Chapter 15
Boosting and Stacking

15.1 The Problem

The ensemble technique relies on an aggregate of models' output to provide a better prediction. Other than voting and bagging, we can use boosting and stacking.

15.2 Boosting

Boosting (or hypothesis boosting) refers to an ensemble method that builds a strong learner out of a combination of weak learners (i.e., learners that perform slightly better than random guessing). The predictors are trained sequentially, and each subsequent predictor tries to correct the current one [1]. The dataset is the same for all algorithms; however, each data instance is subject to a weight based on the outcome of the previous model's success [2]; in each iteration, to factor in the prediction difficulty of incorrectly classified instances, their weight is increased. We usually use this technique when learning a new skill, as we focus our attention on difficult aspects. Boosting algorithms differ in the way they calculate the weights (Fig. 15.1).

15.3 Stacking

In stacking (or stacking generalization [3]), the outputs of several algorithms are used as the input of the main algorithm (called sometime the blender [1]) that is supposed to make the final prediction. Practically, we feed the blender with the predicted outcomes of the preceding algorithms. The training dataset is divided into

C. El Morr et al., *Machine Learning for Practical Decision Making*, International Series in Operations Research & Management Science 334,
https://doi.org/10.1007/978-3-031-16990-8_15

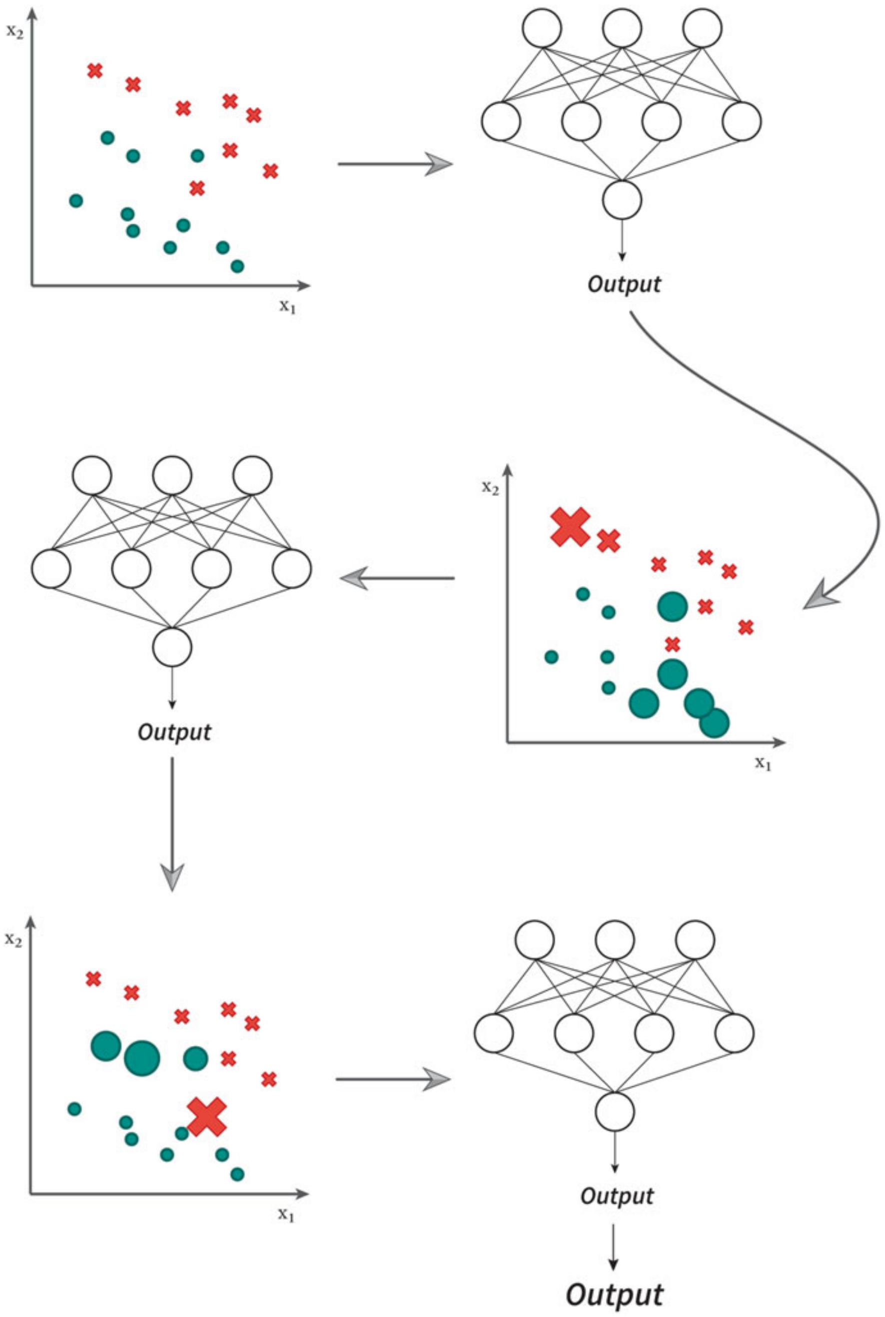


Fig. 15.1 Boosting in action

two parts, a subset (a holdout) to train the blender and a subset to train the other algorithms. The blender uses the outcomes of the other algorithms as input features and the labels from the holdout dataset to train the blender. The blender will learn to predict the labels based on the input features (i.e., the outcomes of the algorithms).

What we have just explained is a stacking mechanism with two layers and one blender. It is possible to create stacking with more than two layers; for instance, in stacking with three layers, the dataset is split into three subsets. The first is used at layer 1 to generate the outcomes, which will act as input for the first blender at layer 2, which will also use the labels of the second subset for training. The second blender at layer 3 will act similarly, i.e., it uses the outcomes of layer 2 and the labels of the third subset for training.

15.4 Boosting Example

15.4.1 AdaBoost Algorithm

AdaBoost is a boosting algorithm that focuses its attention on the training instances that the predecessor algorithm misclassified [4].

Initially, each instance of the dataset is assigned equal weight. Using the training dataset, AdaBoost trains a weak learner classifier, such as a decision tree with one level (called a decision stump). Then, AdaBoost uses the developed model to make predictions about the training dataset and increases the weight for the misclassified instances. The dataset with the updated weights is then used for training in the next iteration. The process continues until the desired number of classifiers is reached or no further improvement in classification can be made.

Once trained, AdaBoost makes predictions by calculating all the predictions of all the predictors, weighting them using the predictors' weights. The predicted class is determined by the majority of the weighted votes [1].

At each iteration, AdaBoost focuses on misclassified instances; this strategy improves the performance of the weak classifiers drastically (Fig. 15.2).

The following is a summary of the training of AdaBoost algorithm for a dataset of m instances and N predictors:

```
Initialize the weights $w^{(i)} = \frac{1}{m}$
For $j = 1$ to $N$
Begin For $j$
    1. For $i = 1$ to $m$
          Begin For $i$
       Calculate $j$th prediction $\widehat{y}_j^{(i)}$ for each instance $x^{(i)}$
          End For $i$
```

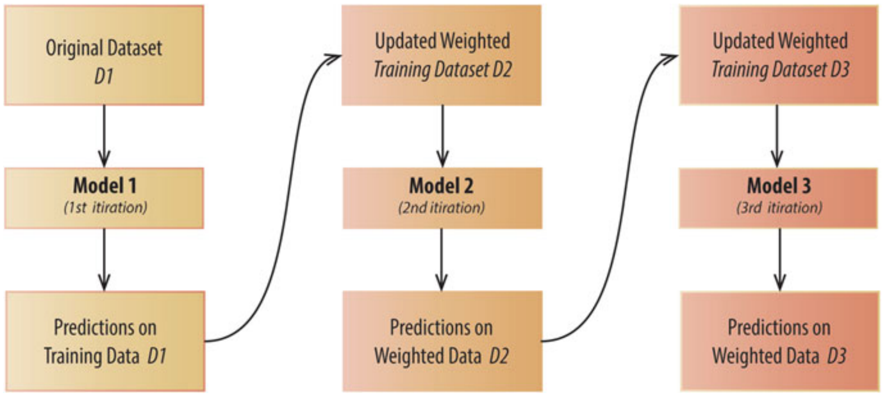

Original Dataset D1
Model 1 (1st itiration)
Predictions on Training Data D1
Updated Weighted Training Dataset D2
Model 2 (2nd itiration)
Predictions on Weighted Data D2
Updated Weighted Training Dataset D3
Model 3 (3rd itiration)
Predictions on Weighted Data D3

Fig. 15.2 Overview of AdaBoost

2. Calculate the *j*th predictor's error rate

$$r_j = \frac{\left(\text{for } \widehat{y}_j^{(i)} \neq y^{(i)}\right) \sum_{i=1}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$$

where $\widehat{y}_j^{(i)}$ is the *j*th prediction for the instance i

3. Calculate the *j*th predictor's weight

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

where η is the learning rate (by default, $\eta = 1$).

4. Update the instances' weights

For $i = 1$ to m

Begin For i

$$\text{if } \left(\widehat{y}_j^{(i)} \neq y^{(i)}\right) \text{ then } w^{(i)} = w^{(i)} \exp\left(\alpha_j\right)$$

End For i

5. Normalize the weights

```
For $i = 1$ to $m$
Begin For $i$
    $w^{(i)} = \frac{w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$
End For $i$
End For $j$
```

To predict using AdaBoost, for a new instance, the weak learners calculate in sequence a predicted value as either +1 (for first class) or –1 (for the second class); each prediction is weighted by the predictor's weight. The weighted sum is calculated; AdaBoost assigns the instance to the first class if the weighted sum is positive and to the second class otherwise. Classifying an instance x with AdaBoost with N predictors can be summarized as follows:

$$\hat{y}(x) = \underset{k}{\operatorname{argmax}} \sum_{\substack{j=1 \\ \hat{y}_j(x)=k}}^{N} \alpha_j$$

15.4.2 AdaBoost Example

Download the "Iris" file from the Weka datasets or from the Kaggle website using the following link: https://www.kaggle.com/uciml/iris. Open the file in Weka and choose the AdaBoost algorithm in the Classify tab (Fig. 15.3).

Check the AdaBoost parameters and get acquainted with them (you can use the More button for more information). Accept the default parameters. Explore particularly the Classifier parameter; you can choose classifiers other than the decision stump (Fig. 15.4). Choose cross-validation with tenfolds (Fig. 15.5) and click the Start button. The output window displays the AdaBoost results (Fig. 15.6).

AdaBoost has performed ten iterations of cross-validation, as per our request. There are 143 (95.33%) correctly classified instances and seven (4.73%) incorrectly classified ones. The root mean squared error (RMSE) that we are trying to minimize is 0.1729. We can notice that the class Iris-setosa was clearly identified with a perfect area under the curve (AUC) (i.e., ROC area). The AUCs for Iris-versicolor and Iris-virginica were 0.92 and 0.93, respectively, indicating a high ability of the model to classify both types of irises. The confusion table shows five Iris-versicolor incorrectly classified as Iris-virginica and two Iris-virginica incorrectly classified

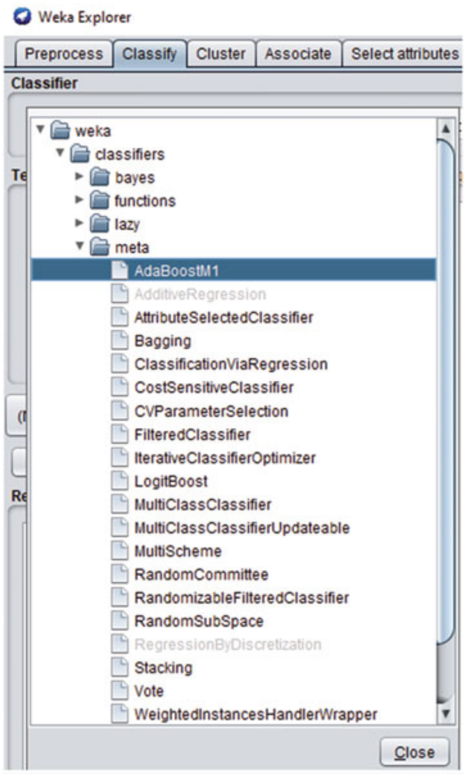
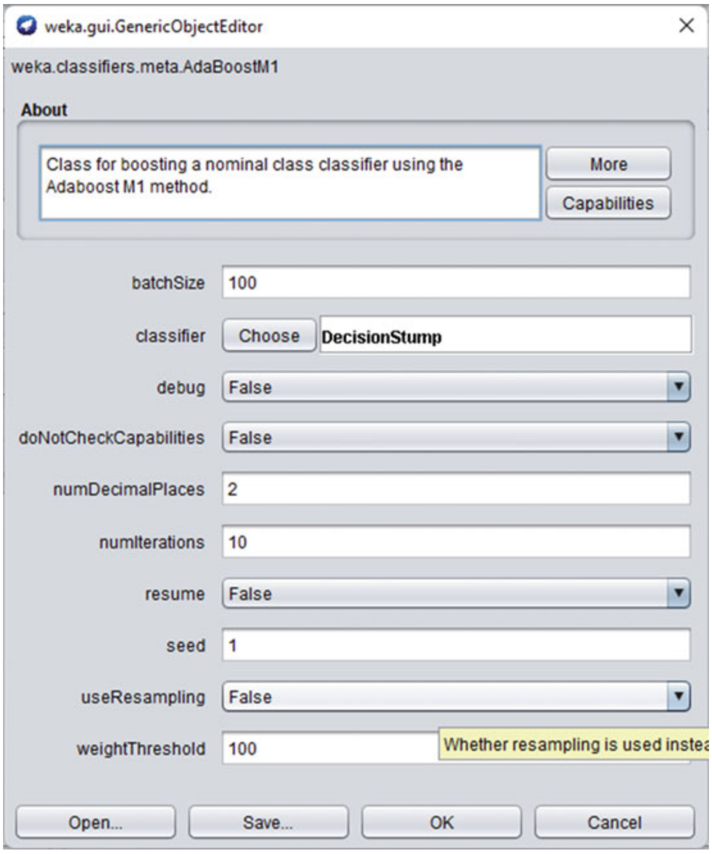


Fig. 15.3 AdaBoost classifier in Weka

as Iris-versicolor. The window shows at the top that the classification was based on the petal length value 2.45 to differentiate between the Iris-setosa and the two other types, the decision being if petal length value is <2.45, then the flower is Iris-setosa.

15.5 Key Terms

1. Boosting
2. Hypothesis boosting
3. Weak learners
4. Stacking
5. Blender
6. Holdout sample
7. AdaBoost
8. Decision stump



weka.classifiers.meta.AdaBoostM1

Class for boosting a nominal class classifier using the Adaboost M1 method.

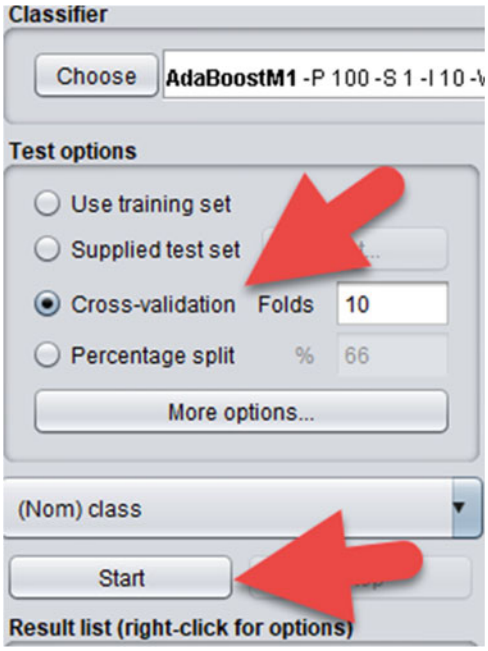
batchSize	100
classifier	DecisionStump
debug	False
doNotCheckCapabilities	False
numDecimalPlaces	2
numIterations	10
resume	False
seed	1
useResampling	False
weightThreshold	100

Whether resampling is used instea

Fig. 15.4 AdaBoost parameters in Weka

15.6 Test Your Understanding

1. Explain how stacking functions.
2. Describe boosting.
3. What are some of the challenges in boosting?
4. What is a decision stump?
5. Cite some of the hyperparameters of AdaBoost.


Classifier

Choose AdaBoostM1 -P 100 -S 1 -I 10 -

Test options

◯ Use training set

◯ Supplied test set

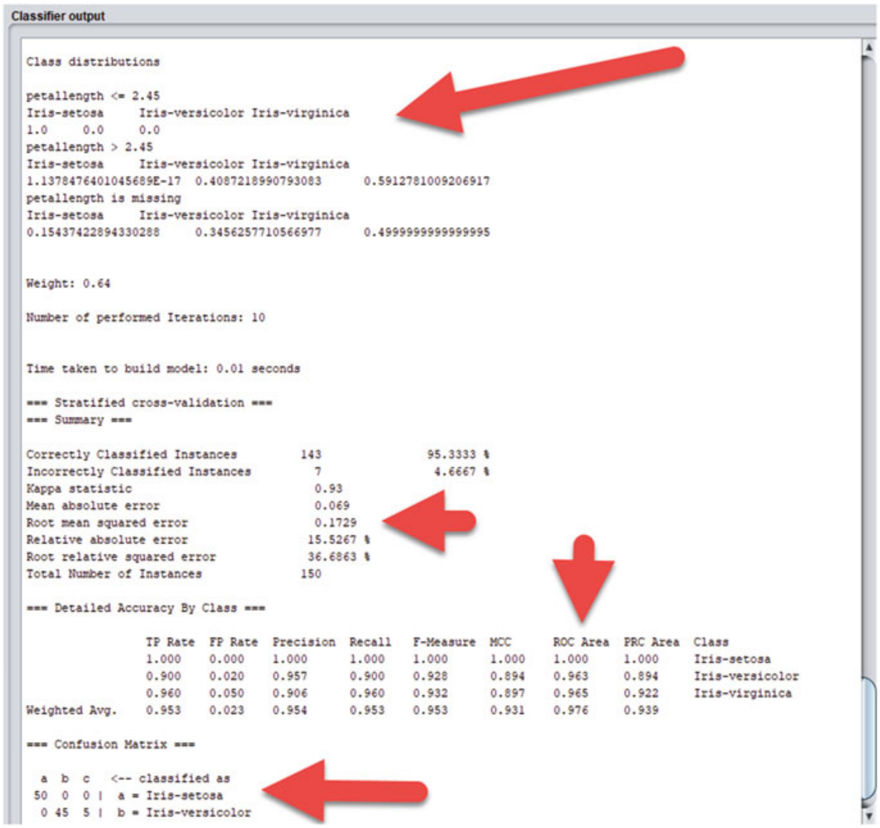
● Cross-validation Folds 10

◯ Percentage split % 66

Fig. 15.5 Choosing to train the model using cross-validation with tenfolds

15.7 Read More

1. Abbruzzo, A., Tamburo, E., Varrica, D., Dongarrà, G., & Mineo, A. (2016). Penalized linear discriminant analysis and Discrete AdaBoost to distinguish human hair metal profiles: The case of adolescents residing near Mt. Etna. Chemosphere, 153, 100–106. doi: 10.1016/j.chemosphere.2016.03.029
2. Barczak, A. L. C., Johnson, M. J., & Messom, C. H. (2008). Empirical evaluation of a new structure for AdaBoost. Paper presented at the Proceedings of the 2008 ACM symposium on Applied computing, Fortaleza, Ceara, Brazil. https://doi.org/10.1145/1363686.1364109
3. Bartlett, P. L., & Traskin, M. (2007). AdaBoost is Consistent. J. Mach. Learn. Res., 8, 2347–2368.
4. Cai, W., Qiu, L., Li, W., Yu, J., & Wang, L. (2019). Practical Fall Detection Algorithm based on Adaboost. Paper presented at the Proceedings of the 2019 4th International Conference on Biomedical Signal and Image Processing (ICBIP 2019), Chengdu, China. https://doi.org/10.1145/3354031.3354056
5. Carreras, X., Màrquez, L., & Padró, L. (2002). Named Entity Extraction using AdaBoost. Paper presented at the proceedings of the 6th conference on Natural language learning - Volume 20. https://doi.org/10.3115/1118853.1118857
6. Carreras, X., Màrquez, L., & Padró, L. (2003). A simple named entity extractor using AdaBoost. Paper presented at the Proceedings of the seventh conference



```
Class distributions

petallength <= 2.45
Iris-setosa     Iris-versicolor Iris-virginica
1.0     0.0     0.0
petallength > 2.45
Iris-setosa     Iris-versicolor Iris-virginica
1.1378476401045689E-17  0.4087218990793083      0.5912781009206917
petallength is missing
Iris-setosa     Iris-versicolor Iris-virginica
0.15437422894330288     0.3456257710566977      0.4999999999999995


Weight: 0.64

Number of performed Iterations: 10


Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         143               95.3333 %
Incorrectly Classified Instances         7                4.6667 %
Kappa statistic                          0.93
Mean absolute error                      0.069
Root mean squared error                  0.1729
Relative absolute error                 15.5267 %
Root relative squared error             36.6863 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-setosa
                 0.900    0.020    0.957      0.900    0.928      0.894    0.963     0.894     Iris-versicolor
                 0.960    0.050    0.906      0.960    0.932      0.897    0.965     0.922     Iris-virginica
Weighted Avg.    0.953    0.023    0.954      0.953    0.953      0.931    0.976     0.939

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 45  5 |  b = Iris-versicolor
```

Fig. 15.6 AdaBoost results when run on the iris dataset

on Natural language learning at HLT-NAACL 2003 - Volume 4, Edmonton, Canada. https://doi.org/10.3115/1119176.1119197

7. Chen, Y., Li, X., & Sun, W. (2020). Research on Stock Selection Strategy Based on AdaBoost Algorithm. Paper presented at the Proceedings of the 4th International Conference on Computer Science and Application Engineering, Sanya, China. https://doi.org/10.1145/3424978.3425084
8. Du, N., Li, K., Mahajan, S. D., Schwartz, S. A., Nair, B. B., Hsiao, C. B., & Zhang, A. (2011). Gene Co-Adaboost: a semi-supervised approach for classifying gene expression data. Paper presented at the Proceedings of the 2nd ACM Conference on Bioinformatics, Computational Biology and Biomedicine, Chicago, Illinois. https://doi.org/10.1145/2147805.2147892
9. Frost, J., Beekers, N., Hengst, B., & Vendeloo, R. (2012). Meeting cancer patient needs: designing a patient platform. Paper presented at the CHI '12 Extended Abstracts on Human Factors in Computing Systems, Austin, Texas, USA. https://doi.org/10.1145/2212776.2223806

10. Gutiérrez-Tobal, G. C., Álvarez, D., Del Campo, F., & Hornero, R. (2016). Utility of AdaBoost to Detect Sleep Apnea-Hypopnea Syndrome From Single-Channel Airflow. IEEE Trans Biomed Eng, 63(3), 636–646. doi: 10.1109/tbme.2015.2467188
11. He, B., Huang, C., Sharp, G., Zhou, S., Hu, Q., Fang, C., . . . Jia, F. (2016). Fast automatic 3D liver segmentation based on a three-level AdaBoost-guided active shape model. Med Phys, 43(5), 2421. doi: 10.1118/1.4946817
12. Hsu, K.-W. (2017). Heterogeneous AdaBoost with stochastic algorithm selection. Paper presented at the Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, Beppu, Japan. https://doi.org/10.1145/3022227.3022266
13. Hu, W., & Hu, W. (2005). Network-Based Intrusion Detection Using Adaboost Algorithm. Paper presented at the Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence. https://doi.org/10.1109/WI.2005.107
14. Kadlček, F., & Fučík, O. (2013). Fast and energy efficient AdaBoost classifier. Paper presented at the Proceedings of the 10th FPGAworld Conference, Stockholm, Sweden. https://doi.org/10.1145/2513683.2513685
15. Memari, N., Ramli, A. R., Bin Saripan, M. I., Mashohor, S., & Moghbel, M. (2017). Supervised retinal vessel segmentation from color fundus images based on matched filtering and AdaBoost classifier. PLoS One, 12(12), e0188939. doi: 10.1371/journal.pone.0188939
16. Mukherjee, I., Rudin, C., & Schapire, R. E. (2013). The rate of convergence of AdaBoost. J. Mach. Learn. Res., 14(1), 2315–2347.
17. Park, S. Y., & Chen, Y. (2017). Patient Strategies as Active Adaptation: Understanding Patient Behaviors During an Emergency Visit. Paper presented at the Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, Colorado, USA. https://doi.org/10.1145/3025453.3025978
18. Reiss, A., Hendeby, G., & Stricker, D. (2013). Confidence-based multiclass AdaBoost for physical activity monitoring. Paper presented at the Proceedings of the 2013 International Symposium on Wearable Computers, Zurich, Switzerland. https://doi.org/10.1145/2493988.2494325
19. Rudin, C., & Schapire, R. E. (2009). Margin-based Ranking and an Equivalence between AdaBoost and RankBoost. J. Mach. Learn. Res., 10, 2193–2232.
20. Saravanakumar, S., & Thangaraj, P. (2019). A Computer Aided Diagnosis System for Identifying Alzheimer's from MRI Scan using Improved Adaboost. J Med Syst, 43(3), 76. doi: 10.1007/s10916-018-1147-7
21. Song, X., Rui, T., Zha, Z., Wang, X., & Fang, H. (2015). The AdaBoost algorithm for vehicle detection based on CNN features. Paper presented at the Proceedings of the 7th International Conference on Internet Multimedia Computing and Service, Zhangjiajie, Hunan, China. https://doi.org/10.1145/2808492.2808497

22. Sun, J., Wang, F., Hu, J., & Edabollahi, S. (2012). Supervised patient similarity measure of heterogeneous patient records. SIGKDD Explor. Newsl., 14(1), 16–24. doi: 10.1145/2408736.2408740
23. Thongkam, J., Xu, G., Zhang, Y., & Huang, F. (2008). Breast cancer survivability via AdaBoost algorithms. Paper presented at the Proceedings of the second Australasian workshop on Health data and knowledge management - Volume 80, Wollongong, NSW, Australia.
24. Wang, B., Qi, Z., Chen, S., Liu, Z., & Ma, G. (2017). Multi-vehicle detection with identity awareness using cascade Adaboost and Adaptive Kalman filter for driver assistant system. PLoS One, 12(3), e0173424. doi: 10.1371/journal.pone.0173424
25. Wang, M. Y., Li, P., & Qiao, P. L. (2016). The Virtual Screening of the Drug Protein with a Few Crystal Structures Based on the Adaboost-SVM. Comput Math Methods Med, 2016, 4809831. doi: 10.1155/2016/4809831
26. Wang, Q., & Wei, X. (2020). The Detection of Network Intrusion Based on Improved Adaboost Algorithm. Paper presented at the Proceedings of the 2020 4th International Conference on Cryptography, Security and Privacy, Nanjing, China. https://doi.org/10.1145/3377644.3377660
27. Wang, Y., Ru, J., Jiang, Y., & Zhang, J. (2019). Adaboost-SVM-based probability algorithm for the prediction of all mature miRNA sites based on structured-sequence features. Sci Rep, 9(1), 1521. doi: 10.1038/s41598-018-38048-7
28. Yang, Y., Liu, C., & Liu, N. (2019). Credit Card Fraud Detection based on CSat-Related AdaBoost. Paper presented at the Proceedings of the 2019 8th International Conference on Computing and Pattern Recognition, Beijing, China. https://doi.org/10.1145/3373509.3373548
29. Yousefi, M., Yousefi, M., Ferreira, R. P. M., Kim, J. H., & Fogliatto, F. S. (2018). Chaotic genetic algorithm and Adaboost ensemble metamodeling approach for optimum resource planning in emergency departments. Artif Intell Med, 84, 23–33. doi: 10.1016/j.artmed.2017.10.002

15.8 Lab

15.8.1 A Working Example in Python

The heart dataset that will be used for this lab can be downloaded from the following link: https://www.kaggle.com/code/ysthehurricane/heart-failure-prediction-using-adaboost-xgboost/data.

That dataset contains 11 features that will be used to predict heart disease events:

- Age: person’s age in years
- Sex: person’s gender
- ChestPainType: chest pain type
- RestingBP: resting blood pressure in mm Hg

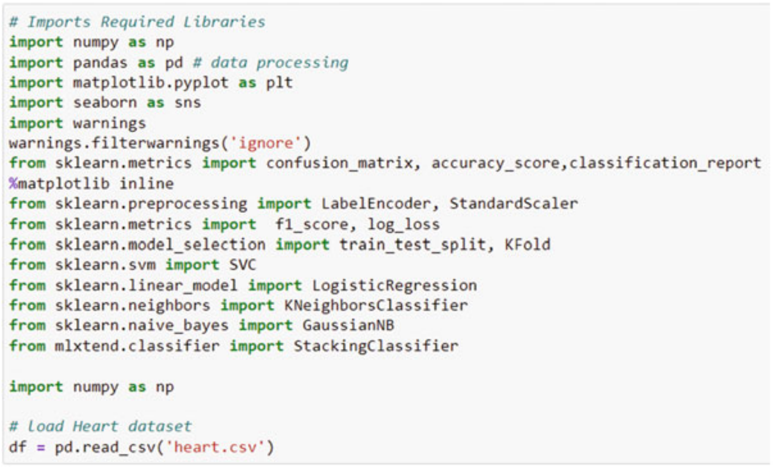
- Cholesterol: serum cholesterol in mm/dL
- FastingBS: blood sugar measurement on fasting
- RestingECG: electrocardiogram results in resting
- MaxHR: maximum heart rate achieved
- ExerciseAngina: exercise-induced angina flag
- Oldpeak: old peak
- ST_Slope: the slope of the peak exercise
- HeartDisease: target class (1 for having heart disease and 0 for not)

15.8.1.1 Loading Heart Dataset

We start by importing the required libraries and loading the heart dataset (Fig. 15.7). When you run the code, if you have not installed previously a needed library you will receive an error message stating that the module was not found, in such cases you need install the missing library using pip.

15.8.1.2 Visualizing Heart Dataset

The next step is to explore the heart dataset visually. We have opted to display the plot heatmap correlations between features' pairs (Fig. 15.8).



```
# Imports Required Libraries
import numpy as np
import pandas as pd # data processing
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import confusion_matrix, accuracy_score,classification_report
%matplotlib inline
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import  f1_score, log_loss
from sklearn.model_selection import train_test_split, KFold
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from mlxtend.classifier import StackingClassifier

import numpy as np

# Load Heart dataset
df = pd.read_csv('heart.csv')
```

Fig. 15.7 Loading heart dataset into pandas

```
#Data Visualisation
plt.figure(1)
sns.heatmap(df.corr())
plt.title('heart failure Correlation')
```

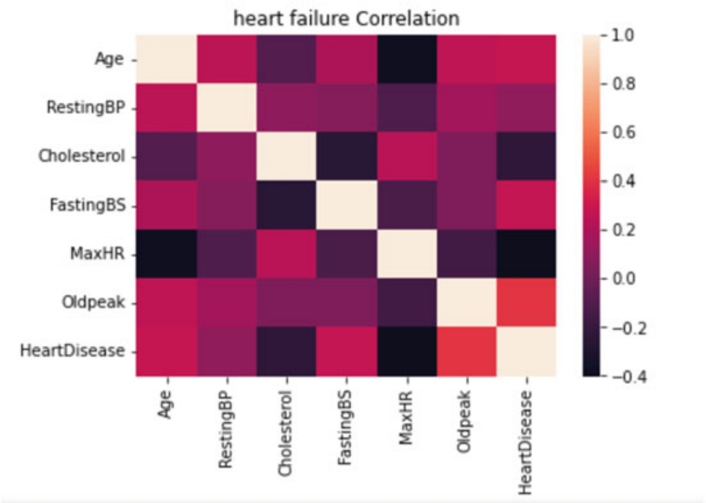


Fig. 15.8 Visualizing heart dataset in heatmap

```
#Preprocess data: converting string to numeric
from sklearn.preprocessing import LabelEncoder
lbl=LabelEncoder()

df['Sex']=lbl.fit_transform(df['Sex'])
df['RestingECG']=lbl.fit_transform(df['RestingECG'])
df['ChestPainType']=lbl.fit_transform(df['ChestPainType'])
df['ExerciseAngina']=lbl.fit_transform(df['ExerciseAngina'])
df['ST_Slope']=lbl.fit_transform(df['ST_Slope'])
```

Fig. 15.9 Preprocess data by mapping string values into numeric ones

15.8.1.3 Preprocess Data

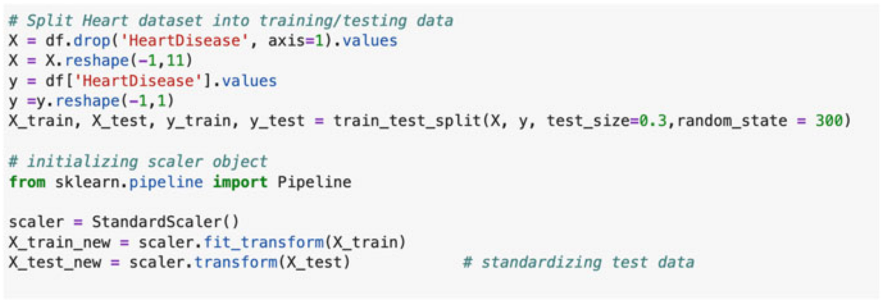
The next step is to convert string values to numeric ones. We have used the LabelEncoder do so (Fig. 15.9), can you achieve the same result using a different approach? Try.

15.8.1.4 Split and Scale Data

We can now choose the features and target, split the original dataset into training and testing datasets and standardize both (Fig. 15.10).

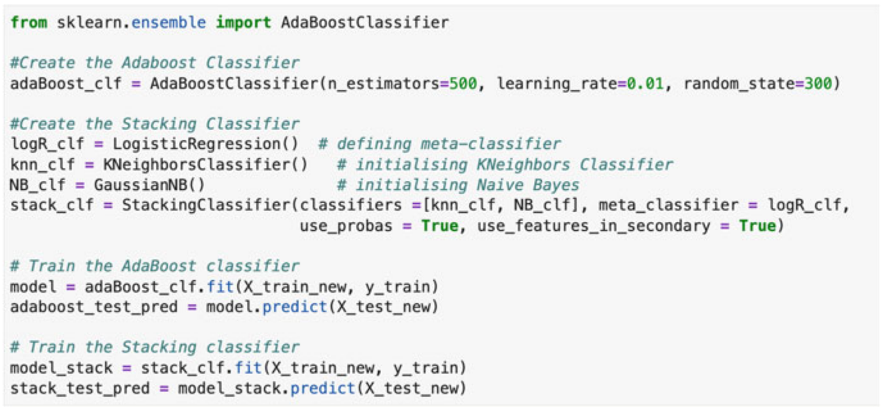
15.8.1.5 Create AdaBoost and Stacking Models

We will use AdaBoost to create a boosting model with a learning_rate=0.01 and n_estimators=500. We will also use a stacking approach using k-nearest neighbors and Gaussian naïve Bayes algorithms as classifiers and logistic regression as a metaclassifier. Then, we train both models on the training dataset and make associated predictions using the testing dataset (Fig. 15.11).



```
# Split Heart dataset into training/testing data
X = df.drop('HeartDisease', axis=1).values
X = X.reshape(-1,11)
y = df['HeartDisease'].values
y =y.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state = 300)

# initializing scaler object
from sklearn.pipeline import Pipeline

scaler = StandardScaler()
X_train_new = scaler.fit_transform(X_train)
X_test_new = scaler.transform(X_test)          # standardizing test data
```

Fig. 15.10 Split and scale heart dataset

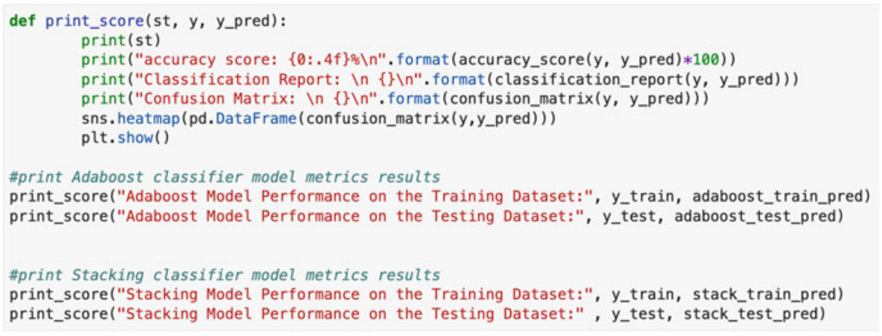


```
from sklearn.ensemble import AdaBoostClassifier

#Create the Adaboost Classifier
adaBoost_clf = AdaBoostClassifier(n_estimators=500, learning_rate=0.01, random_state=300)

#Create the Stacking Classifier
logR_clf = LogisticRegression()  # defining meta-classifier
knn_clf = KNeighborsClassifier()   # initialising KNeighbors Classifier
NB_clf = GaussianNB()              # initialising Naive Bayes
stack_clf = StackingClassifier(classifiers =[knn_clf, NB_clf], meta_classifier = logR_clf,
                               use_probas = True, use_features_in_secondary = True)

# Train the AdaBoost classifier
model = adaBoost_clf.fit(X_train_new, y_train)
adaboost_test_pred = model.predict(X_test_new)

# Train the Stacking classifier
model_stack = stack_clf.fit(X_train_new, y_train)
stack_test_pred = model_stack.predict(X_test_new)
```

Fig. 15.11 Create AdaBoost and stacking models

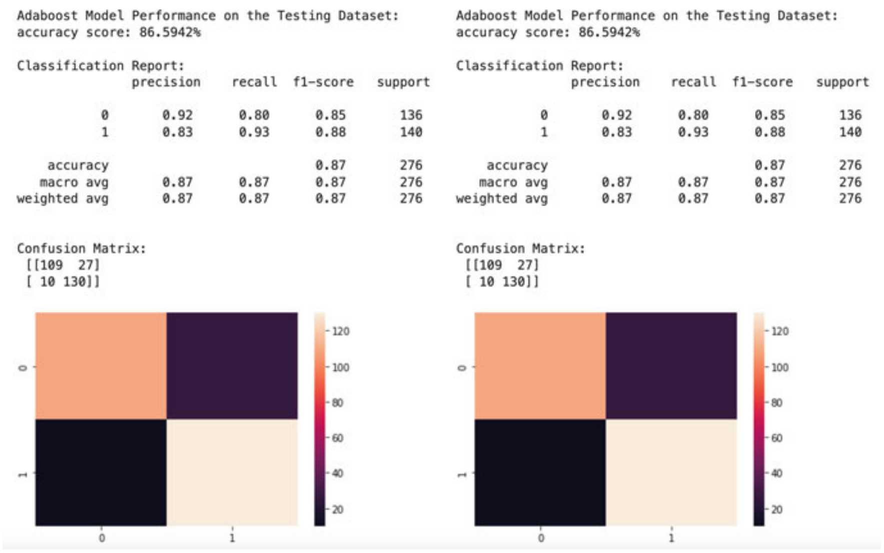
15.8.1.6 Evaluate the AdaBoost and the Stacking Models

The next step is to evaluate the performance of the AdaBoost and Stacking models. We opted to show in this lab the performance on the training and testing datasets for exploration/learning purposes. Figure 15.12 shows the code and Figs. 15.13 and 15.14 show the performance results displayed for AdaBoost and Stacking, respectively.



```python
def print_score(st, y, y_pred):
        print(st)
        print("accuracy score: {0:.4f}%\n".format(accuracy_score(y, y_pred)*100))
        print("Classification Report: \n {}\n".format(classification_report(y, y_pred)))
        print("Confusion Matrix: \n {}\n".format(confusion_matrix(y, y_pred)))
        sns.heatmap(pd.DataFrame(confusion_matrix(y,y_pred)))
        plt.show()

#print Adaboost classifier model metrics results
print_score("Adaboost Model Performance on the Training Dataset:", y_train, adaboost_train_pred)
print_score("Adaboost Model Performance on the Testing Dataset:", y_test, adaboost_test_pred)


#print Stacking classifier model metrics results
print_score("Stacking Model Performance on the Training Dataset:", y_train, stack_train_pred)
print_score("Stacking Model Performance on the Testing Dataset:" , y_test, stack_test_pred)
```

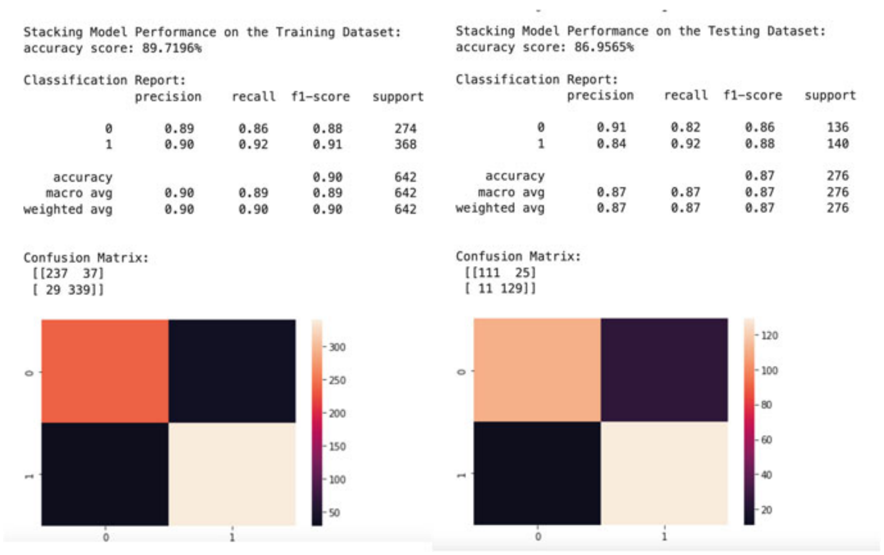
Fig. 15.12 Calculating accuracy and confusion matrix for AdaBoost model



```
Adaboost Model Performance on the Testing Dataset:
accuracy score: 86.5942%

Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.80      0.85       136
           1       0.83      0.93      0.88       140

    accuracy                           0.87       276
   macro avg       0.87      0.87      0.87       276
weighted avg       0.87      0.87      0.87       276

Confusion Matrix:
 [[109  27]
 [ 10 130]]
```

```
Adaboost Model Performance on the Testing Dataset:
accuracy score: 86.5942%

Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.80      0.85       136
           1       0.83      0.93      0.88       140

    accuracy                           0.87       276
   macro avg       0.87      0.87      0.87       276
weighted avg       0.87      0.87      0.87       276

Confusion Matrix:
 [[109  27]
 [ 10 130]]
```

Fig. 15.13 AdaBoost Model Performance on the training and testing Datasets



```
Stacking Model Performance on the Training Dataset:
accuracy score: 89.7196%

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.86      0.88       274
           1       0.90      0.92      0.91       368

    accuracy                           0.90       642
   macro avg       0.90      0.89      0.89       642
weighted avg       0.90      0.90      0.90       642

Confusion Matrix:
 [[237  37]
 [ 29 339]]
```

```
Stacking Model Performance on the Testing Dataset:
accuracy score: 86.9565%

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.82      0.86       136
           1       0.84      0.92      0.88       140

    accuracy                           0.87       276
   macro avg       0.87      0.87      0.87       276
weighted avg       0.87      0.87      0.87       276

Confusion Matrix:
 [[111  25]
 [ 11 129]]
```

Fig. 15.14 Stacking Model Performance on the training and testing Datasets

15.8.1.7 Optimizing the Stacking and AdaBoost Models

The models' performances on the testing datasets are fair. Let us explore the performance of the optimized models after hyperparameter tuning. The results for the Stacking and AdaBoost models are presented in Figs. 15.15 and 15.16, respectively.

15.8.2 Do It Yourself

15.8.2.1 The Heart Disease Dataset Revisited

1. Have you noticed any possible overfitting in the example above?
2. Did you obtain the same results when you run your code? What do you think about those results?
3. During the evaluation step above, we have just applied the models to the testing dataset. That is not the best option. What is a better approach?
4. Use cross-validation to redo the evaluation step.

15.8.2.2 The Iris Dataset

Download the iris dataset and do the following:

```
from sklearn.model_selection import GridSearchCV

sclf = StackingClassifier(classifiers=[knn_clf, NB_clf], meta_classifier=logR_clf)
params = {'kneighborsclassifier__n_neighbors': [1, 5],
          'meta_classifier__C': [0.1, 10.0]}
grid = GridSearchCV(estimator=sclf,
                    param_grid=params,
                    cv=5,
                    refit=True)
grid.fit(X_test_new, y_test)
optimal_Stacking_pred= grid.predict(X_test_new)
print_score("Optimized Stacking Model Performance on the Testing Dataset:", y_test, optimal_Stacking_pred)
```

```
Optimized Stacking Model Performance on the Testing Dataset:
accuracy score: 88.0435%

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.83      0.87       136
           1       0.85      0.93      0.89       140

    accuracy                           0.88       276
   macro avg       0.88      0.88      0.88       276
weighted avg       0.88      0.88      0.88       276


Confusion Matrix:
 [[113  23]
 [ 10 130]]
```

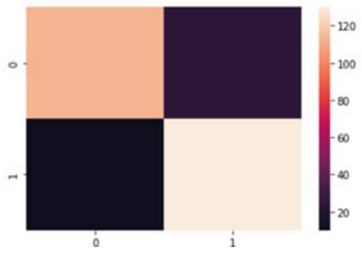
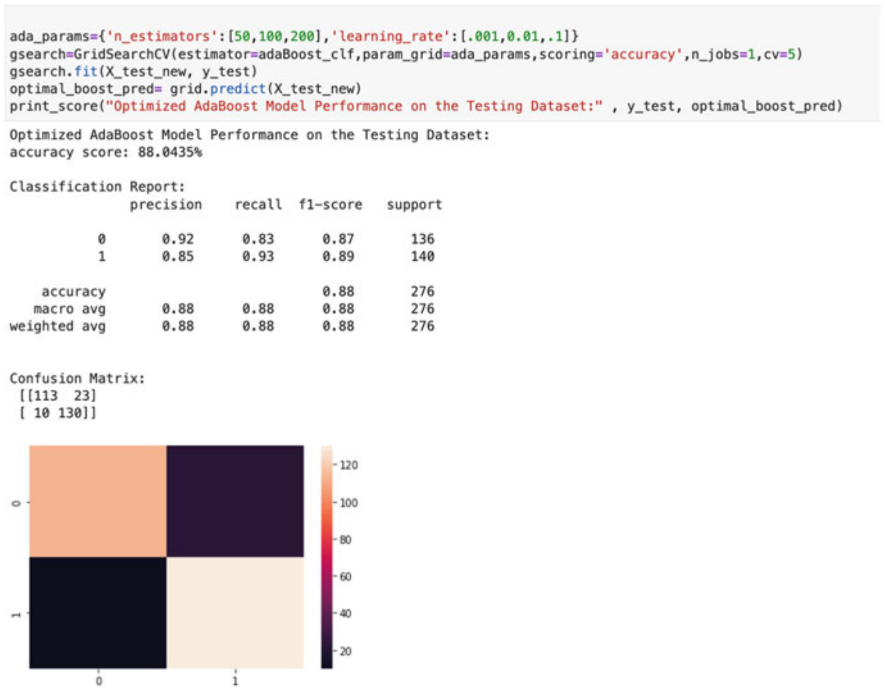


Fig. 15.15 Optimal stacking model performance

1. Load the dataset into pandas.
2. Visualize the dataset and calculate the highest correlations.
3. Preprocess the data.
4. Split the data.
5. Create an AdaBoost model.
6. Evaluate the AdaBoost model.
7. Optimize the AdaBoost model.
8. Create a stack model.
9. Evaluate the stack model.
10. Optimize the stack model.
11. Compare the results between both models and deduce a conclusion.



```
ada_params={'n_estimators':[50,100,200],'learning_rate':[.001,0.01,.1]}
gsearch=GridSearchCV(estimator=adaBoost_clf,param_grid=ada_params,scoring='accuracy',n_jobs=1,cv=5)
gsearch.fit(X_test_new, y_test)
optimal_boost_pred= grid.predict(X_test_new)
print_score("Optimized AdaBoost Model Performance on the Testing Dataset:" , y_test, optimal_boost_pred)
```

```
Optimized AdaBoost Model Performance on the Testing Dataset:
accuracy score: 88.0435%

Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.83      0.87       136
           1       0.85      0.93      0.89       140

    accuracy                           0.88       276
   macro avg       0.88      0.88      0.88       276
weighted avg       0.88      0.88      0.88       276


Confusion Matrix:
 [[113  23]
 [ 10 130]]
```

Fig. 15.16 Optimal AdaBoost model performance

15.8.3 *Do More Yourself*

- https://www.kaggle.com/code/treina/titanic-with-adaboost/data
- https://www.kaggle.com/datasets/zaurbegiev/my-dataset
- https://www.kaggle.com/code/sid321axn/house-price-prediction-gboosting-adaboost-etc/data

References

1. A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (O'Reilly Media, Sebastopol, CA, 2019)
2. Y. Liu, *Python Machine Learning By Example: Build Intelligent Systems Using Python, TensorFlow 2, PyTorch, and Scikit-Learn, 3rd Edition (Kindle Edition)* (Packt, 2020)
3. D.H. Wolpert, Stacked generalization. Neural Netw. **5**(2), 241–259 (1992). https://doi.org/10.1016/S0893-6080(05)80023-1
4. R.E. Schapire, Y. Freund, *Boosting: Foundations and Algorithms* (MIT Press, Cambridge, MA, 2014)