Machine Learning

| Boosting

Prof. Dr. Mostafa Elhosseini

https://youtube.com/ElhosseiniAcademy

Agenda

Boosting

- **AdaBoost** (Adaptive Boosting)
- Gradient Boosting
  - GBM: Vanilla gradient boosting implementation
  - XGBoost (Extreme Gradient Boosting)
  - LightGBM: Focuses on efficiency with large datasets and high-dimensional data using leaf-wise tree growth.
  - CatBoost: Specializes in categorical feature handling without explicit encoding.

Boosting

- Boosting (originally called **hypothesis boosting**) is an ensemble machine learning technique that aims to create a strong predictive model by combining multiple weak learners.

- Unlike **bagging**, which **reduces variance** by averaging multiple models, boosting focuses on **reducing bias** by training models **sequentially**, each trying to **correct the errors of its predecessor**.

- **Reduces bias** by focusing on **misclassified instances**

Boosting

- Start by **training a random model**, which is the first weak learner. Evaluate it on the entire dataset.
- **Shrink** the points that have good predictions and **enlarge** the points that have poor predictions.
- Train a **second weak learner** on this **modified dataset**. We continue in this fashion until we build several models.

Boosting

- The way to combine them into a **strong learner** is the same way as with bagging, namely, by **voting or by averaging the predictions** of the weak learner.

- More specifically, if the learners are **classifiers**, the strong learner predicts the **most common class** predicted by the weak learners (thus the term voting), and if there are **ties**, by choosing **randomly** among them. If the learners are **regressors**, the strong learner predicts the **average** of the predictions given by the weak learners.

AdaBoost

- Pay a bit **more attention** to the training instances that the predecessor **underfit**. This results in new predictors focusing more and more on the hard cases.

- This is the technique used by **AdaBoost**.

- There are many boosting methods available, but by far the most popular are **AdaBoost** (short for adaptive boosting) and gradient boosting.

AdaBoost

- AdaBoost is short for **ada**ptive **boost**ing, and it works for regression and classification
- In AdaBoost, like in random forests, each **weak learner is a decision tree of depth 1**. Unlike random forests, each weak learner is **trained on the whole dataset**, rather than on a portion of it.
- The only caveat is that after each weak learner is trained, we modify the dataset by **enlarging the points that have been incorrectly classified**, so that future weak learners **pay more attention** to these. In a nutshell, AdaBoost works as follows

AdaBoost

- When training an AdaBoost classifier, the algorithm **first trains a base classifier** (such as a decision tree) and uses it to make predictions on the training set.
- The algorithm then **increases the relative weight of misclassified** training instances.

- Then it **trains a second classifier**, using the updated weights, and again makes predictions on the training set, updates the instance weights, and so on

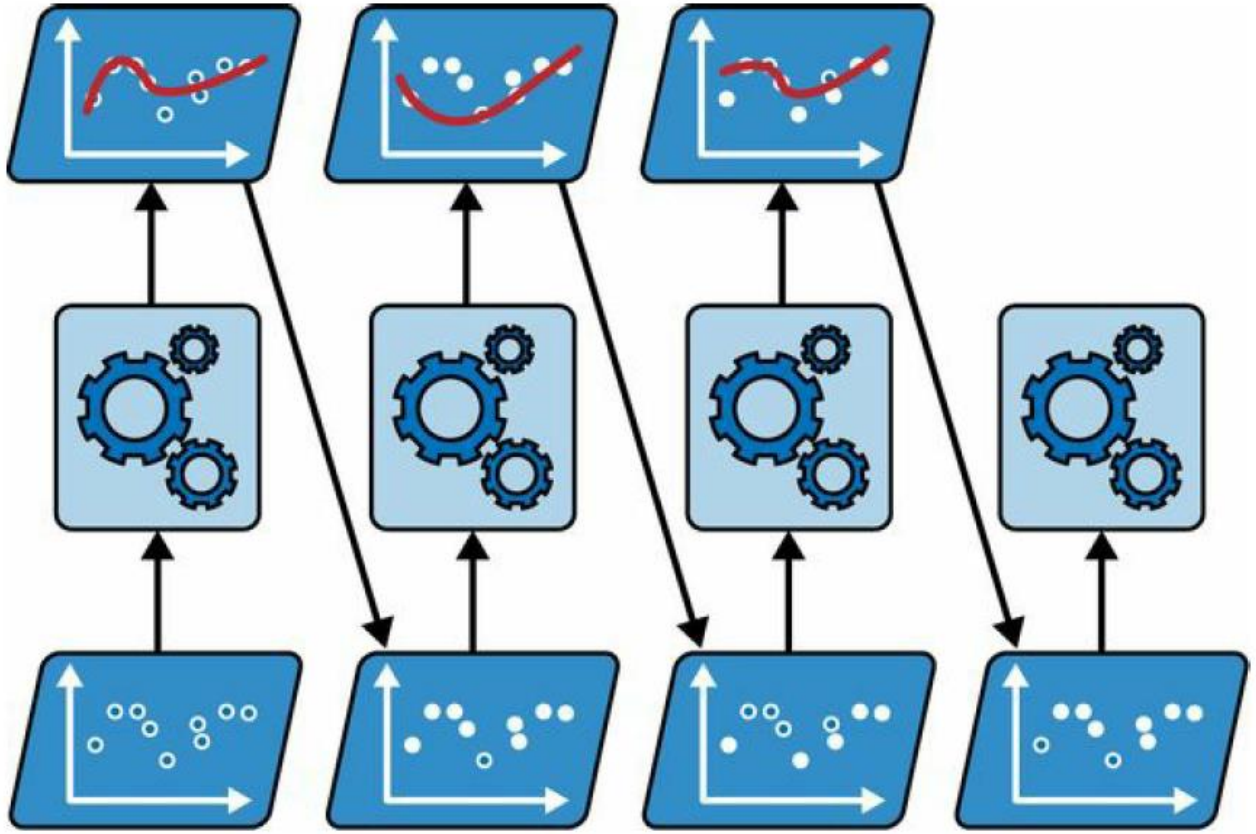
AdaBoost

Pseudocode for training an AdaBoost model

- Train the first weak learner on the first dataset.
- Repeat the following step for each new weak learner:
  - After a weak learner is trained, the points are modified as follows
    - The points that are incorrectly classified are enlarged.
  - Train a new weak learner on this modified dataset.

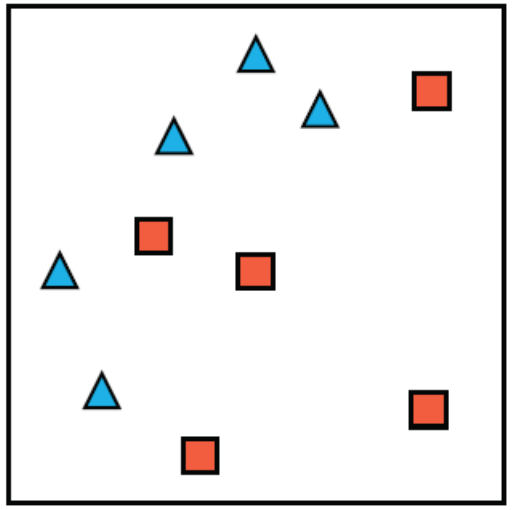
AdaBoost

- AdaBoost classifier



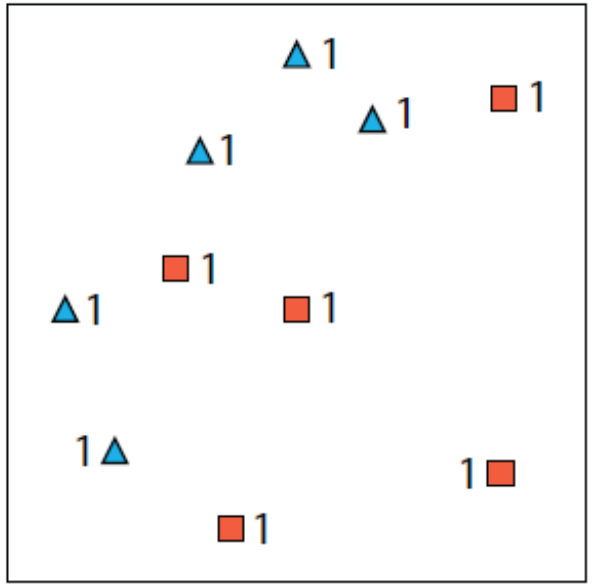
AdaBoost

- The dataset we use has two classes (triangles and squares)



AdaBoost

- The first step is to **assign to each of the points a weight of 1**.

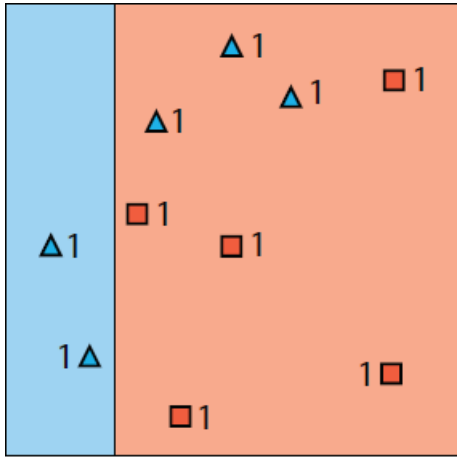


Add a weight of 1
to every point.

AdaBoost

- Next, we build a weak learner on this dataset. Recall that the weak learners are decision trees of depth 1. A decision tree of depth 1 corresponds to the horizontal or vertical line that best splits the points

- correctly classifies the two triangles to its left and the five squares to its right, and incorrectly classifies the three triangles to its right



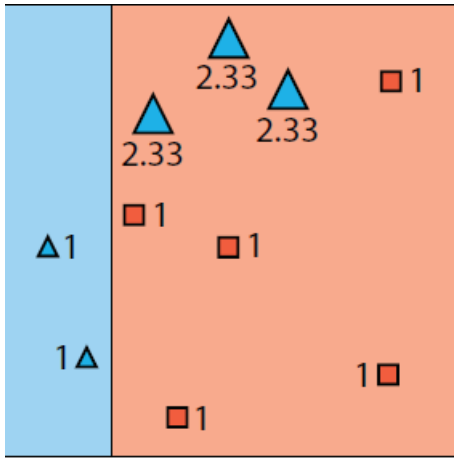
Fit a weak learner.
Correct: 7
Incorrect: 3

AdaBoost

- The next step is to **enlarge the three incorrectly** classified points to give them **more importance** under the eyes of future weak learners.
- To enlarge them, recall that each point initially has a weight of 1
- We define the **rescaling factor** of this weak learner as the number of correctly classified points divided by the number of incorrectly classified points

AdaBoost

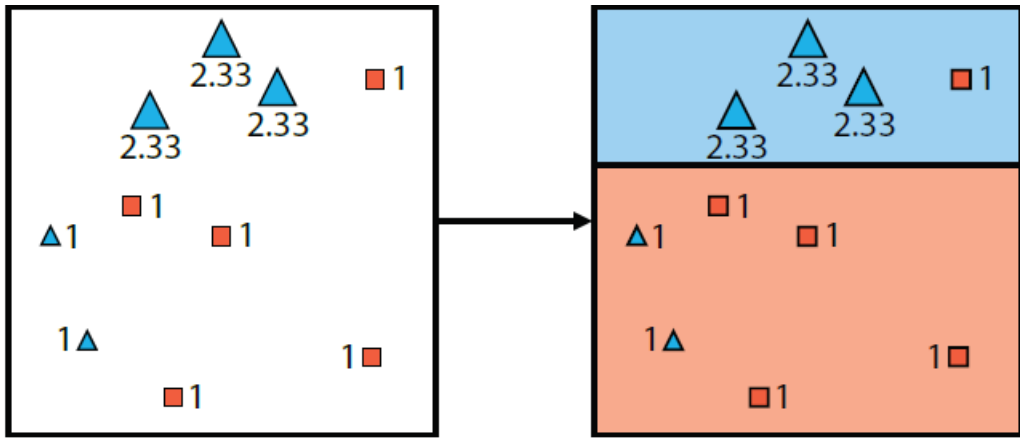
- the rescaling factor is = 7/3 = 2.33. We proceed to rescale every misclassified point by this rescaling factor



Rescale misclassified points by 7/3.

AdaBoost

- Build the next one in the same manner



The rescaled dataset

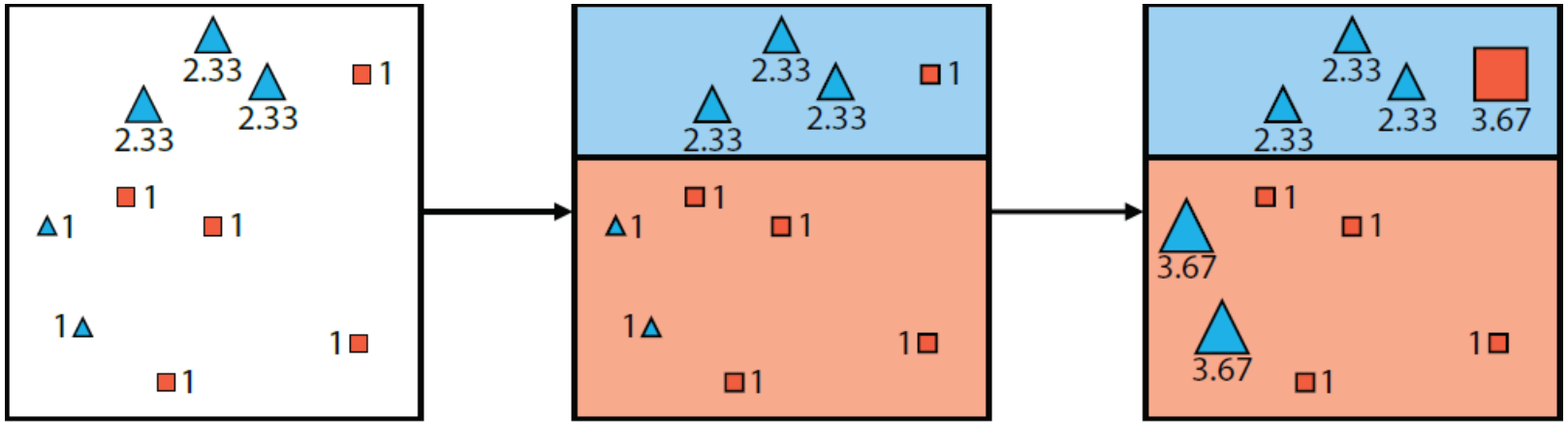
Fit a weak learner.
Sum of correct: 11
Sum of incorrect: 3

AdaBoost

- The rescaling factor is the ratio between the sum of the weights of the correctly classified points and the sum of the weights of the incorrectly classified points.
- The first term is 2.33 + 2.33 + 2.33 + 1 + 1 + 1 + 1 = 11, and the second is 1 + 1 + 1 = 3. Thus, the rescaling factor is 11/3 = 3.67
- We proceed to multiply the weights of the three misclassified points by this factor of 3.67

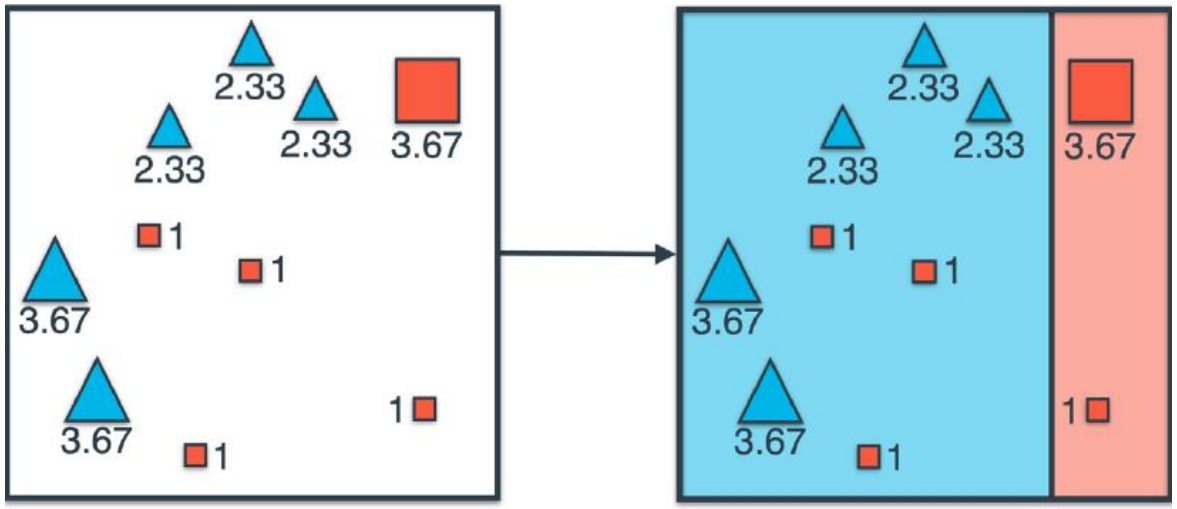
AdaBoost

- We proceed to multiply the weights of the three misclassified points by this factor of 3.67



AdaBoost

- We continue in this fashion until we've built as many weak learners as we want. we build only three weak learners. The third weak learner is a vertical line



AdaBoost

- This is how we build the weak learners. Now, we need to **combine them into a strong learner**. This is similar to what we did with random forests, but using **a little more math**

- The idea is to **get the classifiers to vote**, just as they did in the random forest classifier, but this time, **good learners get more of a say than poor learners**. In the event that a classifier is really bad, then its vote will actually be negative

AdaBoost

- To understand this, imagine we have **three friends**:
  - Truthful Teresa,
  - Unpredictable Umbert, and
  - Lying Lenny.
- Truthful Teresa almost always **tells the truth**, Lying Lenny almost always **lies**, and **Unpredictable** Umbert says the truth roughly half of the time and lies the other half.

- Out of these three friends, which is the least useful one?

AdaBoost

- Truthful Teresa is very reliable
- Lying Lenny - almost always **lies** when we ask him a yes-or-no question, we simply **take as truth the opposite** of what he tells us.
- Unpredictable Umbert serves us no purpose if we have no idea whether he’s telling the truth or lying

AdaBoost

- Now imagine that our **three friends are weak learners** trained in a dataset with two classes.

- Truthful Teresa is a classifier with very high accuracy, Lying Lenny is one with very low accuracy, and Unpredictable Umbert is one with an accuracy close to 50%.

- We want to build a strong learner where the prediction is obtained by a **weighted vote** from the three weak learners

AdaBoost

- The Truthful Teresa classifier gets a high positive score.
- The Unpredictable Umbert classifier gets a score close to zero.
- The Lying Lenny classifier gets a high negative score.

AdaBoost

In other words, the score of a weak learner is a number:

- Is positive when the accuracy of the learner is greater than 0.5
- Is 0 when the accuracy of the model is 0.5
- Is negative when the accuracy of the learner is smaller than 0.5
- Is a large positive number when the accuracy of the learner is close to 1
- Is a large negative number when the accuracy of the learner is close to 0

AdaBoost

- To come up with a good score for a weak learner that satisfies the properties mentioned, we use a popular concept in probability called the **logit**, or **log-odds**

Probability, odds, and log-odds

- **Probability** of this outcome is the number of times it occurred divided by the total number of times we ran the experiment.

- The **odds** of this outcome are the number of times it occurred divided by the number of times it didn't occur.

- For example, the probability of obtaining 1 when we roll a die is 1/6 , but the odds are 1/5

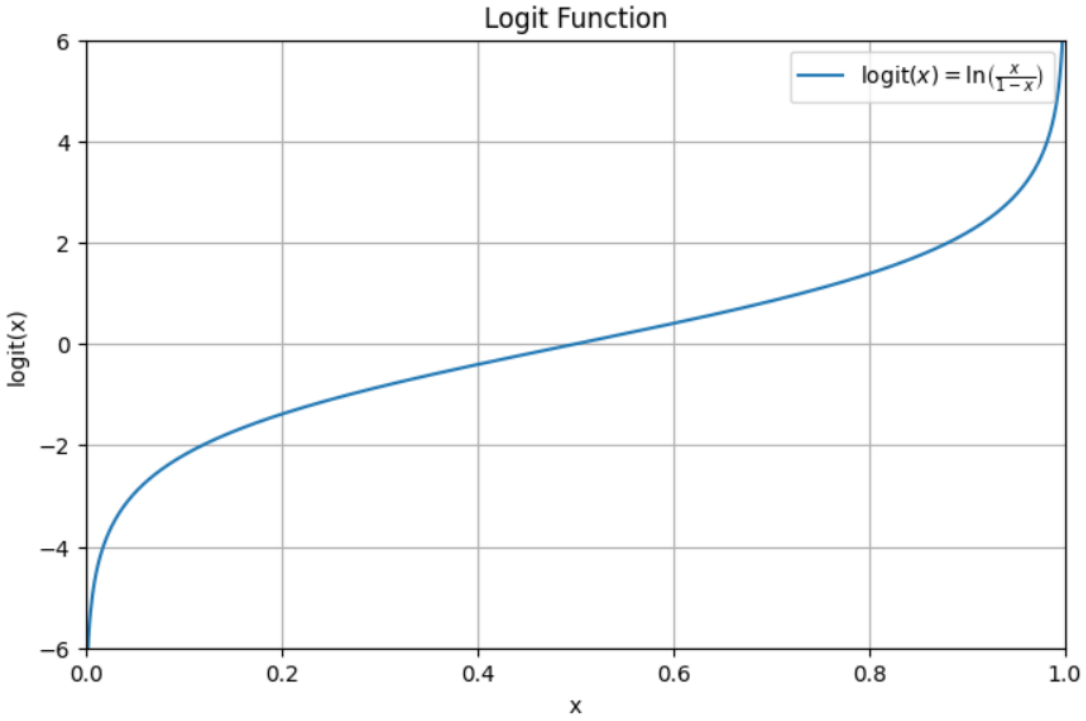
Probability, odds, and log-odds

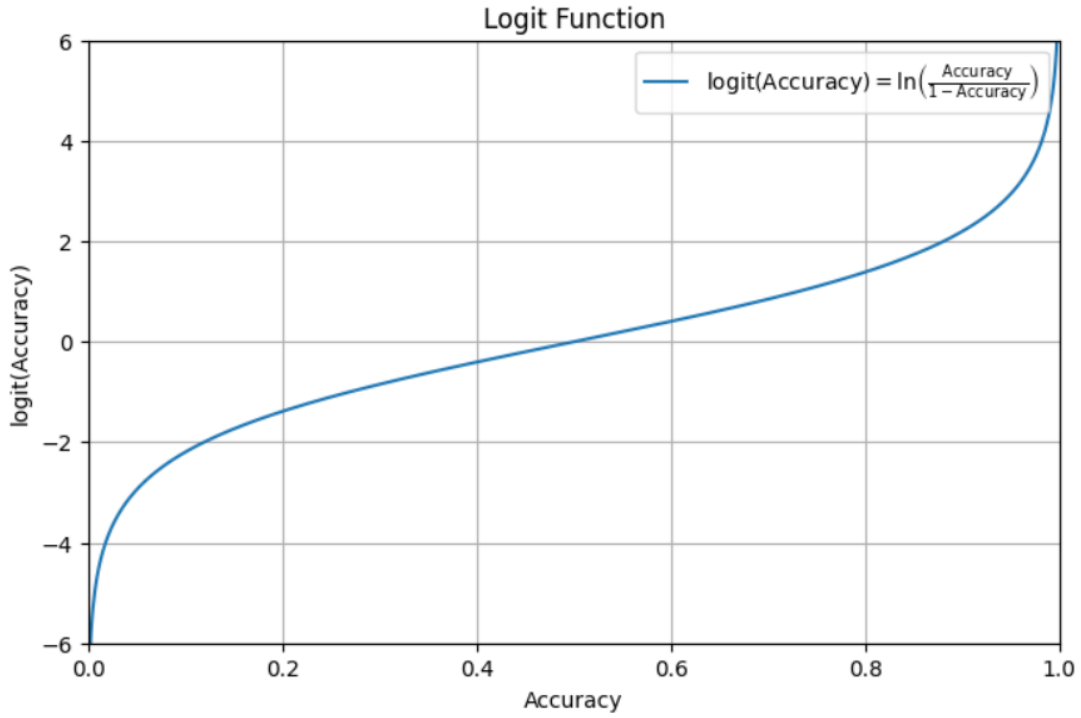
- If the probability of an event is x, then the odds are $\frac{x}{1-x}$.
- Notice that because the probability is a number between 0 and 1, then the odds are a number between 0 and ∞.

- We are looking for a function that satisfies properties
- The odds function is close, but not quite there, because it **outputs only positive values**.
- The way to turn the odds into a function that satisfies properties above is by **taking the logarithm**.

Probability, odds, and log-odds

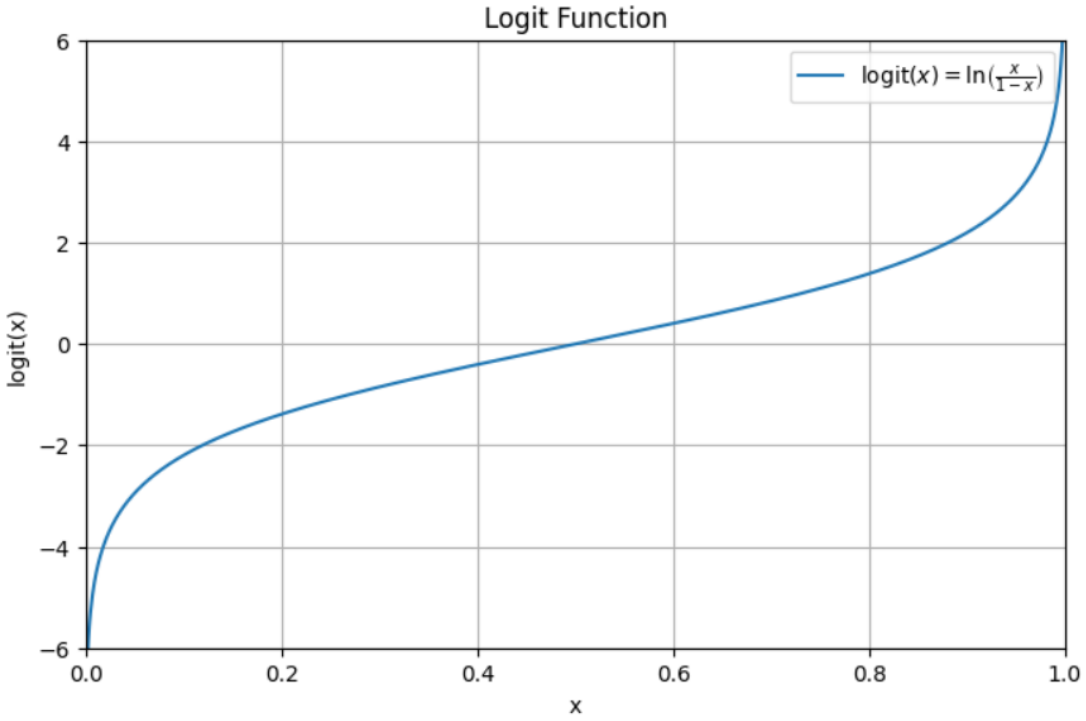
- Log-odds, also called the logit, defined as follows:
- $Log - odd(x) = \ln \frac{x}{1-x}$

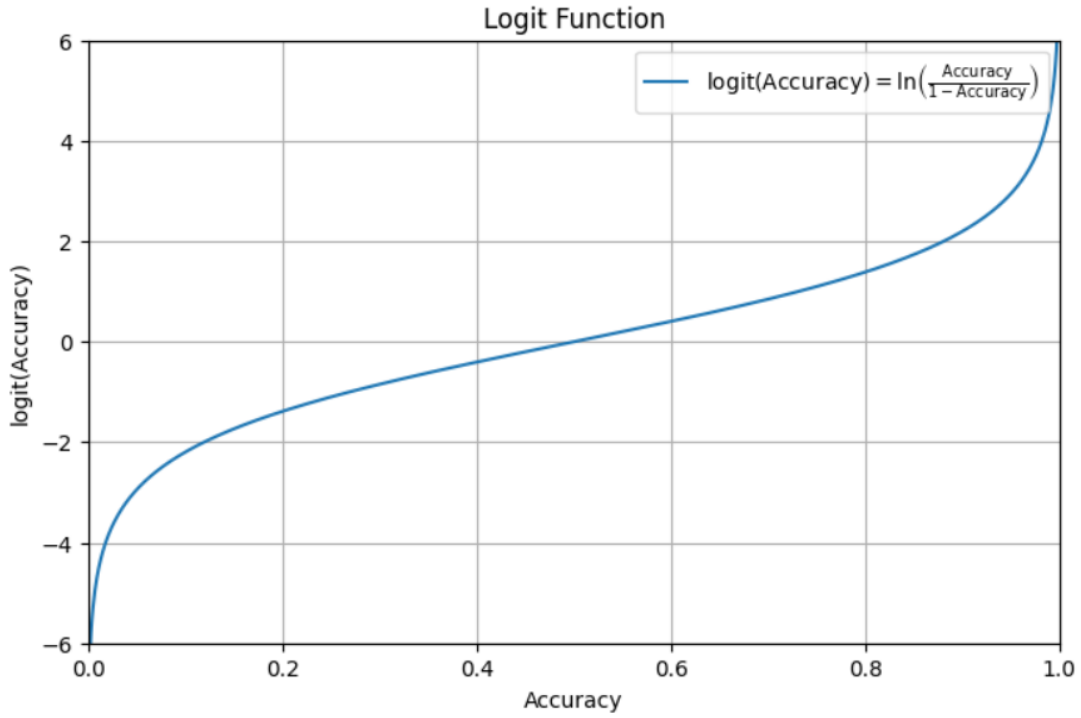




Probability, odds, and log-odds

- Log-odds, also called the logit, defined as follows:
- $Log - odd(x) = \ln \frac{x}{1-x}$

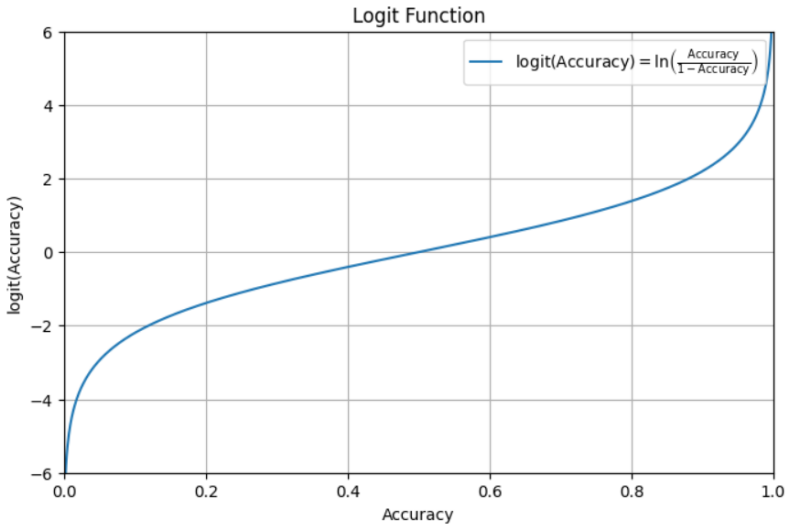




Probability, odds, and log-odds

- $Log - odd(x) = \ln \frac{x}{1-x}$
- Models with high accuracy have high positive scores, models with low accuracy have high negative scores, and models with accuracy close to 0.5 have scores close to 0.

Accuracy	Log-odds (score of the weak learner)
0.01	–4.595
0.1	–2.197
0.2	–1.386
0.5	0
0.8	1.386
0.9	2.197
0.99	4.595



Combining the classifiers

- $Log - odd(x) = \ln \frac{x}{1-x}$
- Now that we've settled on the log-odds as the way to define the scores for all the weak learners, we can proceed to join them to build the strong learner

- **Weak learner 1:**
- Accuracy: 7/10
- Score = $\ln\left(\frac{7}{3}\right) = 0.847$

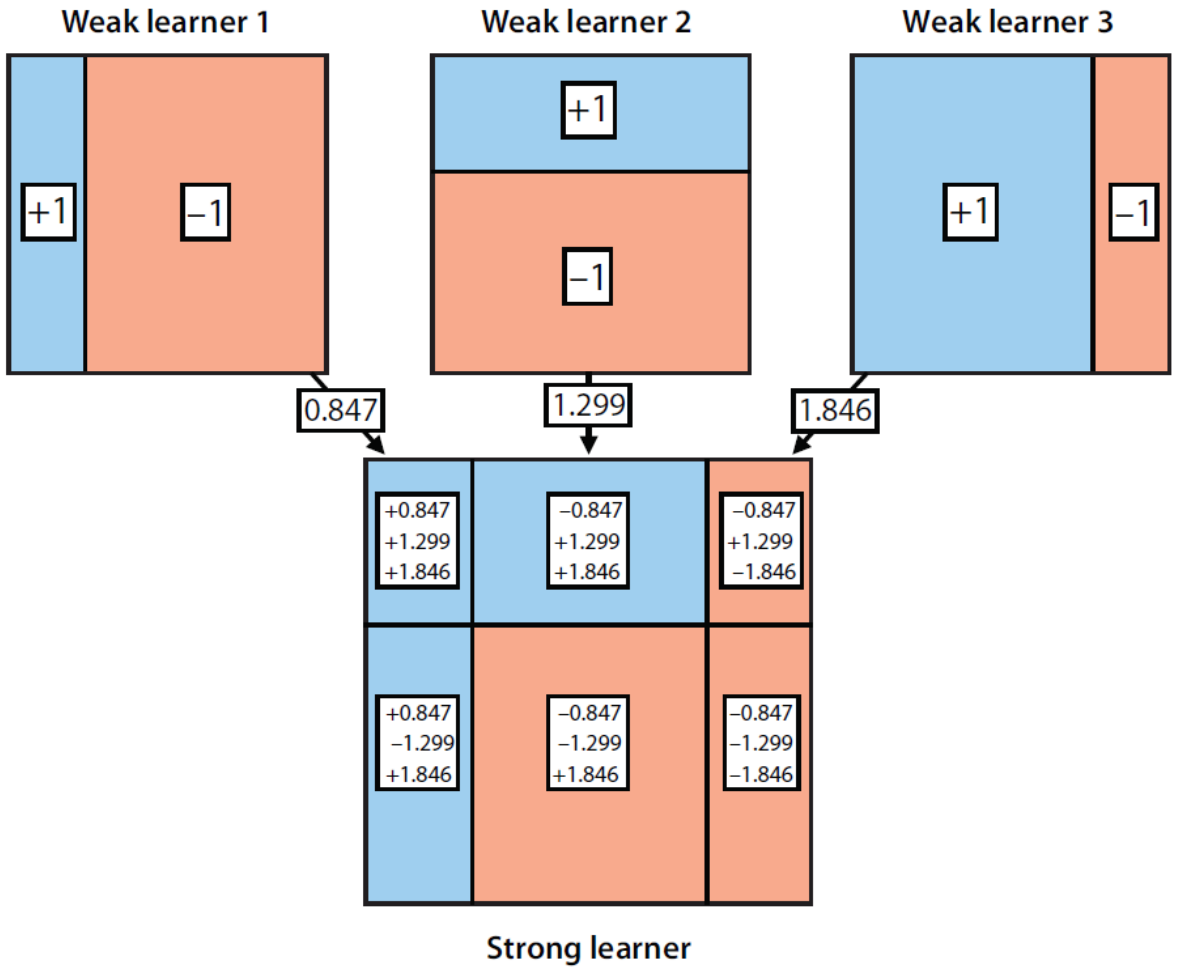
- **Weak learner 2:**
- Accuracy: 11/14
- Score = $\ln\left(\frac{11}{3}\right) = 1.299$

- **Weak learner 3:**
- Accuracy: 19/22
- Score = $\ln\left(\frac{19}{3}\right) = 1.846$

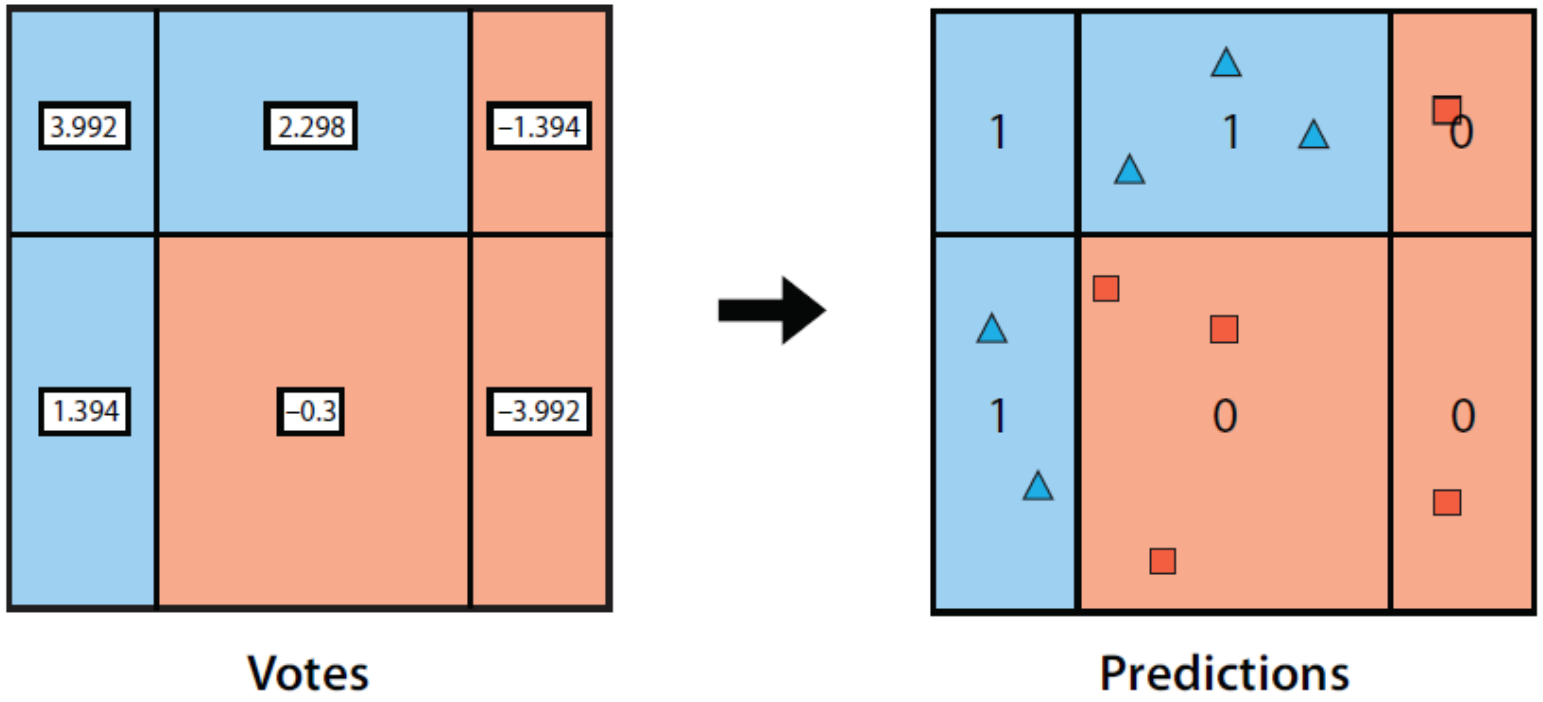
Combining the classifiers

- The prediction that the strong learner makes is obtained by the **weighted vote** of the weak classifiers, where each classifier's vote is its score.
- A simple way to see this is to change the predictions of the weak learners from 0 and 1 **to –1 and 1**, **multiplying each prediction by the score of the weak learner, and adding them**.
- If the resulting prediction is greater than or equal to zero, then the strong learner predicts a 1, and if it is negative, then it predicts a 0.

Combining the classifiers

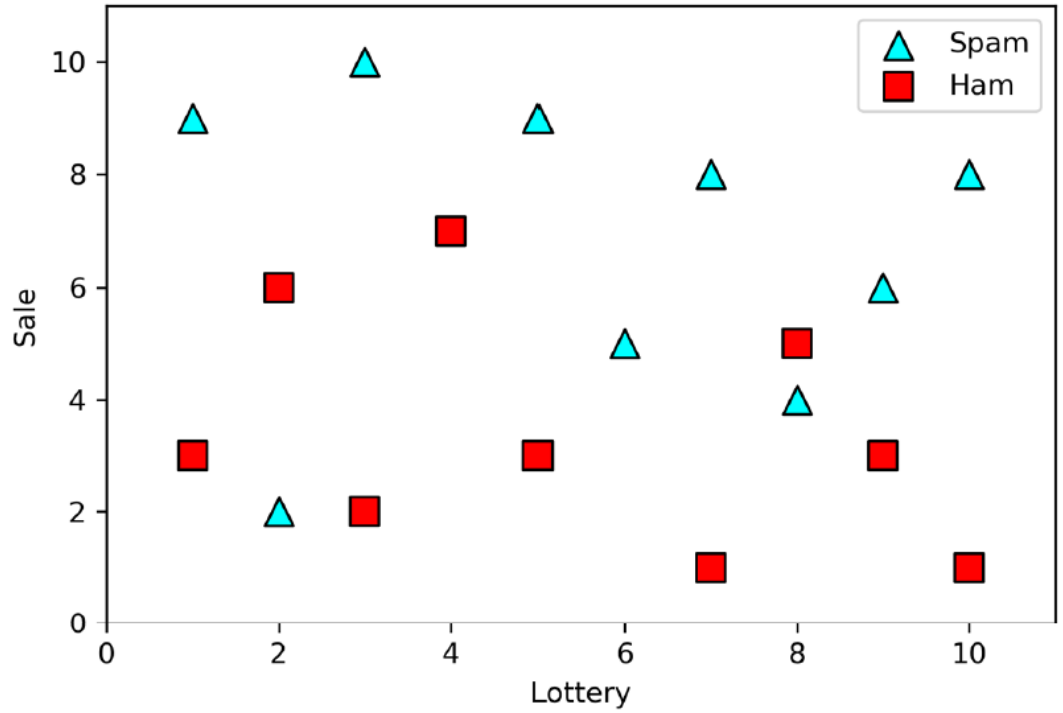


Combining the classifiers



Coding AdaBoost in Scikit-Learn

- how to use Scikit-Learn to train an AdaBoost model
- spam email dataset



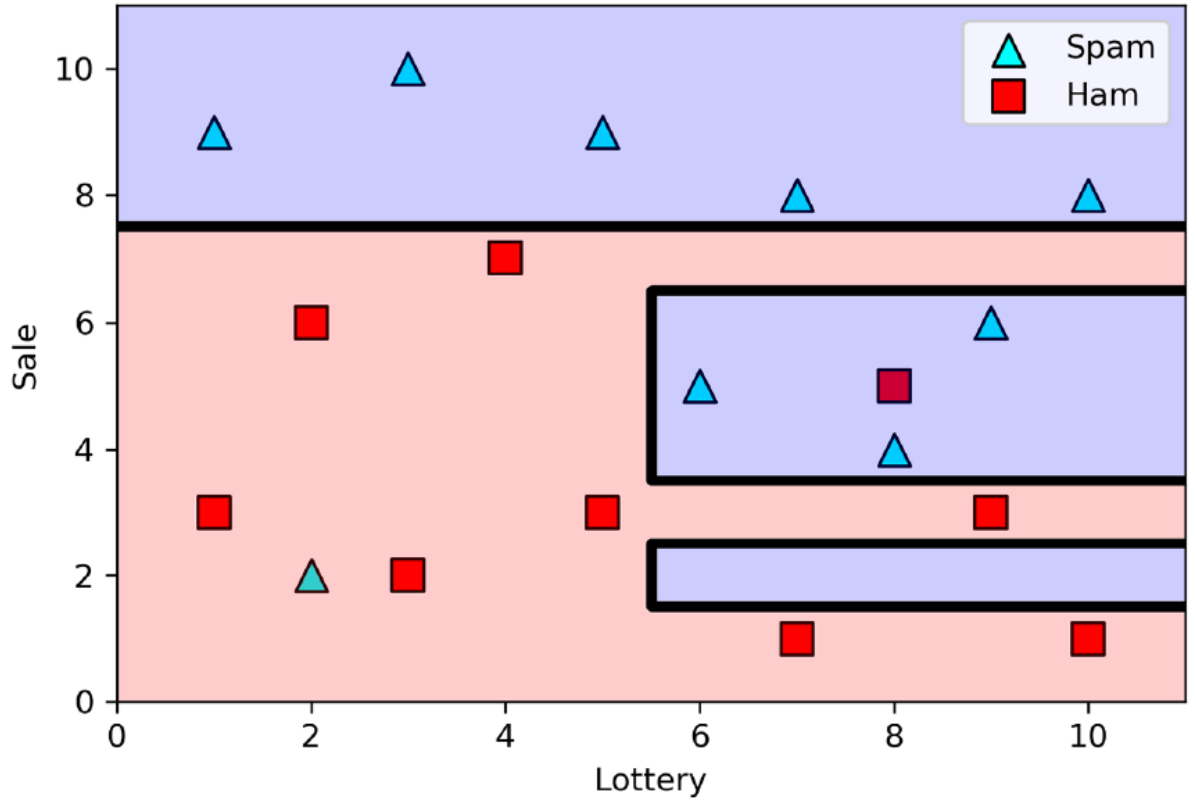
Coding AdaBoost in Scikit-Learn

- The training is done using the AdaBoostClassifier

```
from sklearn.ensemble import AdaBoostClassifier
# Set the random_state so that we always get the same results
adaboost_classifier = AdaBoostClassifier(random_state=0, n_estimators=6)
adaboost_classifier.fit(features, labels)
adaboost_classifier.score(features, labels)
```

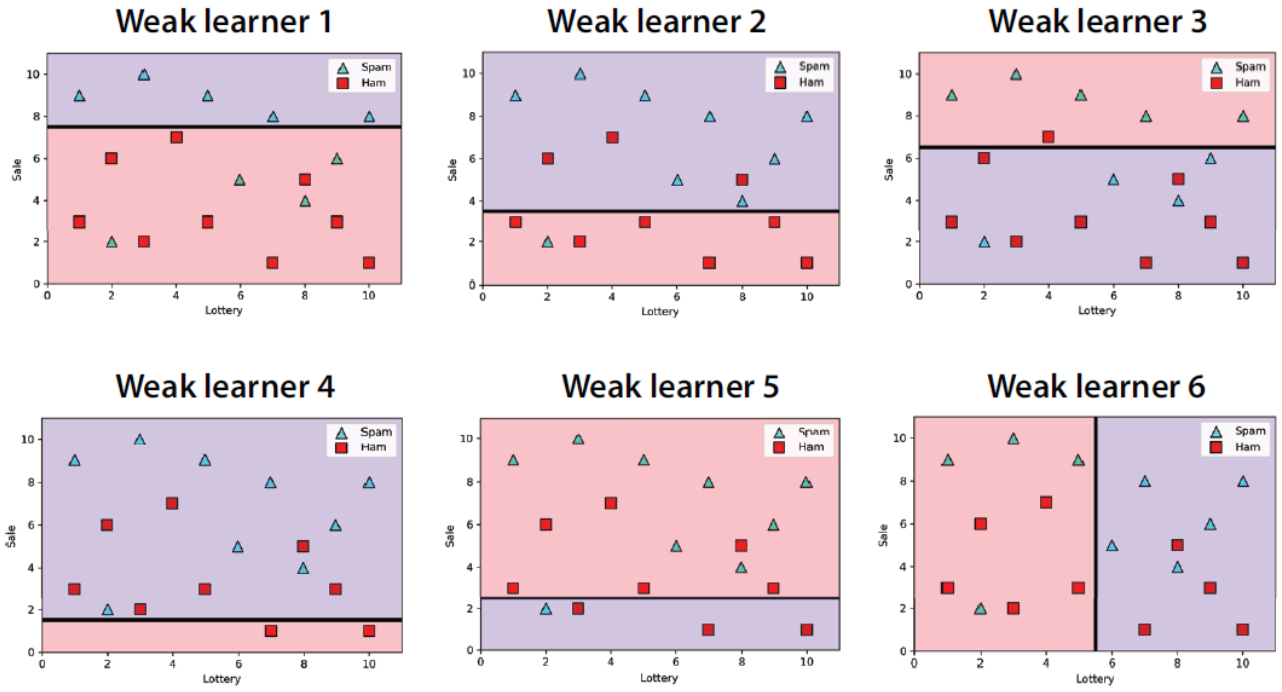
Coding AdaBoost in Scikit-Learn

- The boundary of the resulting model



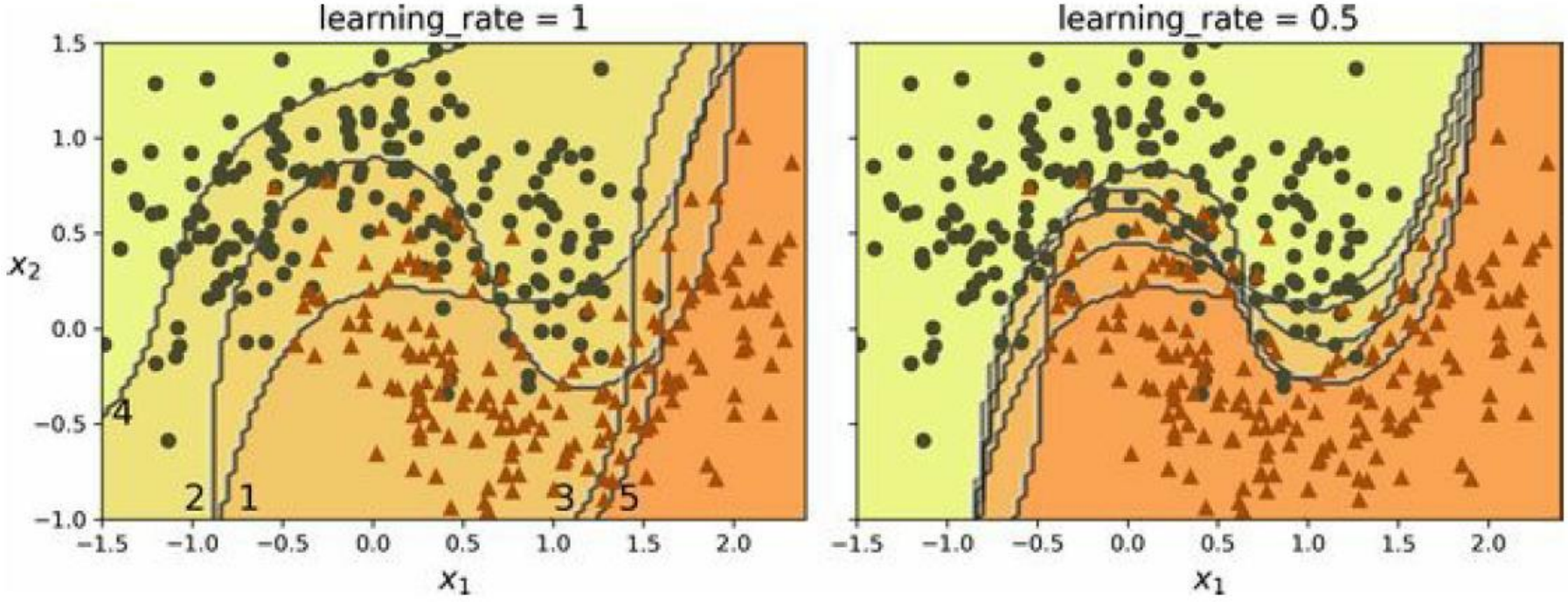
Coding AdaBoost in Scikit-Learn

- six weak learners and their scores. Note that the strong learner in is obtained by assigning a score of 1 to each of the weak learners in and letting them vote.



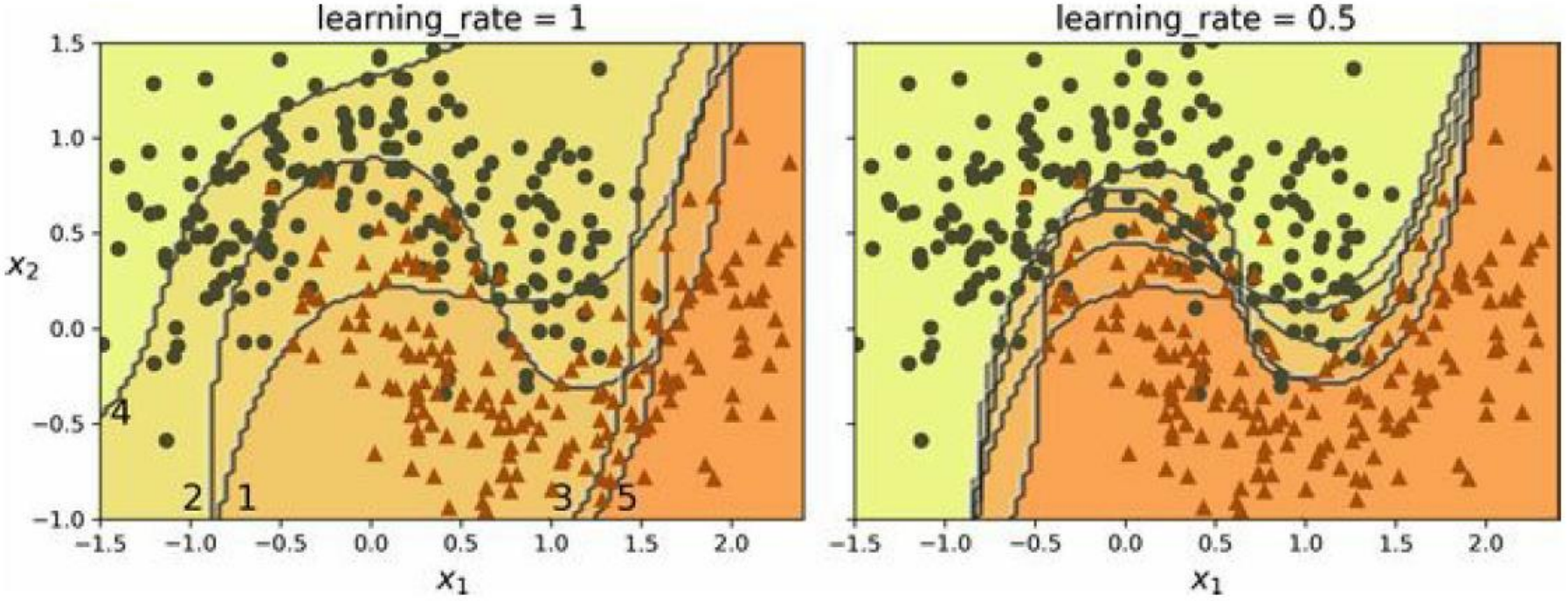
AdaBoost

- Five consecutive predictors on the moons dataset
- Each predictor is a highly regularized SVM classifier with an RBF kernel



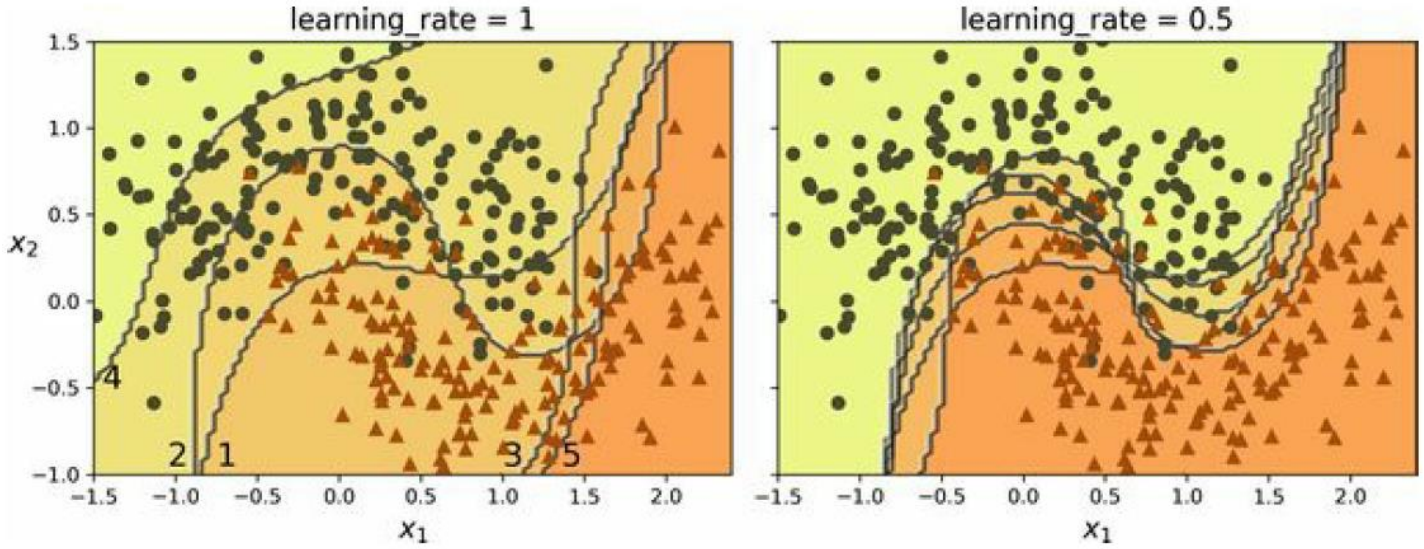
AdaBoost

- The first classifier gets many instances wrong, so their weights get boosted. The second classifier therefore does a better job on these instances, and so on



AdaBoost

- As you can see, this sequential learning technique has some similarities with gradient descent, except that instead of tweaking a single predictor's parameters to minimize a cost function, AdaBoost adds predictors to the ensemble, gradually making it better.



AdaBoost

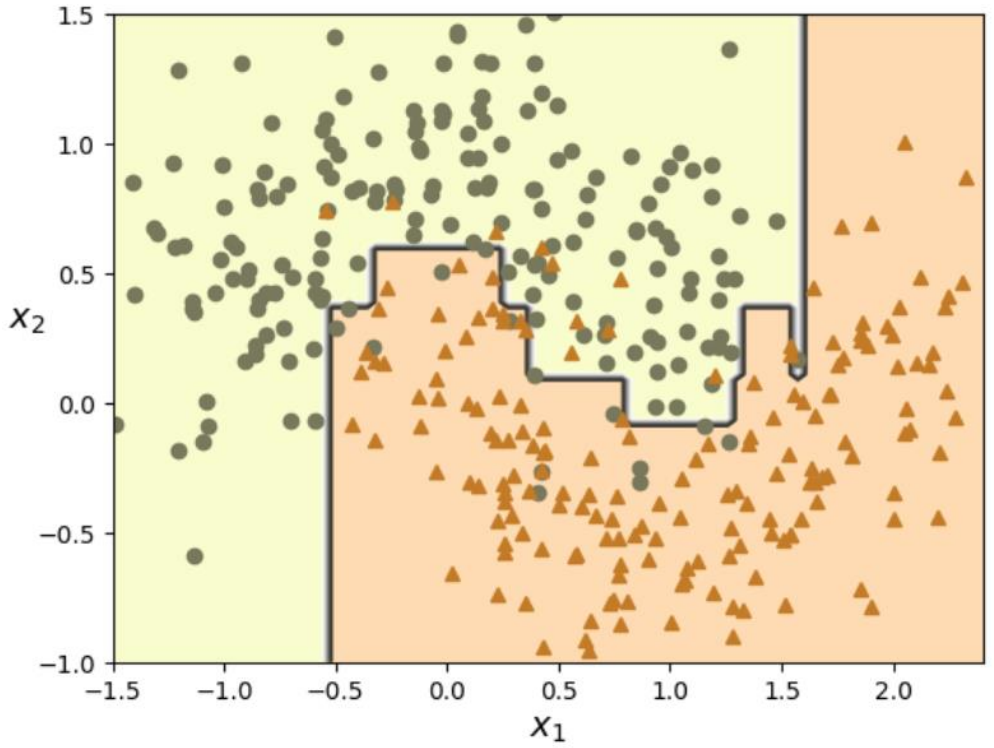
- 30 **decision stumps**
- A decision stump is a decision tree with max_depth=1 — in other words, a tree composed of a single decision node plus two leaf nodes

```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=30,
    learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
```

AdaBoost

- If your AdaBoost ensemble is overfitting the training set, you can try reducing the number of estimators or more strongly regularizing the base estimator.



Gradient Boosting

- Gradient boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor.

- However, **instead of tweaking the instance weights at every iteration** like AdaBoost does, this method tries to **fit the new predictor to the residual errors** made by the previous predictor