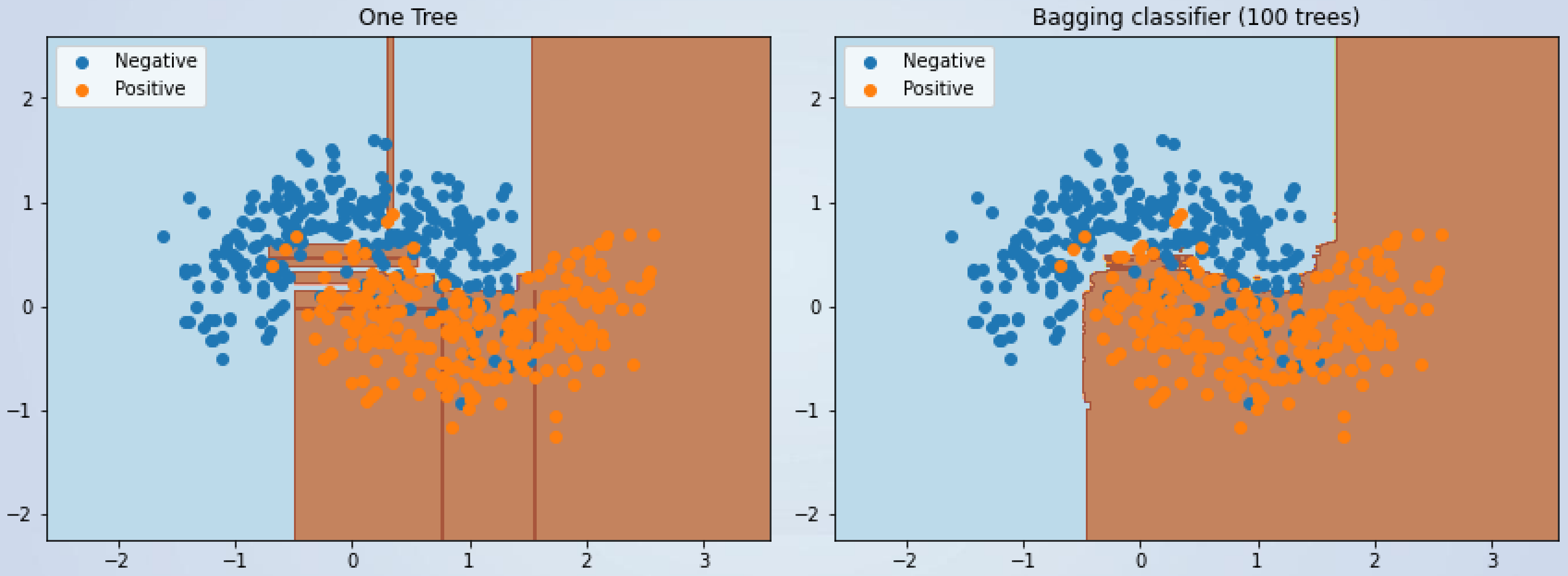
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(max_depth=8, random_state=7)
bagging = BaggingClassifier( DecisionTreeClassifier(), n_estimators=100,
                            max_samples=100, bootstrap=True, n_jobs=-1,
                            random_state=7)

bagging.fit(X_train, y_train)
tree.fit(X_train, y_train)

print('One decision tree', tree.score(X_test, y_test))
print('Bagging', bagging.score(X_test, y_test))
```

Output:

```
Decision tree 0.856
 Bagging 0.896
```

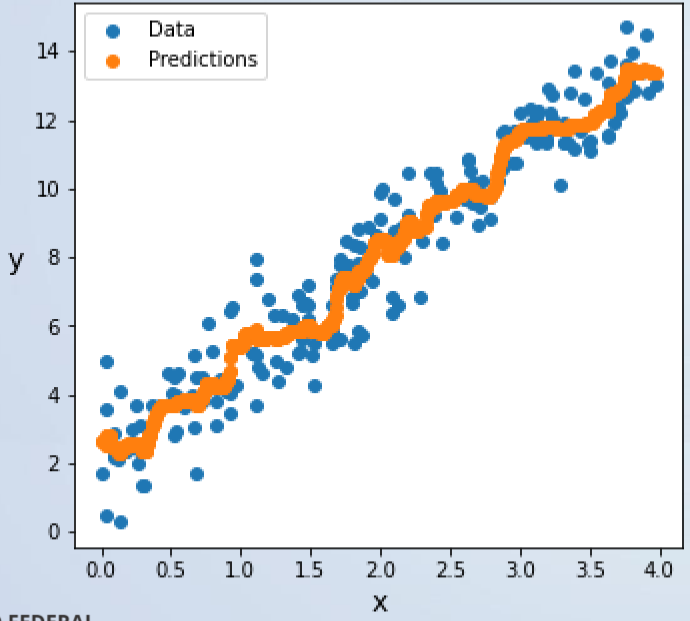
For pasting we would need to change the bootstrap parameter to `False`.

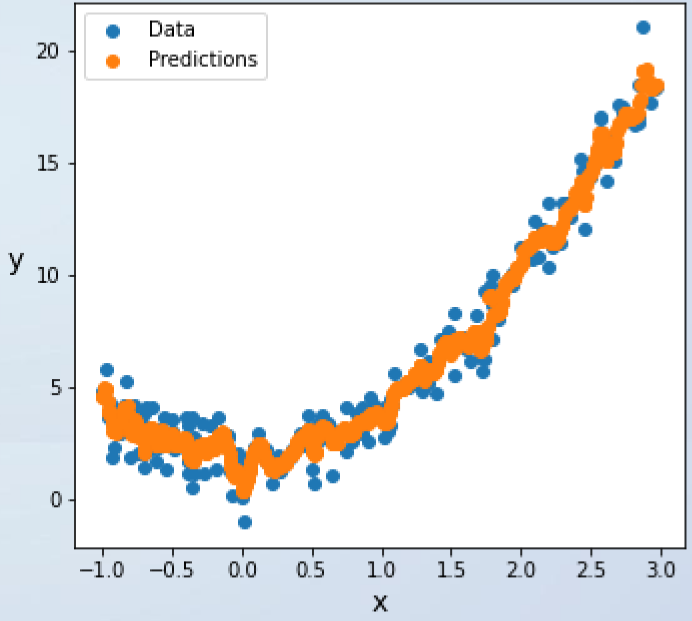
Example: Classification



Example: Regression

```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor

bagging = BaggingRegressor( DecisionTreeRegressor(max_depth=4), n_estimators=100, max_samples=100, bootstrap=True,
                           n_jobs=-1, random_state=7)
```





INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Ceará

HUAWEI

Out-of-bag (oob)

- With bagging, some instances may be sampled several times for any given predictor, while others may not be sampled at all.
- This training instances that are not sampled are called ***out-of-bag*** (oob) instances.
- Since a predictor never sees the oob instances during training, it can be evaluated on these instances, without the need for a separate validation set.
- You can evaluate the ensemble itself by averaging out the oob evaluations of each predictor.

(Gerón, 2019)

Example

```
bagoob = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=100,
    max_samples=100, bootstrap=True, n_jobs=-1,
    oob_score=True, random_state=7 )

bagoob.fit(X_train, y_train)

print('Out-of-bag: ', bagoob.oob_score_)
print('Ensemble: ', bagoob.score(X_test, y_test))
```

Output:

```
Out-of-bag 0.923
 Ensemble 0.896
```

Random Patches and Random Subspaces

- The `BaggingClassifier` class supports sampling the features as well.
- Sampling is controlled by two hyperparameters: `max_features` and `bootstrap_features`.
- Thus, each predictor will be trained on a random subset of the input features, which is useful when dealing with high-dimensional inputs.
- Sampling both training instances and features is called the ***Random Patches*** method.
- Keeping all training instances but sampling features is called the ***Random Subspace*** method.

(Gerón, 2019)

Random Forests

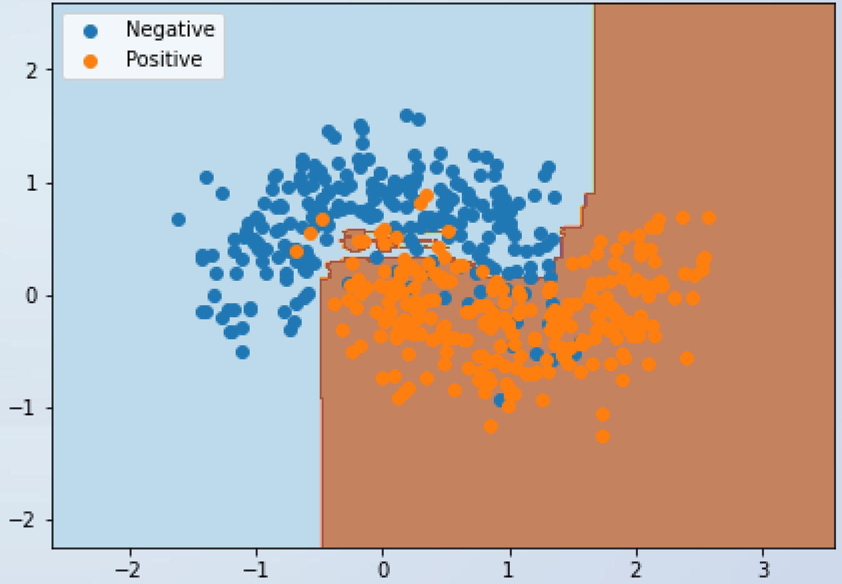
Random Forests

- A ***Random Forest*** is an ensemble of Decision Trees.
- Despite its simplicity, this is one of the most powerful Machine Learning algorithms available today.
- When splitting each node during the construction of a tree, the best split is found either from all input features or a random subset of size $max\_features$.
- Typically, only $\sqrt{n}$ or $\log_2 n$ features are considered, where n is the number of features.
- Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias.
- In practice the variance reduction is often significant, hence yielding an overall better model.

Example: Classification

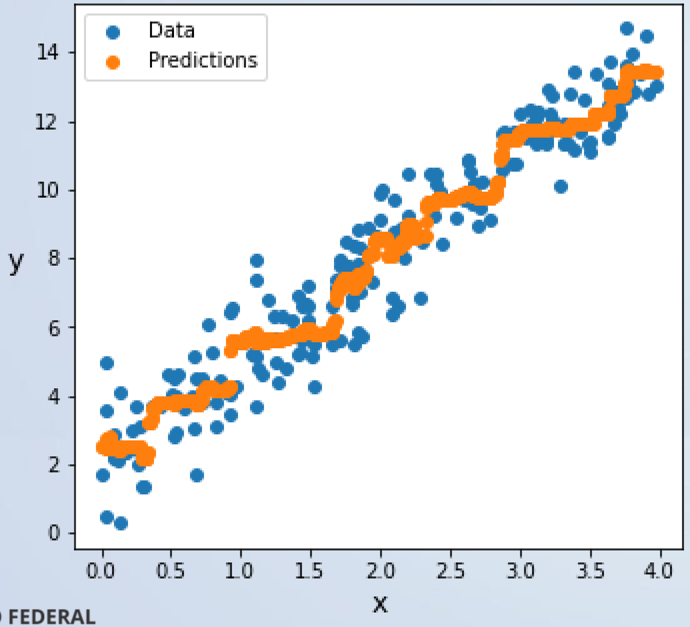
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, max_leaf_nodes=16, random_state=7)
rf.fit(X_train, y_train)

print('RF: ', rf.score(X_test, y_test))
```

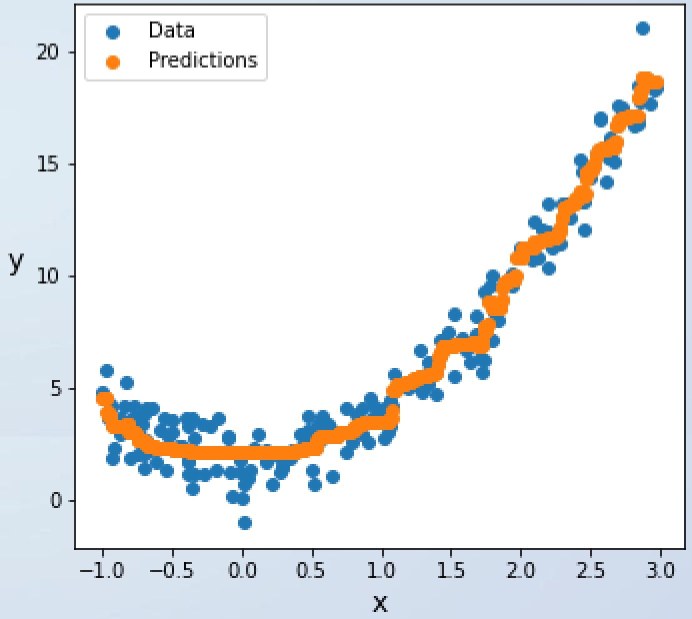
Output: `RF: 0.904`



Example: Regression

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=100, max_depth=4, random_state=7)
```





Feature importance

- Another great quality of Random Forests is that they make it easy to measure the relative importance of each feature.
- Scikit-Learn measures a feature's importance by looking at how much the tree nodes that use that feature reduce impurity on average (across all trees in the forest).
- Scikit-Learn computes this score automatically for each feature after training, then it scales the results so that the sum of all importances is equal to 1.
- You can access the result using the `feature_importances_` variable.

(Gerón, 2019)

Feature importance (Gerón, 2019)

```
from sklearn.datasets import load_iris

iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
rnd_clf.fit(iris["data"], iris["target"])

for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
  print(name, score)
```

Output:

```
sepal length (cm) 0.10562816803464888
sepal width (cm) 0.024812579550922375
petal length (cm) 0.42969033978737586
petal width (cm) 0.4398689126270529
```

Boosting

Boosting

- The idea of Boosting is to combine several weak classifiers and build a much better classifier.
- A weak learning algorithm is a method that performs just slightly better than random guessing.
- The general idea of most boosting methods is to train weak learners sequentially, each trying to correct its predecessor.
- The most popular boosting methods are ***AdaBoost*** and ***Gradient Boosting***.
- In general, $Boosting \succ Random\ Forest \succ Bagging \succ Single\ Tree$.

AdaBoost (Adaptive Boosting)

- The idea of AdaBoost is to update the weight of the misclassified samples (on the training set), making the next predictor focus more on these samples.
- For the first predictor, the weights of the m training samples are set to $\frac{1}{m}$.
- In general, after training the j^{th} predictor, a weighted error rate for this predictor is computed:

$$r_j = \frac{\sum_{i=1}^{m} w_i \mathbb{I}(y_i \neq \hat{y}_i)}{\sum_{i=1}^{m} w_i}$$

  - $\mathbb{I}(y_i \neq \hat{y}_i) = \begin{cases} 1, & if\ y_i \neq \hat{y}_i \\ 0, & if\ y_i = \hat{y}_i \end{cases}$

AdaBoost (Adaptive Boosting)

- Afterwards, it computes a weight for this predictor:

$$\alpha_j = \eta \ln \frac{1 - r_j}{r_j}$$

- Next, the weights of the misclassified samples are updated:

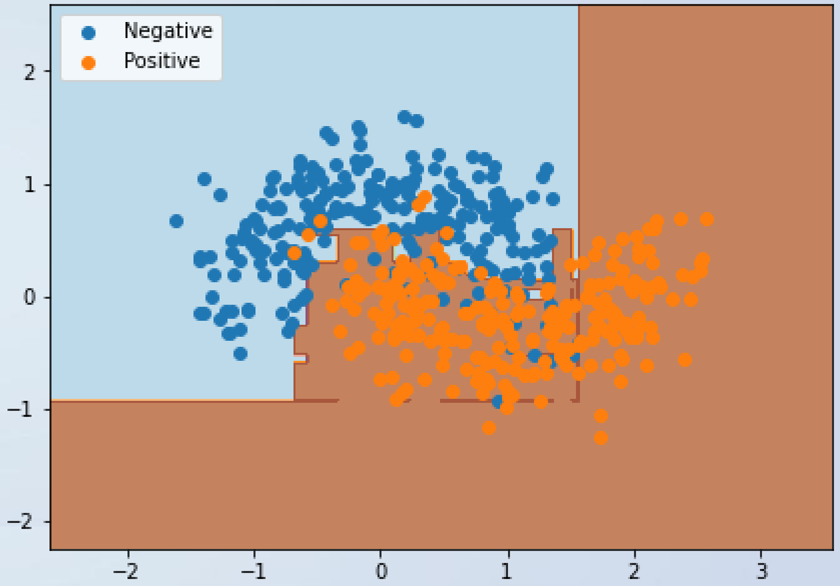
$$w_i = w_i \exp(\alpha_j)$$

- Then, the next predictor is trained using the updated weights.
- This process is repeated until the desired number of predictors is reached, or when a perfect predictor is found.
- Finally, to make predictions AdaBoost computes the predictions of all predictors and weights them using the weights α_j.

Example: Classification

```python
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1),
    n_estimators=200, learning_rate=0.5,
    random_state=7)

ada_clf.fit(X_train, y_train)
print(ada_clf.score(X_test, y_test))
```

Output: 0.856

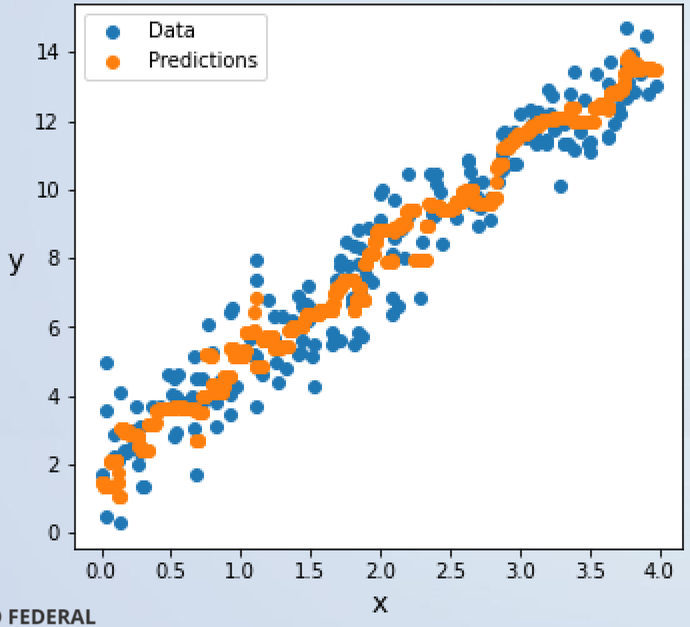


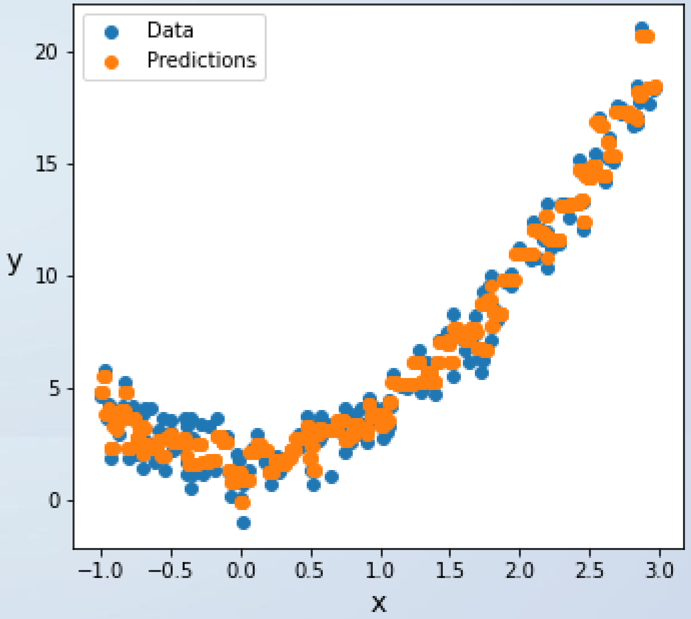
INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Ceará

HUAWEI

Example: Regression

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor

ada = AdaBoostRegressor( DecisionTreeRegressor(max_depth=4), n_estimators=100, random_state=7)
```





INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Ceará

HUAWEI

Gradient Boosting

- Fit an additive model (ensemble) $\sum_t \rho_t h_t(x)$ in a forward stage-wise manner.
- In each stage, introduce a weak learner to compensate the shortcomings of the existing weak learners.
- However, unlike AdaBoost that tweaks the instance weights at every iteration, this method tries to fit the new predictor to the *residual errors* made by the previous predictor.
- As this method is mostly used with Decision Trees, it is commonly called Gradient Tree Boosting (GTB) or Gradient Boosted Decision Trees (GBDT).

Gradient Boosting (regression)

- Consider the regression problem where the goal is to reach a model F to predict values of the form $\hat{y} = F(x)$ by minimizing the MSE.
- Our loss function is: $L\big(y, F(x)\big) = \frac{\left(y - F(x)\right)^2}{2}$ and we want to minimize $J = \sum_{i=1}^{m} L\big(y_i, F(x_i)\big)$ by adjusting $F(x_1), F(x_2), \cdots, F(x_m)$.
- We can treat $F(x_j)$ as parameters and take derivatives:

$$\frac{\partial J}{\partial F(x_j)} = \frac{\partial \sum_{i=1}^{m} L\big(y_i, F(x_i)\big)}{\partial F(x_j)} = \frac{\partial L\left(y_j, F(x_j)\right)}{\partial F(x_j)} = F(x_j) - y_j$$

- Therefore, we can interpret the residuals as gradients.

$$y_j - F(x_j) = -\frac{\partial J}{\partial F(x_j)}$$

Gradient Boosting (regression)

- To improve our model, we can "walk" in the direction of the negative gradient.

$$F_{t+1}(x_j) = F_t(x_j) + h_t(x_j)$$

- Since $h_t(x_j) = y_j - F_t(x_j) = -\frac{\partial J}{\partial F_t(x_j)}$, we have:

$$F_{t+1}(x_j) = F_t(x_j) - \frac{\partial J}{\partial F_t(x_j)}$$

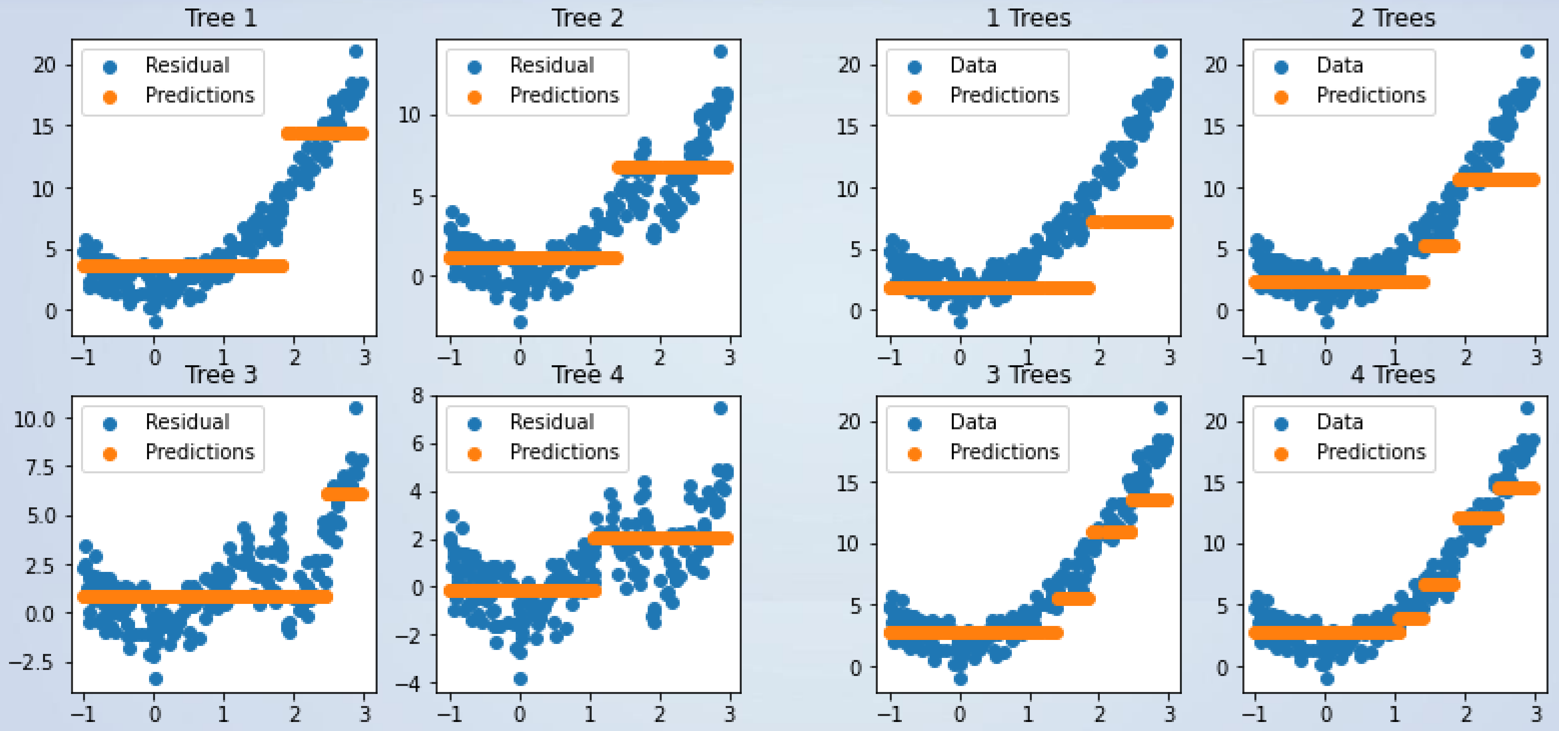
- In summary, we fit h to the negative gradient and update our model F based on that.

Example

- To better understand Gradient Boosting, let’s look at a regression problem.
- We will consider the nonlinear data we have been using for regression.
- The following code sequentially builds four decision stumps and ajust each of them to the residual error of the previous tree.

```python
from sklearn.tree import DecisionTreeRegressor

alpha = 0.5          # learning rate
Learners = list();
dY = np.ravel(y2)

for k in range(4):
  Learners.append(DecisionTreeRegressor(max_depth=1).fit(x2, dY))
  dY = dY - alpha*Learners[k].predict(x2)
```

Example



Tree 1
Tree 2
1 Trees
2 Trees
Tree 3
Tree 4
3 Trees
4 Trees
Residual
Predictions
Data

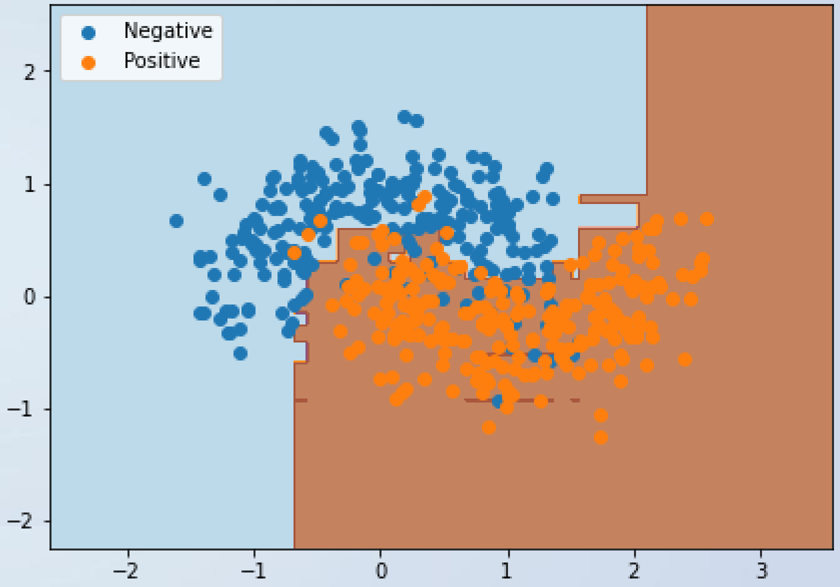
XGBoost (Chen, 2016)

- *Extreme Gradient Boosting (*XGBoost) is a scalable machine learning system for tree boosting.
- The system is available as an open source package.
- XGBoost is often an important component of the winning entries in ML competitions.
- The most important factor behind the success of XGBoost is its scalability in all scenarios.
- The scalability of XGBoost is due to several important systems and algorithmic optimizations.
- More information: https://dl.acm.org/doi/pdf/10.1145/2939672.2939785

Example: Classification

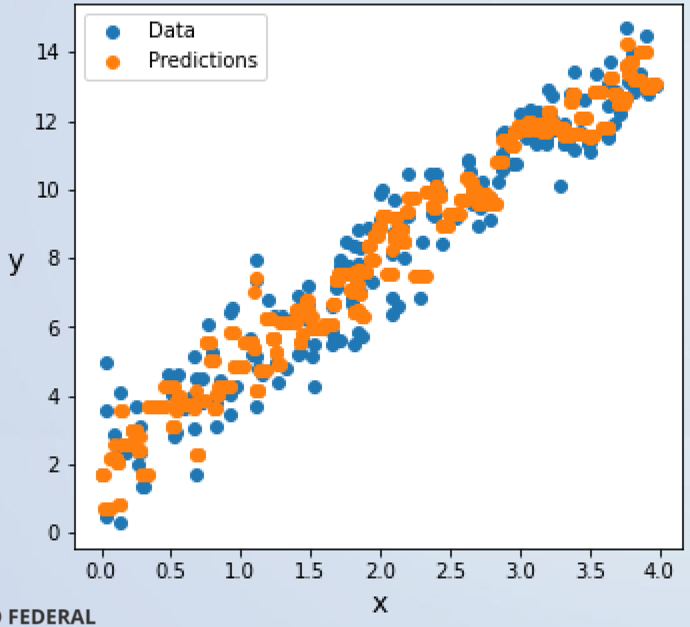
```
from sklearn.ensemble import GradientBoostingClassifier

gb_clf = GradientBoostingClassifier(max_depth=1,
                                    n_estimators=200,
                                   learning_rate=0.5,
                                    random_state=7)


gb_clf.fit(X_train, y_train)
print(gb_clf.score(X_test, y_test))
```

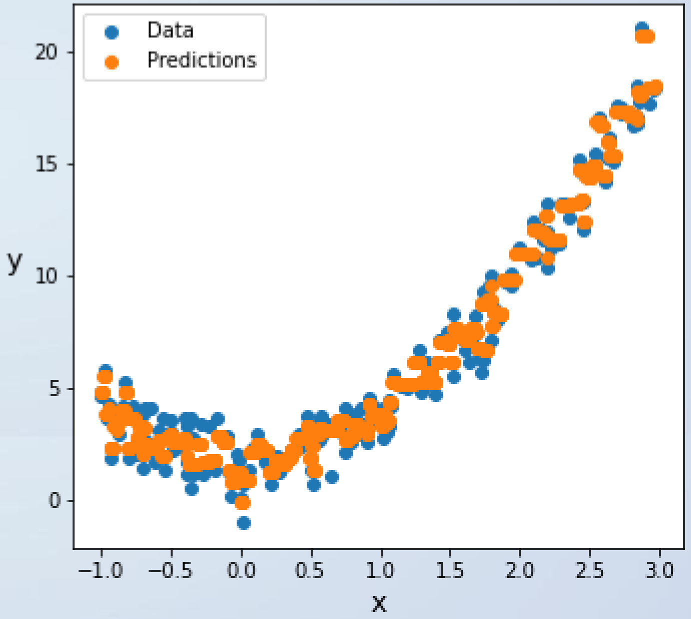
Output: 0.88



Example: Regression

```
from sklearn.ensemble import GradientBoostingRegressor

gb = GradientBoostingRegressor( max_depth=4, n_estimators=100, random_state=7)
```





INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Ceará

HUAWEI

Stacking

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Ceará

HUAWEI

Stacking

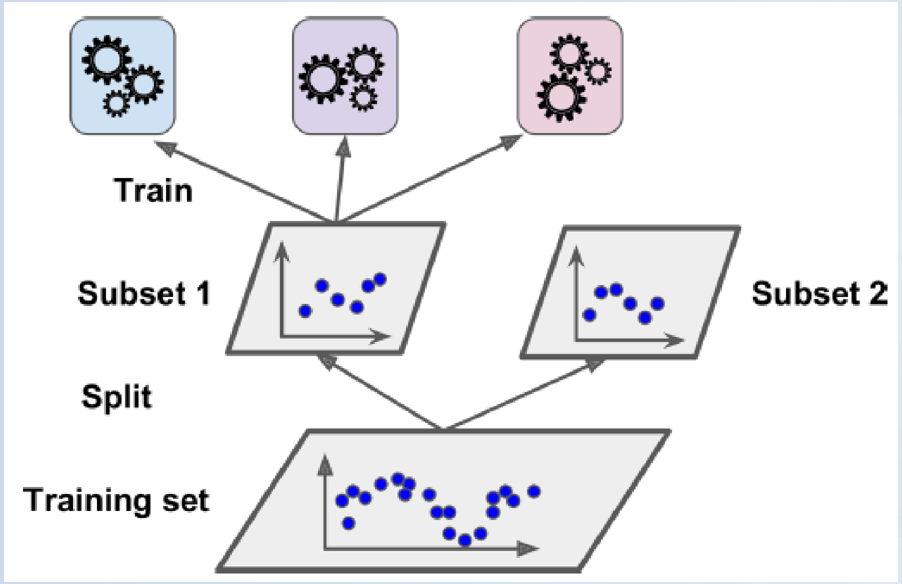
- Stacking is an ensemble machine learning algorithm that learns how to best combine the predictions from multiple well-performing machine learning models.
- It involves combining the predictions from multiple machine learning models on the same dataset, like bagging and boosting.
- However:
  - Unlike bagging, in stacking, the models are typically different and fit on the same dataset.
  - Unlike boosting, in stacking, a single model is used to learn how to best combine the predictions from the contributing models.

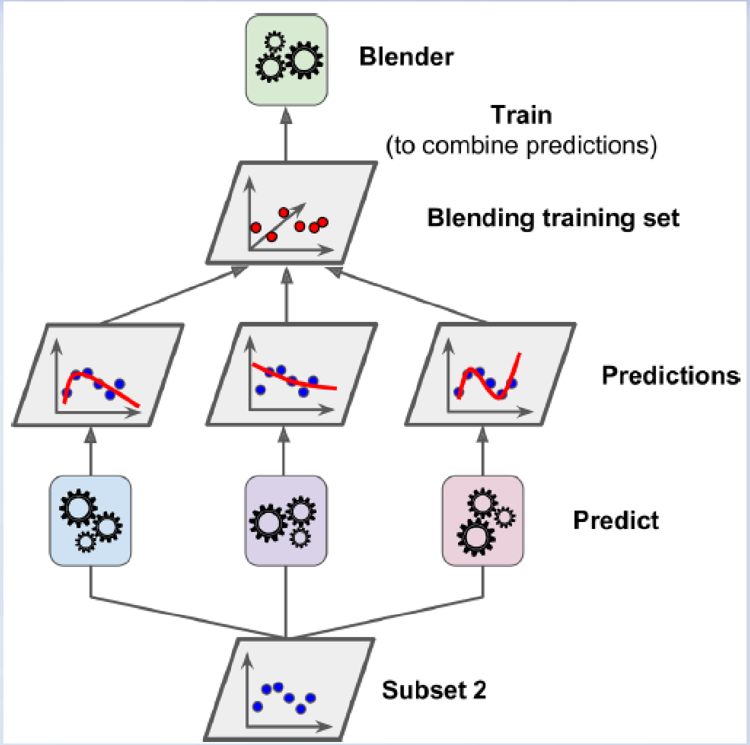
https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/

Stacking (Gerón, 2019)

- To train the blender, a common approach is to use a hold-out set.
- First, the training set is split into two subsets. The first subset is used to train the predictors in the first layer.
- Next, the first layer’s predictors are used to make predictions on the second (held-out) set.
- This ensures that the predictions are “clean”, since the predictors never saw these instances during training.
- We then create a new training set using these predicted values as input features and keeping the target values.
- Finally, the blender is trained on this new training set, so it learns to predict the target value, given the first layer’s predictions.

Stacking (Gerón, 2019)



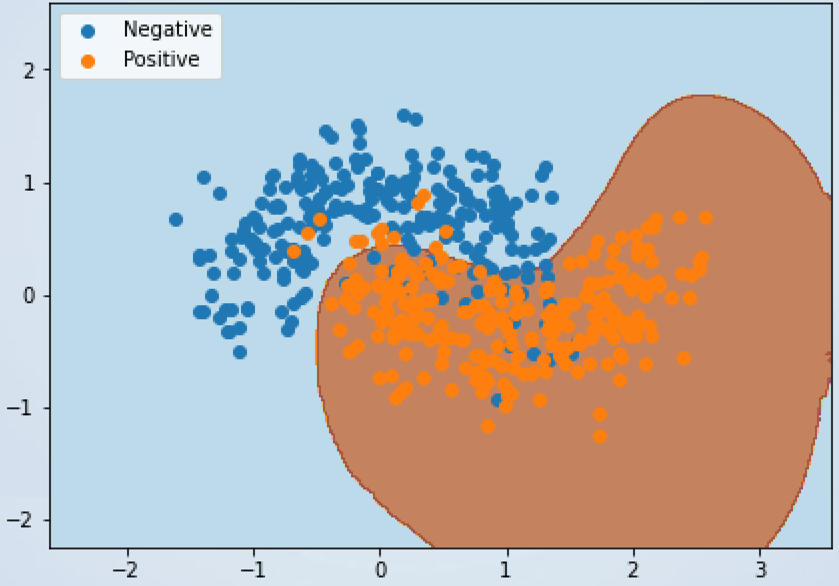


Example: Classification

```
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

estimators = [('rf', RandomForestClassifier(n_estimators=50, random_state=7)),
              ('svm', SVC(random_state=7))]

stacking = StackingClassifier(estimators=estimators,
                              final_estimator=LogisticRegression())
```

```
Output: 0.888
```

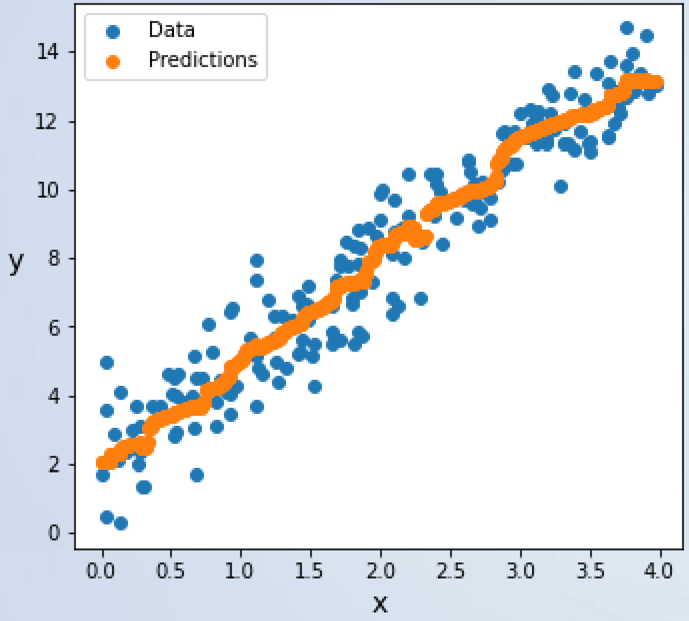
Example: Classification

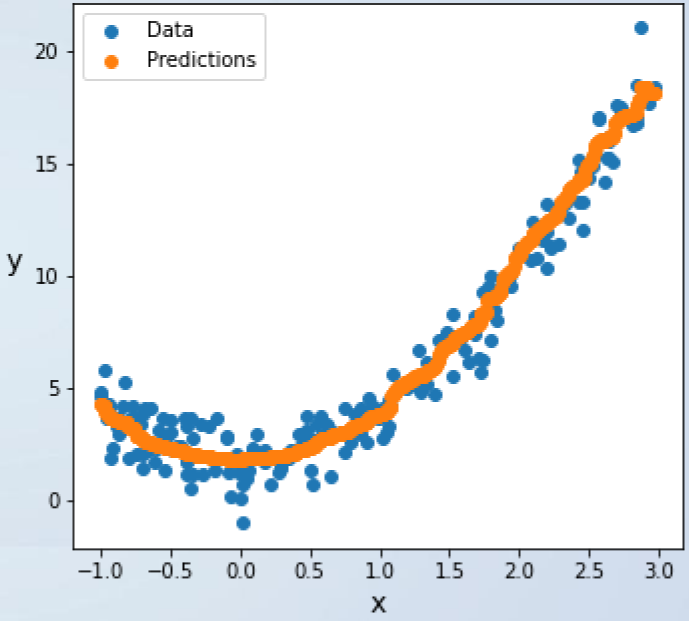


Example: Regression

```python
from sklearn.ensemble import StackingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR

estimators = [('rf', RandomForestRegressor(n_estimators=50, max_depth=4,
                                            random_state=7)),
              ('svr', SVR())]

stacking = StackingRegressor(estimators=estimators,
                              final_estimator=LinearRegression())

stacking.fit(X_train, y_train)
print(stacking.score(X_test, y_test))
```

Example: Regression





References

1. Géron, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.
2. Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016.
3. Cheng Li. "A Gentle Introduction to Gradient Boosting".
4. https://scikit-learn.org/stable/modules/ensemble.html

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Ceará

HUAWEI

That's all Folks!