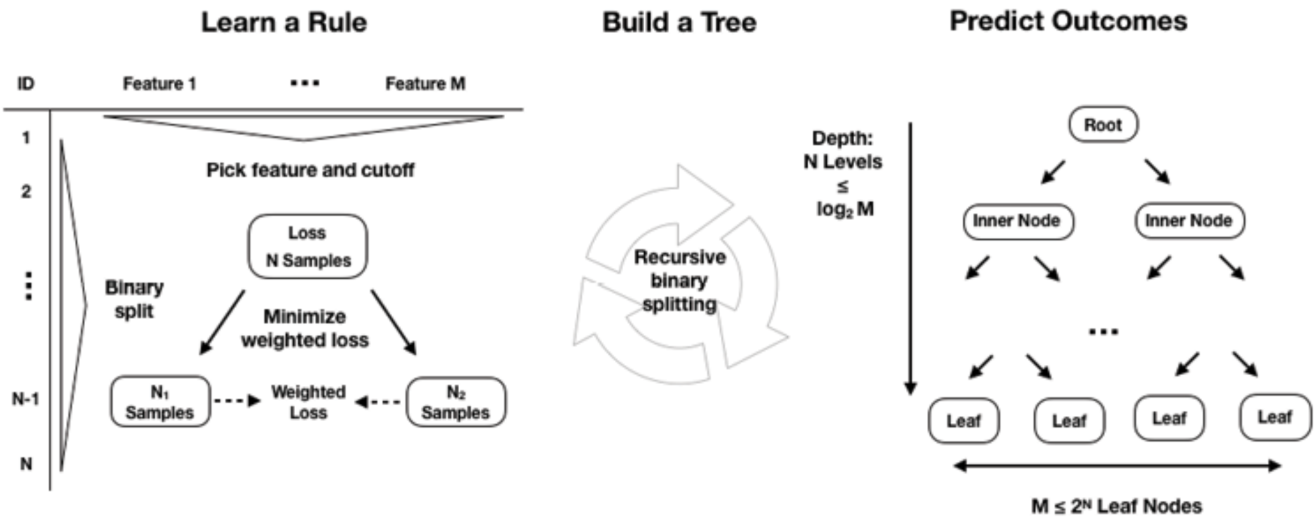
Last Class



From: Machine Learning for Algorithmic Trading - Second Edition<\center>

Ensemble Learning and Random Forests

- Suppose you ask a question to random people?
  - Which response is the best? People or Expert's response?
- The voice of the people is god's voice?
- In plain English, "wisdom of the crowd"
- Ensemble methods aggregate predictions (such as classifiers or regressors)
- Often a group of predictions is better than the best individual predictor
- TWC's example

Ensemble Learning and Random Forests

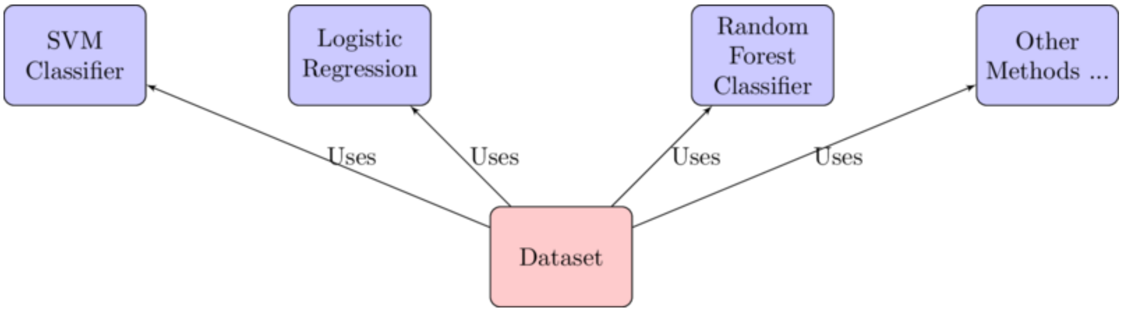
- For instance, suppose you created a group of trees and select the most 'voted' output
- Netflix Prize Competition ([https://netflixprize.com/leaderboard.html (https://netflixprize.com/leaderboard.html)](https://netflixprize.com/leaderboard.html))
- We will discuss the following methods
  - Bagging
  - Boosting
  - Stacking
  - Random Forests
  - ...

Import libraries

In [1]:
```
%load_ext tikzmagic
```

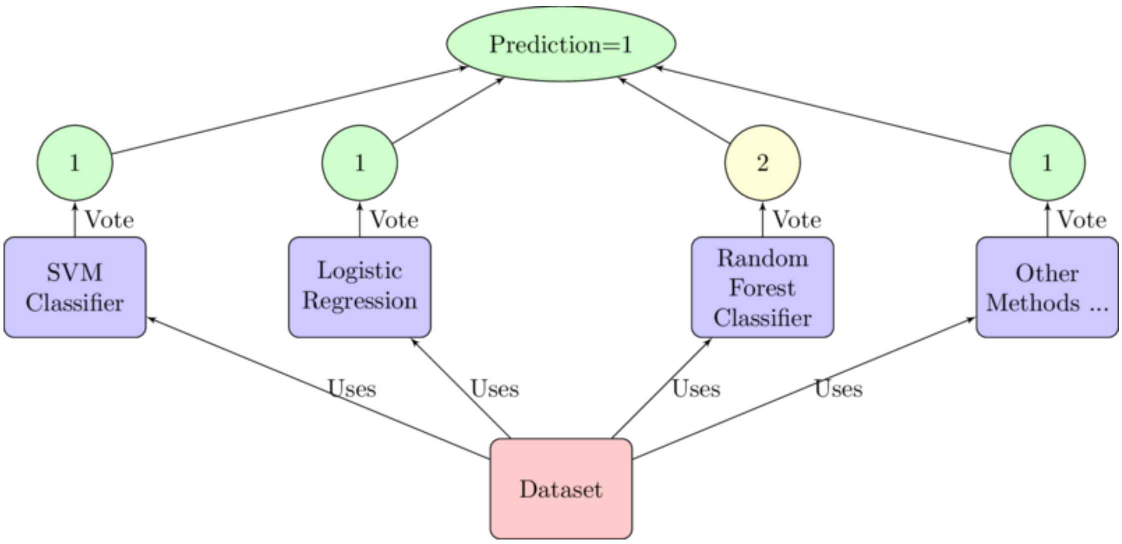
Ensemble Methods

In [2]:
```
%%tikz -s 800,600
% Ensemble method
\input{ensemble.tikz}
```



Prediction Voting

In [3]:
```
%%tikz -s 800,600
% Prediction
\input{prediction.tikz}
```



Ensemble methods

- This voting classifier often achieves higher accuracy than the best classifier in the ensemble
- Even if each classifier is a *weak learner* the ensemble can be a *strong leaner*
- Lets observe what happens with a biased coin in the light of big numbers law

Bernoulli distribution and the law of big numbers

- Discrete probability distribution of a random variable which takes the value 1 with probability p and the value 0 with probability $q = 1 - p$
- It can be used to represent a (possibly biased) coin toss where 1 and 0 would represent "heads" and "tails" (or vice versa)
- The probability mass function f of this distribution, over possible outcomes k, is

$$f(k;p) = \begin{cases} p & \text{if } k = 1, \\ q = 1 - p & \text{if } k = 0 \end{cases}$$

Bernoulli distribution and the law of big numbers

In [4]:
```
import numpy as np

def bernouli(p, k=1):
    """
    Bernouli distribution
    """
    return (np.random.random(k)) > 1 - p
```

In [5]:
```
bernouli(0.5, 10)
```

Out[5]:
```
array([ True,  True,  True,  True,  True,  True, False,  True,  Tr
ue,
        True])
```

In [6]:
```
np.random.random() > 0.5
```

Out[6]:
```
False
```

Bernoulli distribution and the law of big numbers

Lets check out what happens with the proportion of heads and tails for a biased coin along the time.

In [7]:
```
def heads_proportion(p=0.5, k=1):
    """
    Return the proportion of heads and tails for
    bernoulli distribution
    """
    outcome = bernouli(p, k)
    return np.count_nonzero(outcome) / k
```

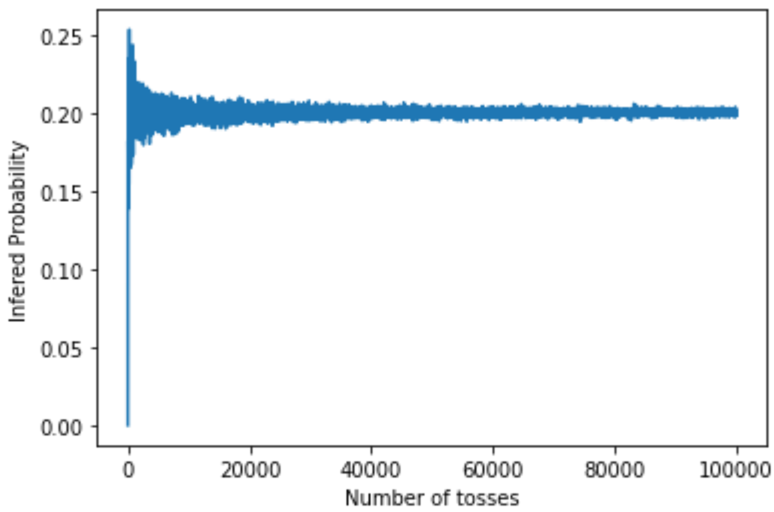
In [8]:
```
# heads_proportion(0.5, 10)
# np.count_nonzero(bernouli(0.5, 10))
heads_proportion(0.5, 100)
```

Out[8]: 0.47

Bernoulli distribution and the law of big numbers

In [9]:
```
p = 0.2
domain = range(1, 100000, 10)
y = [heads_proportion(p, x) for x in domain]
```

In [10]:
```
from matplotlib import pyplot as plt

plt.plot(domain, y)
plt.ylabel("Infered Probability")
plt.xlabel("Number of tosses")
plt.show()
```



What about ensemble methods?

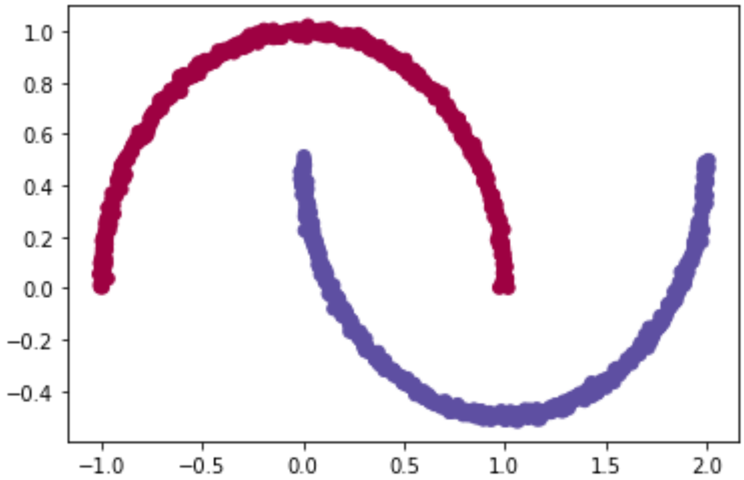
- Suppose you build an ensemble containing 1,000 classifiers that are individually correct only 51% of the time
- If you predict the majority voted class, you can hope high accuracy!
- This is only true if all classifiers are perfectly independent, making uncorrelated errors
- Each toss of a coin would corresponds to a classification in the ensemble method
- One way to get diverse classifiers is to train them using very different algorithms

Make moons dataset

In [11]:
```
from sklearn.datasets import make_moons

np.random.seed(10)
X, y = make_moons(1000, noise=0.01)
```

In [12]:
```
plt.scatter(X[:,0], X[:,1], s=40, c=y, cmap=plt.cm.Spectral)
```

Out[12]: <matplotlib.collections.PathCollection at 0x1185da550>



In [13]:
```
X, y
```

Out[13]:
```
(array([[ 1.55178908, -0.32364143],
        [ 1.95492811,  0.23658871],
        [ 0.05141544,  0.22707542],
        ...,
        [ 2.00746081,  0.4761756 ],
        [ 0.71336134,  0.69238784],
        [-0.11974147,  0.99494821]]),
 array([1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1,
0, 1, 0,
        0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
1, 1, 1,
        0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1,
1, 1, 1,
        1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
0, 1, 1,
        1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
0, 0, 0,
        0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1,
1, 0, 0,
        1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
0, 1, 1,
        0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,
0, 0, 0,
        1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
0, 0, 0,
        0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
1, 0, 1,
        1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0,
1, 1, 1,
        0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0,
1, 1, 1,
        1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0,
1, 0, 1,
        0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0,
0, 0, 0,
        1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1,
0, 0, 1,
        1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
0, 0, 1,
        0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0,
1, 0, 0,
        0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1,
0, 0, 0,
        0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
0, 1, 1,
        1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0,
1, 1, 1,
        0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
1, 0, 0,
        0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1, 0, 1,
        1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 1, 0,
        0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1,
1, 1, 0,
        1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 0,
        0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1,
1, 0, 0,
```

```
        1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0,
1, 1, 1,
        1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1,
        1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
1, 1, 1,
        0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1,
1, 0, 0,
        1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0,
1, 1, 0,
        1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
1, 1, 1,
        0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0,
0, 1, 1,
        0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0,
1, 1, 1,
        1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
0, 1, 1,
        1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
1, 1, 0,
        1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0,
1, 1, 1,
        1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
0, 0, 0,
        1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 0,
        0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1,
1, 0, 0,
        1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0,
1, 1, 1,
        0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
1, 0, 1,
        0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1,
1, 1, 1,
        0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
1, 1, 0,
        1, 1, 0, 0, 0, 0, 1, 1, 0, 0]))
```

Creating Voting Classifier

In [14]:

```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

# Split data into train and test datasets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=42
)
```

Creating Voting Classifier

In [15]:
```
log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_cl
f)],
    voting='hard')

voting_clf.fit(X_train, y_train)
```

Out[15]:
```
VotingClassifier(estimators=[('lr',
                              LogisticRegression(C=1.0, class_weig
ht=None,
                                                 dual=False, fit_i
ntercept=True,
                                                 intercept_scaling
=1,
                                                 l1_ratio=None, ma
x_iter=100,
                                                 multi_class='auto
',
                                                 n_jobs=None, pena
lty='l2',
                                                 random_state=Non
e,
                                                 solver='lbfgs', t
ol=0.0001,
                                                 verbose=0, warm_s
tart=False)),
                             ('rf',
                              RandomForestClassifier(bootstrap=Tru
e,
                                                     ccp_alpha=0.
0,
                                                     class_weight=
None,
                                                     cr...
                                                     oob_score=Fal
se,
                                                     random_state=
None,
                                                     verbose=0,
                                                     warm_start=Fa
lse)),
                             ('svc',
                              SVC(C=1.0, break_ties=False, cache_s
ize=200,
                                  class_weight=None, coef0=0.0,
                                  decision_function_shape='ovr', d
egree=3,
                                  gamma='scale', kernel='rbf', max
_iter=-1,
                                  probability=False, random_state=
None,
                                  shrinking=True, tol=0.001, verbo
se=False))],
                 flatten_transform=True, n_jobs=None, voting='hard
',
                 weights=None)
```

Checking the accuracy of ensemble models

In [16]:
```
from sklearn.metrics import accuracy_score

from sklearn.metrics import accuracy_score
for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        print(clf.__class__.__name__, accuracy_score(y_test, y_pre
d))
```

```
LogisticRegression 0.8775
RandomForestClassifier 0.9975
SVC 1.0
VotingClassifier 0.9975
```

In [17]:
```
test = [1, 0]

#X_test[index], y_test[index]

#log_clf.predict([test]),rnd_clf.predict([test]), svm_clf.predict
([test]), voting_clf.predict([test])
log_clf.predict_proba([test]), 1 ,rnd_clf.predict_proba([test]), 0
#, svm_clf.predict([test]), voting_clf.predict([test])
```

Out[17]:
```
(array([[0.12773719, 0.87226281]]), 1, array([[0.5, 0.5]]), 0)
```

In [18]:
```
p_1 = (0.34 + 0.84912372) / 2
p_0 = (0.15087628 + 0.66) / 2
p_1, p_0
```

Out[18]:
```
(0.59456186, 0.40543814)
```

Soft × Hard voting

- Soft voting
    - If all classifiers are able to estimate class probabilities (i.e., they have a predict_proba() method)
    - 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
- Hard voting
    - Average the outcomes

Soft Voting

In [19]:
```
log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC(probability=True)

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_cl
f)],
    voting='hard')

voting_clf.fit(X_train, y_train)
```

Out[19]:
```
VotingClassifier(estimators=[('lr',
                              LogisticRegression(C=1.0, class_weig
ht=None,
                                                 dual=False, fit_i
ntercept=True,
                                                 intercept_scaling
=1,
                                                 l1_ratio=None, ma
x_iter=100,
                                                 multi_class='auto
',
                                                 n_jobs=None, pena
lty='l2',
                                                 random_state=Non
e,
                                                 solver='lbfgs', t
ol=0.0001,
                                                 verbose=0, warm_s
tart=False)),
                             ('rf',
                              RandomForestClassifier(bootstrap=Tru
e,
                                                     ccp_alpha=0.
0,
                                                     class_weight=
None,
                                                     cr...
                                                     oob_score=Fal
se,
                                                     random_state=
None,
                                                     verbose=0,
                                                     warm_start=Fa
lse)),
                             ('svc',
                              SVC(C=1.0, break_ties=False, cache_s
ize=200,
                                  class_weight=None, coef0=0.0,
                                  decision_function_shape='ovr', d
egree=3,
                                  gamma='scale', kernel='rbf', max
_iter=-1,
                                  probability=True, random_state=N
one,
                                  shrinking=True, tol=0.001, verbo
se=False))],
                 flatten_transform=True, n_jobs=None, voting='hard
',
                 weights=None)
```

Checking the accuracy of ensemble models

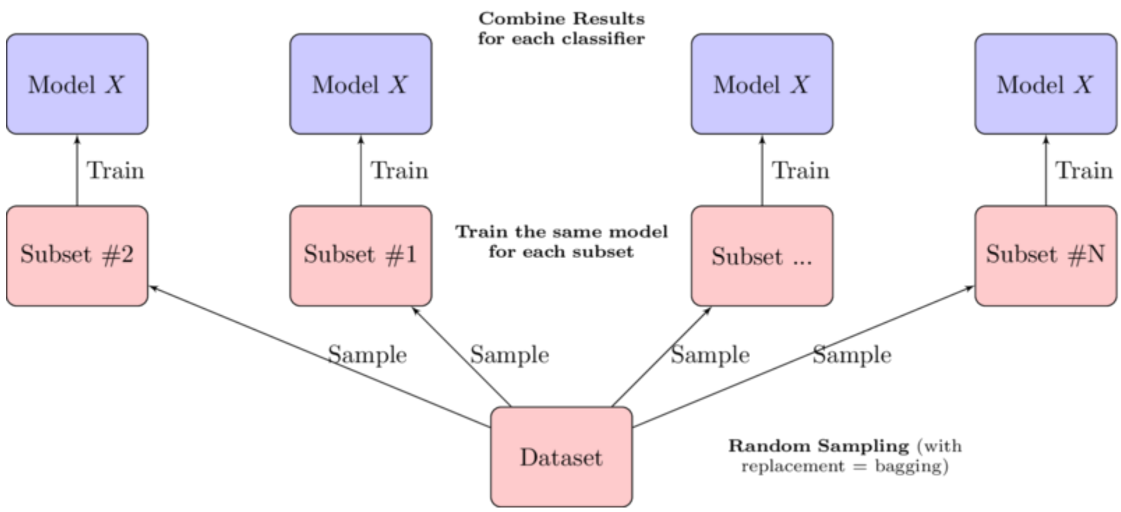
In [20]:
```
from sklearn.metrics import accuracy_score

from sklearn.metrics import accuracy_score
for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        print(clf.__class__.__name__, accuracy_score(y_test, y_pre
d))
```

```
LogisticRegression 0.8775
RandomForestClassifier 1.0
SVC 1.0
VotingClassifier 1.0
```

Bagging and Pasting

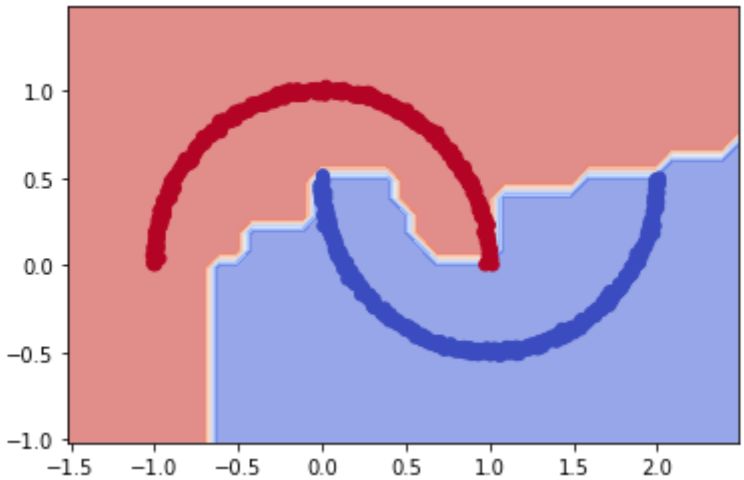
- One way to get a diverse set of classifiers is to use very different training algorithms (previous method).
- Another approach is to use the **same training algorithm** for every predictor, but to **train them on different random subsets** of the training set
- When sampling is performed with replacement, this method is called **bagging** (short for **bootstrap aggregating**)
- When sampling is performed without replacement, it is called **pasting**
- The aggregation function is typically the **statistical mode** (i.e., the most frequent prediction, just like a hard voting classifier)

In [21]:
```
%%tikz -s 800,600
% Bagging ensemble method
\input{bagging.tikz}
```



In [22]:
```
# Helper function to plot a decision boundary.
# If you don't fully understand this function don't worry, it just
generates the contour plot below.
def plot_decision_boundary(pred_func):
    # Set min and max values and give it some padding
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    h = 0.1
    # Generate a grid of points with distance h between them
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_mi
n, y_max, h))
    # Predict the function value for the whole gid
    Z = pred_func(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    # Plot the contour and training examples
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm_r, alpha=0.6)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm_r)
```

In [23]:
```
plot_decision_boundary(lambda x: clf.predict(x))
```



In [24]:
```
X_train
```

Out[24]:
```
array([[-0.96072314,  0.31069197],
       [-0.80813721,  0.60955648],
       [ 0.92397443,  0.37828277],
       ...,
       [ 0.09245089,  0.9869399 ],
       [ 0.78166873,  0.59646951],
       [ 0.75690617, -0.48029051]])
```

General Ensembling Methods

- Pasting draws random samples from the training data without replacement, whereas bagging samples with replacement.
- Random subspaces randomly sample from the features (that is, the columns) without replacement.
- Random patches train base estimators by randomly sampling both observations and features.

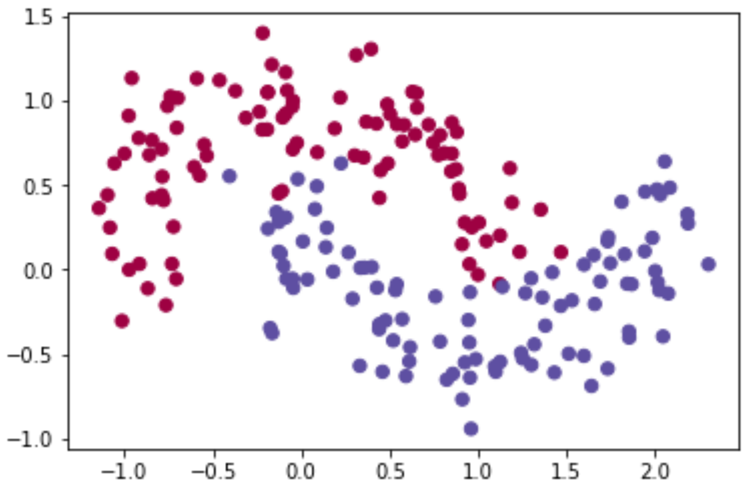
Bagging and Pasting in `Scikit-Learn`

- `Scikit-learn` offers `BaggingClassifier` and `BaggingRegressor`
- `bootstrap=False` parameter uses Pasting
- `n_jobs` parameter tells `Scikit-Learn` the number of CPU cores

Bagging and Pasting in `Scikit-Learn`

In [50]:
```
np.random.seed(1)
X, y = make_moons(200, noise=0.2)
# Split data into train and test datasets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=42
)
plt.scatter(X[:,0], X[:,1], s=40, c=y, cmap=plt.cm.Spectral)
```

Out[50]: <matplotlib.collections.PathCollection at 0x12ca80eb8>

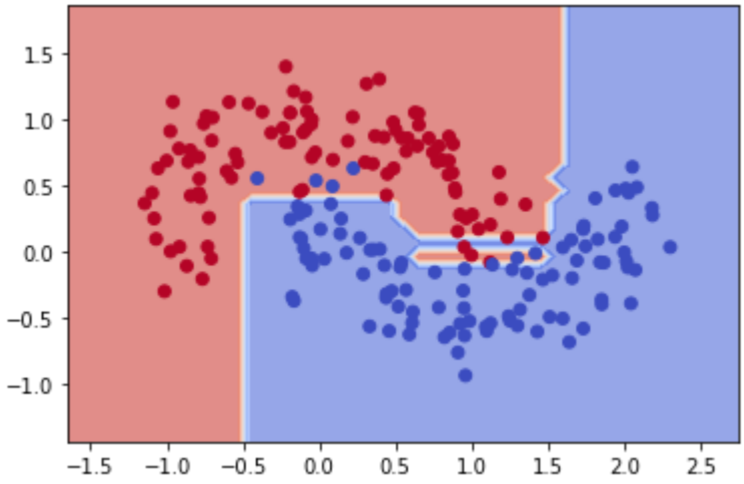


Bagging and Pasting in `Scikit-Learn`

In [52]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=10000,
    max_samples=110, bootstrap=True, n_jobs=-1)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

Bagging and Pasting in Scikit-Learn

In [53]:
```
plot_decision_boundary(lambda x: bag_clf.predict(x))
```

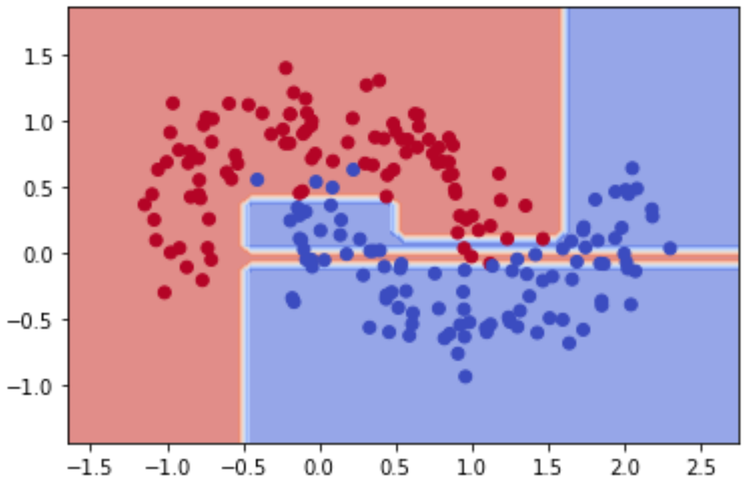


Set parameter `bootstrap=False` (Pasting Method)

In [54]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=1000,
    max_samples=100, bootstrap=False, n_jobs=-1)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

Bagging and Pasting in Scikit-Learn

In [55]:
```
plot_decision_boundary(lambda x: bag_clf.predict(x))
```



Out-of-Bag Evaluation

- Some instances may be sampled several times for any given predictor, while others may not be sampled at all
- `BaggingClassifier` samples m training instances with replacement `(bootstrap=True)`
- You can evaluate the ensemble itself by averaging out the oob (Out-of-Bag) evaluations of each predictor.

In [56]:
```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1, oob_score=True)
bag_clf.fit(X_train, y_train)
bag_clf.oob_score_
```

Out[56]: 0.9666666666666667

Out-of-Bag Evaluation

In [57]:
```
from sklearn.metrics import accuracy_score

y_pred = bag_clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[57]: 0.9375

Random Subspaces

- BaggingClassifier class supports sampling the features as well
- Hyperparameters: `max_features` and `bootstrap_features`
- Particularly useful when you are dealing with high-dimensional inputs (such as images)
- This is called Random Subspaces method

Class Exercise:

Check the accuracy of the bagging with the utilization of random subspaces Remember **`max_features`** and **`bootstrap_features`**

Random Forests

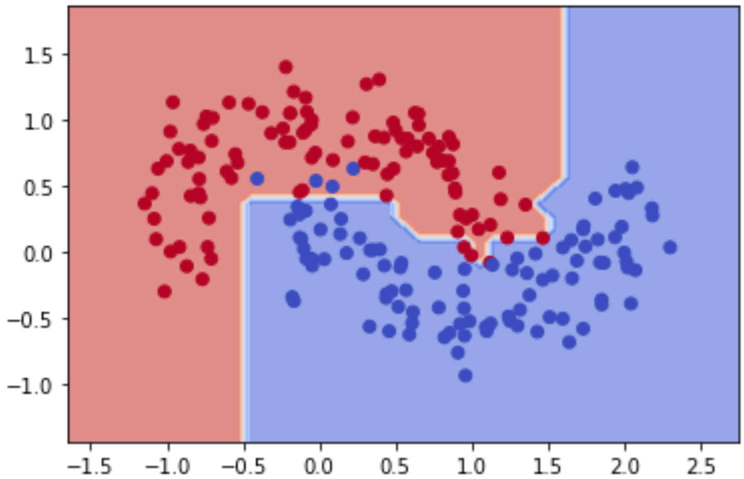
Random Forest is an ensemble of Decision Trees, with the following features:

- Trained via the bagging method (or sometimes pasting)
- Typically with `max_samples` set to the size of the training set
- Instead of programming an explicit `BaggingClassifier` , we use a `DecisionTreeClassifier` directly (less verbose)

Random Forests

In [59]:
```
from sklearn.ensemble import RandomForestClassifier

np.random.seed(0)
rnd_clf = RandomForestClassifier(n_estimators=1000, max_leaf_nodes=
20, n_jobs=-1)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

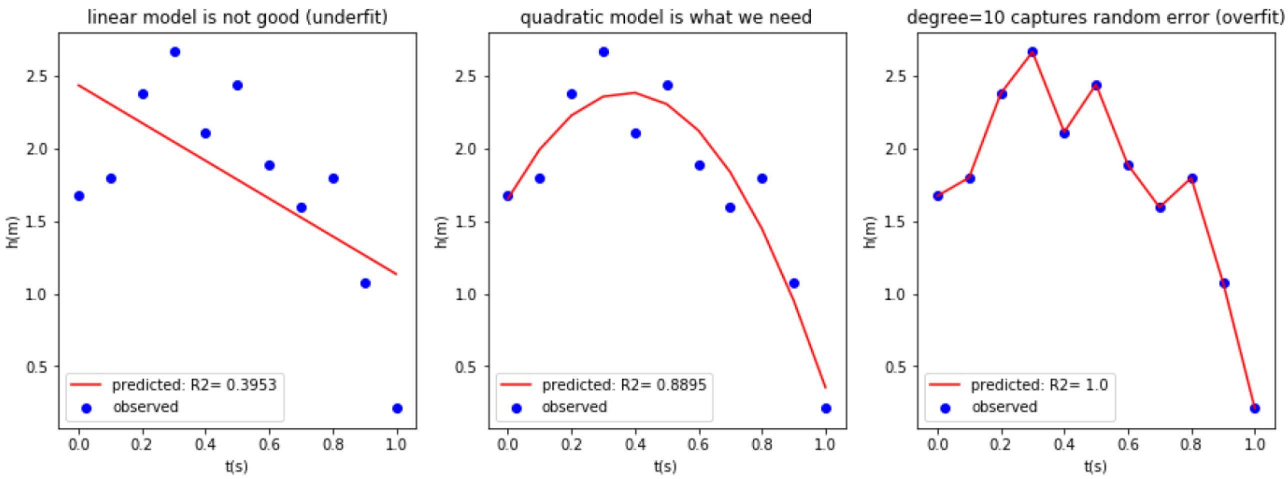
In [60]:
```
plot_decision_boundary(lambda x: rnd_clf.predict(x))
```



Important Note

- The Random Forest algorithm introduces extra randomness when growing trees
- Instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features.
- This results in a greater tree diversity, which (once again) trades a higher bias for a lower variance

Bias × Variance Trade-off



Reference: [https://medium.com/towards-artificial-intelligence/bias-variance-tradeoff-illustration-using-pylab-202943bf4c78 (https://medium.com/towards-artificial-intelligence/bias-variance-tradeoff-illustration-using-pylab-202943bf4c78)](https://medium.com/towards-artificial-intelligence/bias-variance-tradeoff-illustration-using-pylab-202943bf4c78)

Feature Importance

- It measures a feature's importance by looking at how much the tree nodes that use that feature reduce impurity on average
  - It 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
- Let's check the code

Feature Importance

In [61]:
```
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_impor
tances_):
    print(name, score)
```

```
sepal length (cm) 0.09499845753172727
sepal width (cm) 0.02416244911495009
petal length (cm) 0.439325164784758
petal width (cm) 0.4415139285685646
```

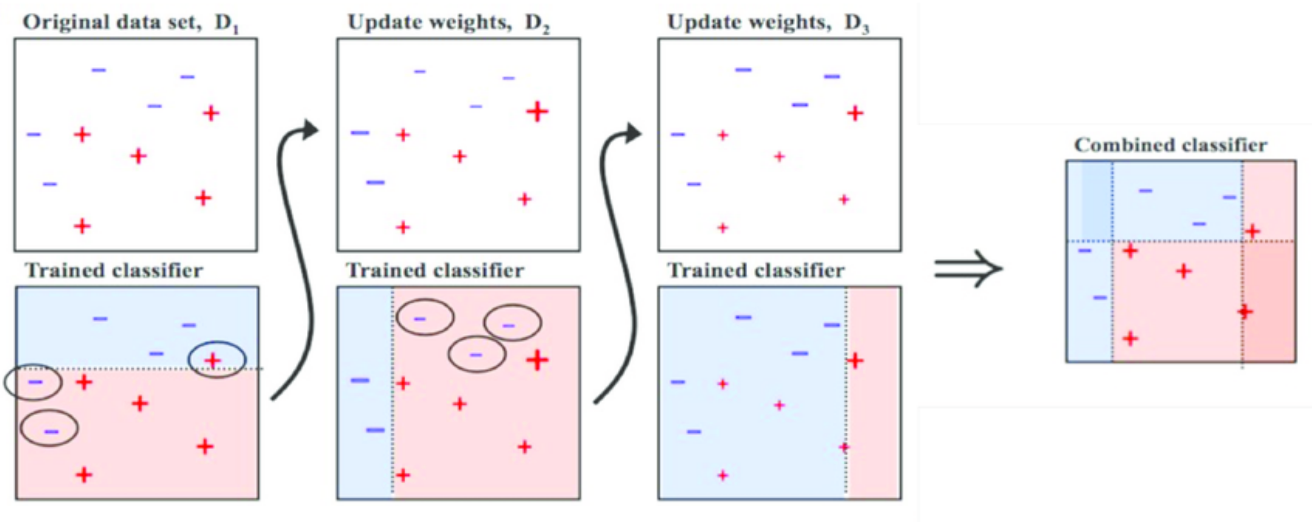
Boosting

- It refers to any Ensemble method that can combine several weak learners into a strong learner
- Adaptative Boosting
- Gradient Boosting

AdaBoosting

- One way for a new predictor to correct its predecessor is to pay a bit more attention to the training instances that the predecessor underfitted
- This results in new predictors fo- cusing more and more on the hard cases.
- The relative weight of misclassified training instances is then increased in new models

Adaboosting or Adaptatibe Boosting



https://www.researchgate.net/figure/Training-of-an-AdaBoost-classifier-The-first-classifier-trains-on-unweighted-data-then_fig3_306054843 (https://www.researchgate.net/figure/Training-of-an-AdaBoost-classifier-The-first-classifier-trains-on-unweighted-data-then_fig3_306054843) Marsh, Brendan. (2016). Multivariate Analysis of the Vector Boson Fusion Higgs Boson.

AdaBoosting

- There is one important drawback to this sequential learning technique
  - It cannot be parallelized, since 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.

Gradient Boosting

- It works by sequentialling adding predictors to an ensenmble
- Each new model corrects its predecessor
- Let's see the code.

Train the first DecisionTreeRegressor

In [35]:
```
from sklearn.tree import DecisionTreeRegressor
tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

Out[35]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=2,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_spli
t=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='depre
cated',
                      random_state=None, splitter='best')
```

Now train a second DecisionTreeRegressor on the residual errors made by the first predictor:

In [36]:
```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```

Out[36]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=2,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_spli
t=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='depre
cated',
                      random_state=None, splitter='best')
```

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

In [37]:
```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

Out[37]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=2,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_spli
t=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='depre
cated',
                      random_state=None, splitter='best')
```

In [41]:
```
y_pred = sum(tree.predict(X[:10]) for tree in (tree_reg1, tree_reg
2, tree_reg3))
y_pred
```

Out[41]:
```
array([ 0.01587839,  0.01587839,  0.01587839,  1.02800279,  0.8562
6373,
        0.83817587, -0.04448161,  0.85626373,  0.01587839,  0.8562
6373])
```

In [42]:
```
y[:10]
```

Out[42]:
```
array([0, 0, 0, 1, 1, 1, 0, 0, 0, 1])
```

It can be simply

In [45]:
```
from sklearn.ensemble import GradientBoostingRegressor
gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learn
ing_rate=1.0)
gbrt.fit(X, y)
```

Out[45]:
```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='fri
edman_mse',
                          init=None, learning_rate=1.0, loss='ls',
max_depth=2,
                          max_features=None, max_leaf_nodes=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_estimato
rs=3,
                          n_iter_no_change=None, presort='deprecat
ed',
                          random_state=None, subsample=1.0, tol=0.
0001,
                          validation_fraction=0.1, verbose=0, warm
_start=False)
```

In [48]:
```
gbrt.predict(X[:10]), y_pred
```

Out[48]:
```
(array([ 0.01587839,  0.01587839,  0.01587839,  1.02800279,  0.856
26373,
         0.83817587, -0.04448161,  0.85626373,  0.01587839,  0.856
26373]),
 array([ 0.01587839,  0.01587839,  0.01587839,  1.02800279,  0.856
26373,
         0.83817587, -0.04448161,  0.85626373,  0.01587839,  0.856
26373]))
```

Further Reading

- Decision Trees and random forests: https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991 (https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991)
- Boosting Material: https://medium.com/diogo-menezes-borges/boosting-with-adaboost-and-gradient-boosting-9cbab2a1af81 (https://medium.com/diogo-menezes-borges/boosting-with-adaboost-and-gradient-boosting-9cbab2a1af81)
- Ensemble methods: https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f (https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f)

In [ ]: