Ensemble methods: AdaBoost

Glenn Bruns
CSUMB

Learning outcomes

After this lecture you should be able to:

1. Define ‘random forest’, and use random forests in Scikit-Learn
2. Define the concepts of ‘weak learner’ and ‘boosting’
3. Describe the operation of AdaBoost in detail
4. Use AdaBoost in Scikit-Learn

Random Forests

Random Forest = bagging + decision trees (roughly)

- usually the decision trees are trained with data sets as large as the full training set
- when creating a node in the decision tree, only a random subset of the features are considered
  - a subset is randomly chosen at each node
  - the goal is greater diversity among the classifiers

In random forests, diversity both in predictors and in data

Decision trees and feature importance

In a decision tree, important features tend to appear near the root.

One way to measure feature importance is to see the depth at which it (first) appears in the tree.

sklearn's RandomForestClassifier outputs a variable importance score based on average depth across trees.

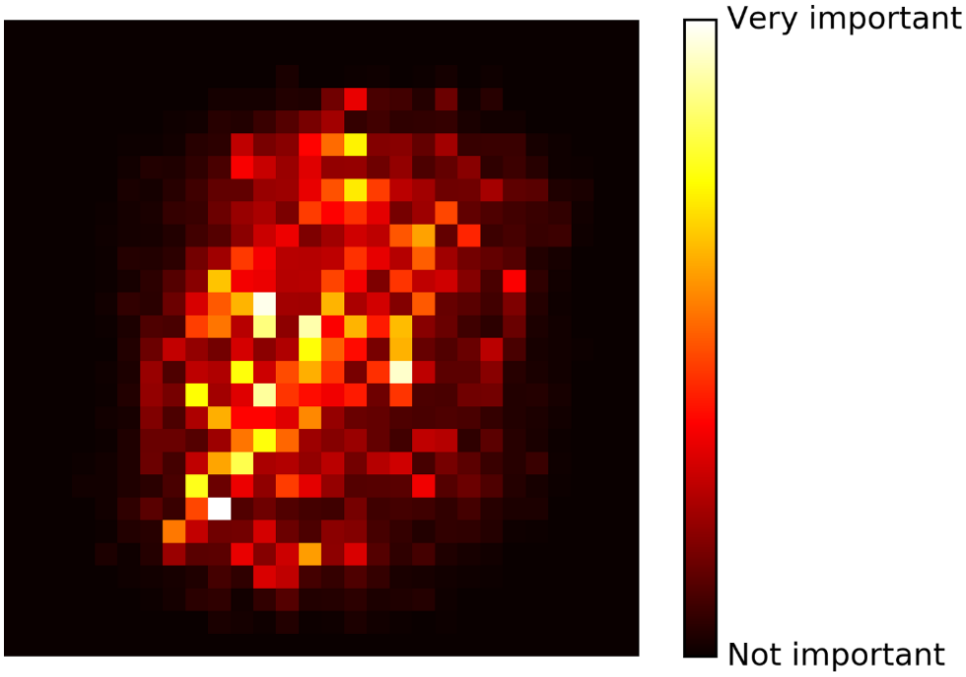


Figure 7-6. MNIST pixel importance (according to a Random Forest classifier)

The importance of each pixel in classification of handwritten digit images (MNIST data, RandomForestClassifier)

Boosting

Another ensemble learning method.

Idea: you can build a very good learning algorithm from a weak one

Weak learner: a predictor that does only a little better than chance

Weak learners are usually also imagined to be simple, like a one-node decision tree.

In the words of Freund and Schapire:

Boosting refers to this general problem of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb.

Freund and Schapire, A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting, Journal of Computer and System Sciences, 1997.

AdaBoost

Predictors are trained sequentially.

1. Train the first predictor. Look how it performs on the training set.
2. Train a second predictor, but:
   - weight the training data so that instances misclassified in step 1 get higher weight
   - train to minimize the "weighted sum of misclassifications"
3. Train additional predictors in the same way, first updating weights based on misclassifications of the previous predictor.
4. When making predictions of the ensemble on test data, the predictors have different weights depending on their accuracy.

AdaBoost example

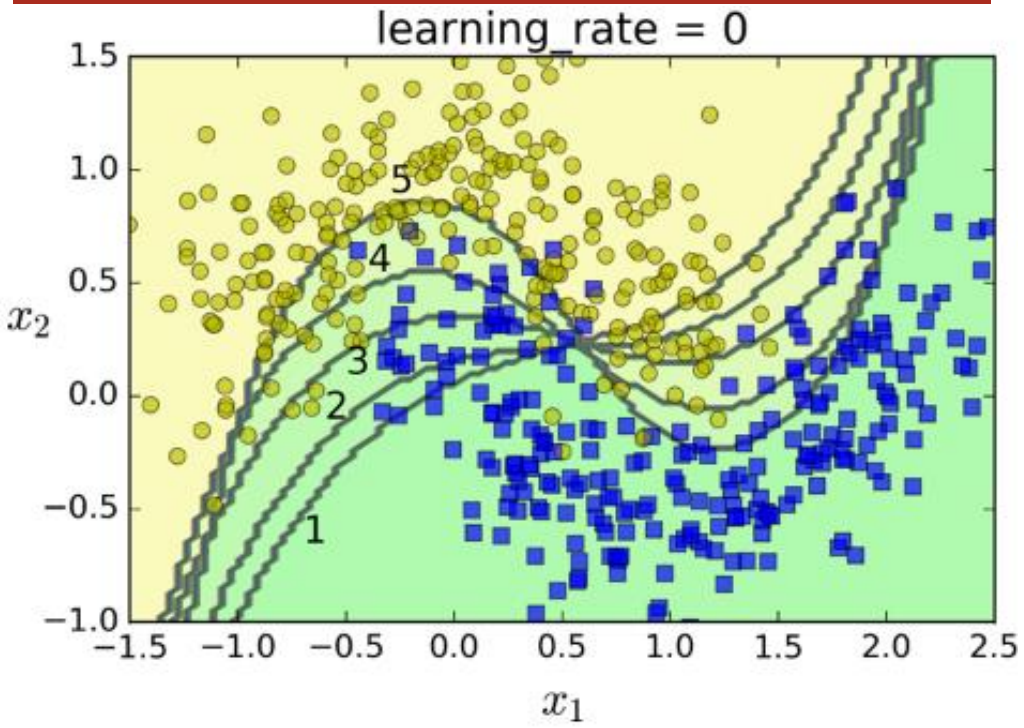


Figure and code from Géron

```
sample_weights = np.ones(m)
    for i in range(5):
        plt.subplot(subplot)
        svm_clf = SVC(kernel="rbf", C=0.05, random_state=42)
        svm_clf.fit(X_train, y_train, sample_weight=sample_weights)
        y_pred = svm_clf.predict(X_train)
        sample_weights[y_pred != y_train] *= (1 + learning_rate)
        plot_decision_boundary(svm_clf, X, y, alpha=0.2)
```

AdaBoost example

training data | initial weight values

x_1	x_2	y	weight	$\hat{y}$	
.2	234	0	0.2	0	✓
.5	43	0	0.2	1	
-.1	54	1	0.2	1	✓
.6	3	0	0.2	0	✓
.3	302	1	0.2	0	

error rate is 0.4

predictor weight is 0.405

updated weight values

predictor 1

x_1	x_2	y	weight	$\hat{y}$	
.2	234	0	0.167		
.5	43	0	0.25		
-.1	54	1	0.167		
.6	3	0	0.167		
.3	302	1	0.25		

predictor 2

AdaBoost detail: predictor error rates

1. Compute weighted predictor error rate r_j

$$r_j = \frac{\sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$$

where $\hat{y}_j^{(i)}$ is the j^{th} predictor's prediction for the i^{th} instance.

In this example,

$$r_j = \frac{0.25 + 0.167}{1}$$

$$= 0.417$$

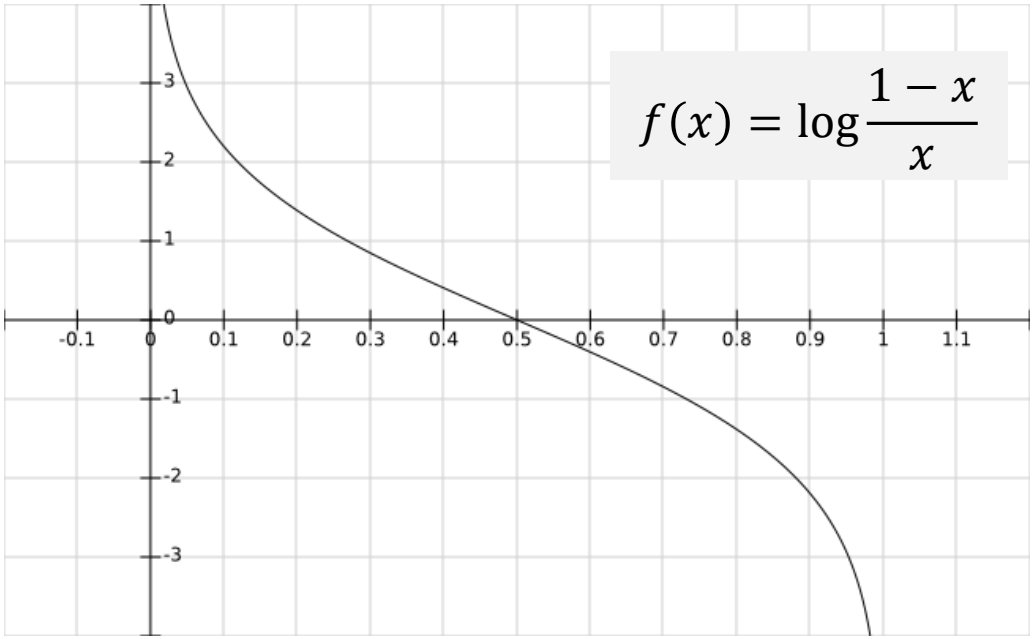
x_1	x_2	y	weight	$\hat{y}$	
.2	234	0	0.167	0	✓
.5	43	0	0.25	1	
-.1	54	1	0.167	1	✓
.6	3	0	0.167	1	
.3	302	1	0.25	1	✓

AdaBoost detail: predictor weights

2. Compute predictor weight α_j

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

(r_j is the predictor's weighted error rate)



$$f(x) = \log \frac{1 - x}{x}$$

$$\log \frac{1-.417}{.417} = 0.335$$

Predictor weight will be positive if error rate < 0.5, else negative

AdaBoost detail: instance weights

3. Update instance weights

for $i = 1, 2, \cdots, m$

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j) & \text{if } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

In our example, $\exp(\alpha_j) = 1.39$

x_1	x_2	y	weight	$\hat{y}$	
.2	234	0	0.167	0	✓
.5	43	0	0.25	1	
-.1	54	1	0.167	1	✓
.6	3	0	0.167	1	
.3	302	1	0.25	1	✓

adjust weights	normalize
→ 0.167	0.143
→ 0.25 * 1.39 = 0.348	0.300
→ 0.167	0.143
→ 0.167 * 1.39 = 0.232	0.200
→ 0.25	0.215

AdaBoost detail: making predictions

Compute predictions from all of the predictors, then weight them by the predictor weights.

Predict the class with greatest sum of weighted predictions.

$$\hat{y}(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \sum_{\substack{j=1 \\ \hat{y}_j(\mathbf{x})=k}}^{N} \alpha_j \quad \text{where } N \text{ is the number of predictors.}$$

assume N = 4

	predictor 1	predictor 2	predictor 3	predictor 4
prediction	1	2	1	3
weight	0.41	0.335	0.89	1.54

	class 1	class 2	class 3
sum of weighted votes	0.41 + 0.89	0.335	1.54

prediction of the ensemble is 1

AdaBoost with Scikit-Learn

```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5)
ada_clf.fit(X_train, y_train)
```

The code builds an AdaBoost classifier that uses "decision stumps" as the predictors.

A decision stump is a decision tree with a height of 1: it contains only a root node and two leaf nodes.

SAMME is a multi-class AdaBoost.

SAMME.R is a variant of SAMME that can use class probability information from predictors.

Summary

- Random forests ≅ bagging + decision trees
- Boosting concept: build good predictors from "weak learners"
- AdaBoost
  - predictors are trained sequentially
  - training instances have weights
  - the weights are adjusted after each predictor is trained to focus training on instances that are misclassified