Module 6 : Classification

- To solve the classification problems.
- Reference Notebook : Introduction to Data Mining

Introduction

- Classification is the task of predicting a nominal-valued attribute (known as class label) based on the values of other attributes (known as predictor variables).
- Goals:
    1. To provide examples of using different classification techniques from the scikit-learn library package.
    2. To demonstrate the problem of model overfitting.

6.1 Vertebrate Dataset

```
# Load the vertebrate data set
import pandas as pd

data = pd.read_csv('vertebrate.csv',header='infer')
data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Name	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class
0	human	1	1	0	0	1	0	mammals
1	python	0	0	0	0	0	1	reptiles
2	salmon	0	0	1	0	0	0	fishes
3	whale	1	1	1	0	0	0	mammals
4	frog	0	0	1	0	1	1	amphibians
5	komodo	0	0	0	0	1	0	reptiles
6	bat	1	1	0	1	1	1	mammals
7	pigeon	1	0	0	1	1	0	birds
8	cat	1	1	0	0	1	0	mammals
9	leopard shark	0	1	1	0	0	0	fishes
10	turtle	0	0	1	0	1	0	reptiles
11	penguin	1	0	1	0	1	0	birds
12	porcupine	1	1	0	0	1	1	mammals
13	eel	0	0	1	0	0	0	fishes
14	salamander	0	0	1	0	1	1	amphibians

```
# Convert the Class into a binary classification task : mammals VS non-mammals

data['Class']=data['Class'].replace(['fishes','birds','amphibians','reptiles'],'non-mammals')
data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Name	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class
0	human	1	1	0	0	1	0	mammals
1	python	0	0	0	0	0	1	non-mammals
2	salmon	0	0	1	0	0	0	non-mammals
3	whale	1	1	1	0	0	0	mammals
4	frog	0	0	1	0	1	1	non-mammals
5	komodo	0	0	0	0	1	0	non-mammals
6	bat	1	1	0	1	1	1	mammals
7	pigeon	1	0	0	1	1	0	non-mammals
8	cat	1	1	0	0	1	0	mammals
9	leopard shark	0	1	1	0	0	0	non-mammals
10	turtle	0	0	1	0	1	0	non-mammals
11	penguin	1	0	1	0	1	0	non-mammals
12	porcupine	1	1	0	0	1	1	mammals
13	eel	0	0	1	0	0	0	non-mammals
14	salamander	0	0	1	0	1	1	non-mammals

Cross-tabulation

Apply Pandas cross-tabulation to examine the relationship between different attributes.

```
# apply cross-tabulation to examine the relation between
# the Warm-blooded and Gives Birth attributes with respect to the class

pd.crosstab([data['Warm-blooded'],data['Gives Birth']],data['Class'])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Class	mammals	non-mammals
Warm-blooded	Gives Birth		
0	0	0	7
	1	0	1
1	0	0	2
	1	5	0

The results above show that it is possible to distinguish mammals from non-mammals using thes two attributes alone since each combination of their attribute values would yield only instances that belong to the same class.

For example,mammals can be identified as warm-blooded vertebrates that give birth to their young. Such a relationship can also be derived using a decision tree classifier, as shown by the example given in the next subsection.

3.2 Decision Tree Classifier

Apply a decision tree classifier to the vertebrate dataset described in the previous subsection.

```
from sklearn import tree

Y = data['Class']
X = data.drop(['Name','Class'],axis=1)

clf = tree.DecisionTreeClassifier(criterion='entropy',max_depth=3)
clf = clf.fit(X,Y)
```

The preceding commands will extract the predictor(X) and target class(Y) attributes from the vertebrate dataset and create a decision tree classifier object using entropy as its impurity measure for splitting criterion.

The decision tree class in Python sklearn library also supports using 'gini' as impurity measure.

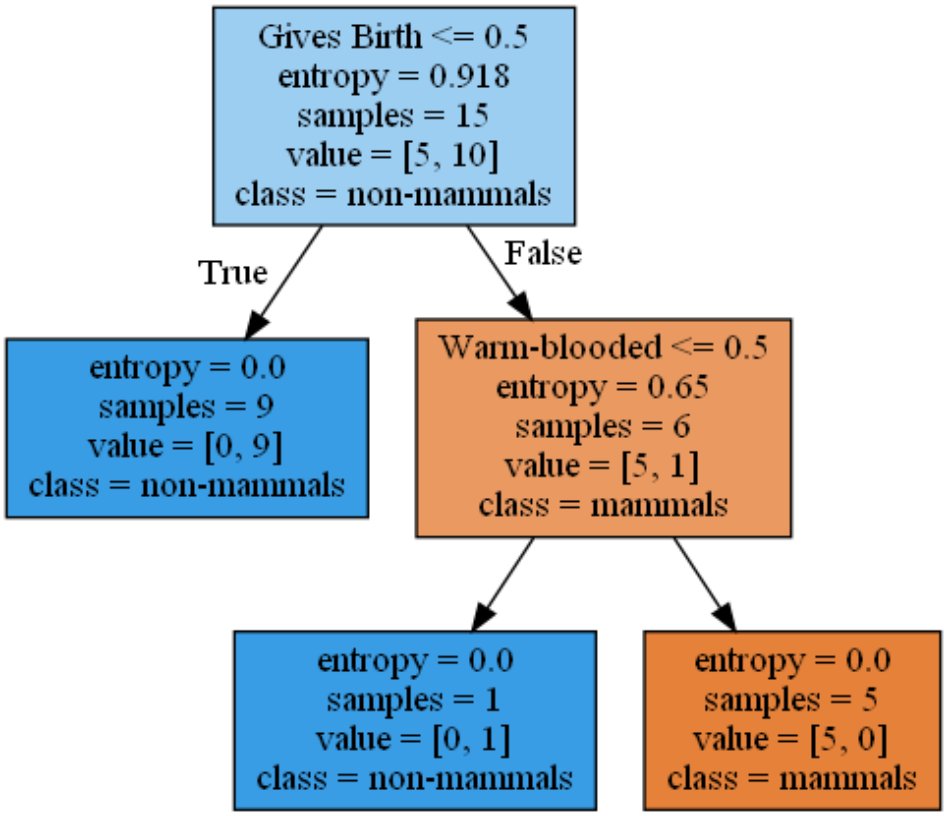
The classifier above is also constrained to generate trees with a maximum depth equals to 3.

Next,the classifier is trained on the labeled data using the fit() function.

Plot the resulting decision tree obtained after training the classifier.

To do this, install both graphviz and pydotplus

```
import pydotplus
from IPython.display import Image

dot_data = tree.export_graphviz(clf,feature_names=X.columns,class_names=['mammals','non-mammals'],filled=True,out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```




```
# apply the decision tree to classify the following test examples
testData = [['gila monster',0,0,0,0,1,1,'non-mammals'],
            ['platypus',1,0,0,0,1,1,'mamals'],
            ['owl',1,0,0,1,1,0,'non-mammals'],
            ['dolphin',1,1,1,0,0,0,'mammals']]
testData = pd.DataFrame(testData,columns = data.columns)
testData
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Name	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class
0	gila monster	0	0	0	0	1	1	non-mammals
1	platypus	1	0	0	0	1	1	mamals
2	owl	1	0	0	1	1	0	non-mammals
3	dolphin	1	1	1	0	0	0	mammals

```
# first extract the predictor and target class attributes from the test data
# and then apply the decision tree classifier to predict their classes.

testY = testData['Class']
testX = testData.drop(['Name','Class'],axis = 1)

predY = clf.predict(testX)
predictions = pd.concat([testData['Name'],pd.Series(predY,name = 'Predicted Class')], axis = 1)
predictions
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Name	Predicted Class
0	gila monster	non-mammals
1	platypus	non-mammals
2	owl	non-mammals
3	dolphin	mammals

Except for platypus,which is an egg-laying mammal,the classifier correctly predicts the class label of the test examples.

We can calculate the accurancy of the classifier on the test data as shown by the example given below.

```
from sklearn.metrics import accuracy_score

print("Accuracy on test data is %.2f"%(accuracy_score(testY,predY)))
```

```
Accuracy on test data is 0.75
```

```
testY
```

```
0    non-mammals
1         mamals
2    non-mammals
3        mammals
Name: Class, dtype: object
```

```
predY
```

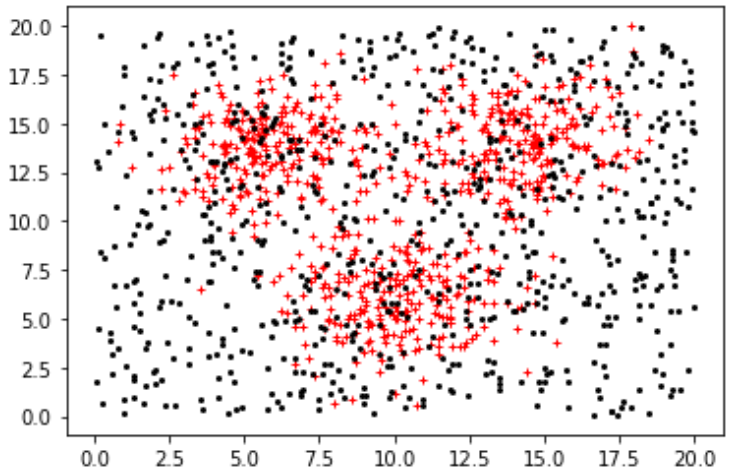
```
array(['non-mammals', 'non-mammals', 'non-mammals', 'mammals'],
      dtype=object)
```

3.3 Model Overfitting

- To illustrate the problem of model overfitting,we consider a two-dimensional dataset containing 1500 labeled instances,each of which is assigned to one of two classes,0 or 1.
- Instances from each class are generated as follows:
  1. Instances from class 1 are generated from a mixture of 3 Gaussian distributions,centered at [6,14].
  2. Instances from class 0 are generated from a uniform distribution in a square region,whose sides have a length equals to 20.
- For simplicity,both classes have equal number of labeled instances.The code for generating and plotting the data is shown below. All instances from class 1 are shown in red while those from class 0 are shown in black.

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.random import random

%matplotlib inline

N = 1500

mean1 = [6,14]
mean2 = [10,6]
mean3 = [14,14]
cov = [[3.5,0],[0,3.5]]   # diagonal covariance

np.random.seed(50)

X = np.random.multivariate_normal(mean1, cov, int(N/6))
X = np.concatenate((X, np.random.multivariate_normal(mean2, cov, int(N/6))))
X = np.concatenate((X, np.random.multivariate_normal(mean3, cov, int(N/6))))
X = np.concatenate((X, 20*np.random.rand(int(N/2),2)))
Y = np.concatenate((np.ones(int(N/2)),np.zeros(int(N/2))))

plt.plot(X[:int(N/2),0],X[:int(N/2),1],'r+',X[int(N/2):,0],X[int(N/2):,1],'k.',ms=4)
```

```
[<matplotlib.lines.Line2D at 0x1c5188ab248>,
 <matplotlib.lines.Line2D at 0x1c5191f7748>]
```



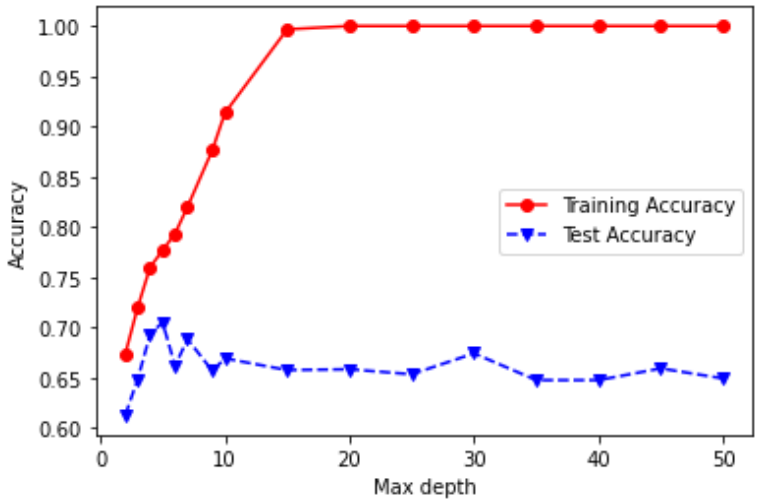
In this example, we reserve 80% of the labeled data for training and the remaining 20% for testing.

We then fit decision trees of different maximum depths (from 2 to 50) to the training set and plot their respective accuracies when applied to the training and test sets.

```
#####################################################
# Training and Test set creation
#####################################################

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.8, random_state = 1)

from sklearn import tree
from sklearn.metrics import accuracy_score

#####################################################
# Model fitting and evvaluation
#####################################################

maxdepths = [2,3,4,5,6,7,9,10,15,20,25,30,35,40,45,50]

trainAcc = np.zeros(len(maxdepths))
testAcc = np.zeros(len(maxdepths))

index = 0
for depth in maxdepths:
    clf = tree.DecisionTreeClassifier(max_depth = depth)
    clf = clf.fit(X_train,Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    trainAcc[index] = accuracy_score(Y_train, Y_predTrain)
    testAcc[index] = accuracy_score(Y_test, Y_predTest)
    index += 1

#####################################################
# Plot of training and test accuracies
#####################################################

plt.plot(maxdepths,trainAcc,'ro-',maxdepths,testAcc,'bv--')
```

```
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Max depth')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```



The plot above shows that training accuracy will continue to improve as the maximum depth of the tree increases (i.e., as the model becomes more complex).

However, the test accuracy initially improves up to a maximum depth of 5, before it gradually decreases due to model overfitting

3.4 Alternative Classification Techniques

- Besides decision tree classifier, the Python sklearn library also supports other classification techniques.
- In this section, we provide examples to illustrate how to apply the k-nearest neighbor classifier, linear classifiers(logistic regression and support vector machine), as well as ensemble methods (boosting, bagging, and random forest) to the 2-dimensional data given in the previous section.

3.4.1 K-Nearest neighbor classifier

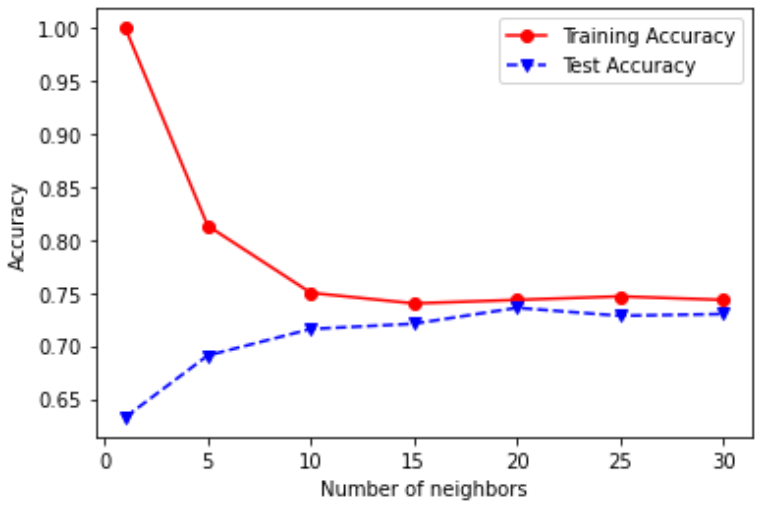
- In this approach, the class label of a test instance is predicted based on the majority class of its k closest training instances.
- The number of nearest neighbors, k, is a hyperparameter that must be provided by the user, along with the distance metric.
- By default, we can use Euclidean distance (which is equivalent to Minkowski distance with an exponent factor equals to p=2)

$$Minkowski\ \ Distance(x,y) = [\sum_{i=1}^{N} \|x_i - y_i\|^p]^{\frac{1}{p}}$$

```
from sklearn.neighbors import KNeighborsClassifier
```

```
import matplotlib.pyplot as plt
%matplotlib inline

numNeighbors = [1, 5, 10, 15, 20, 25, 30]
trainAcc = []
testAcc = []

for k in numNeighbors:
    clf = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)
    clf.fit(X_train, Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    trainAcc.append(accuracy_score(Y_train, Y_predTrain))
    testAcc.append(accuracy_score(Y_test, Y_predTest))

plt.plot(numNeighbors, trainAcc, 'ro-', numNeighbors, testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```



3.4.2 Linear Classifiers

Linear classifiers such as logistic regression and support vector machine (SVM) constructs a linear separating hyperplane to distinguish instances from different classes.

For logistic regression, the model can be described by the following equation:

$$P(y=1|x) = \frac{1}{1+exp^{-w^T x-b}} = \sigma(w^T x+b)$$

The model parameters (w,b) are estimated by optimizing the following regularized negative log-likelihood function:

$$(w^*, b^*) = arg\min_{w,b} -\sum_{i=1}^{N} y_i log[\sigma(w^T x_i + b)] + (1 - y_i) log[\sigma(-w^T x_i - b)] + \frac{1}{C}\Omega([w, b])$$

where C is a hyperparameter that controls the inverse of model complexity (smaller values imply stronger regularization) while $\Omega(.)$ is the regularization term,which by default,is assumed to be an $l_2 - norm$ in sklearn.

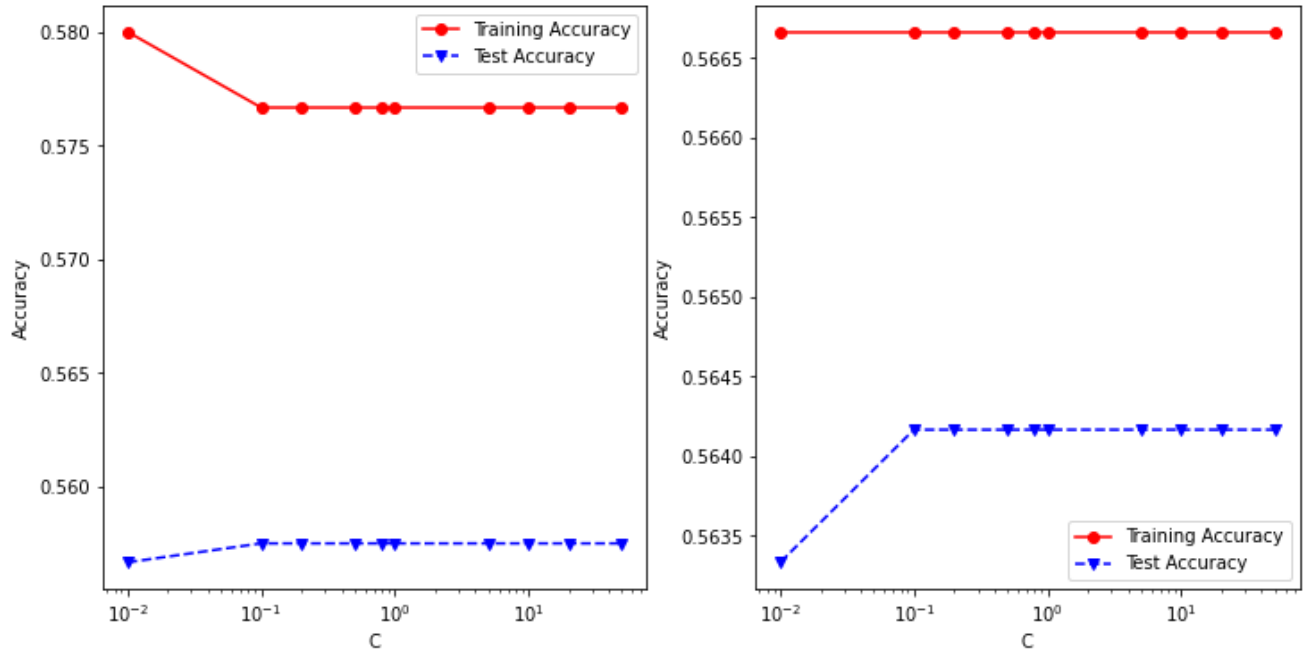
For support vector machine,the model parameters$(w*, b*)$ are estimated by solving the following constrained optimization problem:

$$\min_{w^*,b^*,\xi} \frac{\|w\|^2}{2} + \frac{1}{C}\sum_i \xi_i$$

$$s.t. \forall_i : y_i[w^T \phi(x_i) + b] \geq 1 - \xi_i, \xi_i \geq 0$$

```
from sklearn import linear_model
from sklearn.svm import SVC

C = [0.01, 0.1, 0.2, 0.5, 0.8, 1, 5, 10, 20, 50]
LRtrainAcc = []
LRtestAcc = []
SVMtrainAcc = []
SVMtestAcc = []

for param in C:
    clf = linear_model.LogisticRegression(C = param)
    clf.fit(X_train,Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    LRtrainAcc.append(accuracy_score(Y_train,Y_predTrain))
    LRtestAcc.append(accuracy_score(Y_test,Y_predTest))

    clf = SVC(C = param,kernel = 'linear')
    clf.fit(X_train,Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    SVMtrainAcc.append(accuracy_score(Y_train,Y_predTrain))
    SVMtestAcc.append(accuracy_score(Y_test,Y_predTest))

fig,(ax1,ax2) = plt.subplots(1,2,figsize=(12,6))
ax1.plot(C,LRtrainAcc,'ro-',C,LRtestAcc,'bv--')
ax1.legend(['Training Accuracy','Test Accuracy'])
ax1.set_xlabel('C')
ax1.set_xscale('log')
ax1.set_ylabel('Accuracy')

ax2.plot(C,SVMtrainAcc,'ro-',C,SVMtestAcc,'bv--')
ax2.legend(['Training Accuracy', 'Test Accuracy'])
ax2.set_xlabel('C')
ax2.set_xscale('log')
ax2.set_ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```



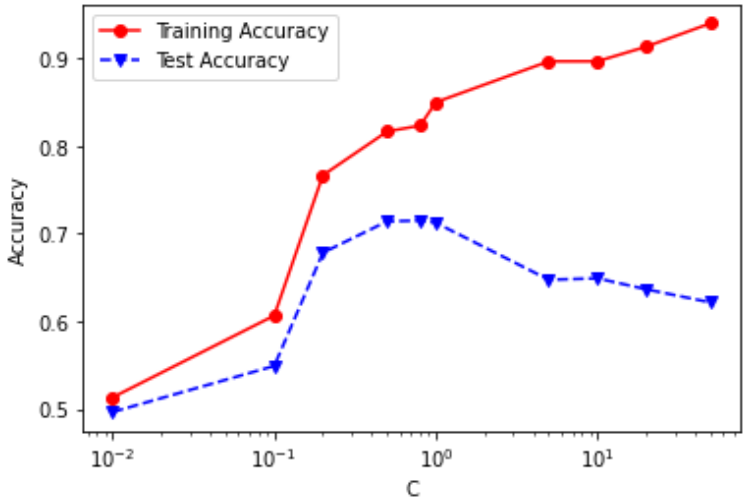
Note that linear classifiers perform poorly on the data since the true decision boundaries between classes are nonlinear for the given 2-dimensional dataset.

3.4.3 Nonlinear Support Vector Machine

The code below shows an example of using nonlinear support vector machine with a Gaussian radial basis function kernel to fit the 2-dimensional dataset.

```
from sklearn.svm import SVC

C = [0.01, 0.1, 0.2, 0.5, 0.8, 1, 5, 10, 20, 50]
SVMtrainAcc = []
SVMtestAcc = []

for param in C:
    clf = SVC(C=param,kernel='rbf',gamma='auto')
    clf.fit(X_train, Y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    SVMtrainAcc.append(accuracy_score(Y_train, Y_predTrain))
    SVMtestAcc.append(accuracy_score(Y_test, Y_predTest))

plt.plot(C, SVMtrainAcc, 'ro-', C, SVMtestAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('C')
plt.xscale('log')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```



3.4.4 Ensemble Methods

An ensemble classifier constructs a set of base classifiers from the training data and performs classification by taking a vote on the predictions made by each base classifier. We consider 3 types of ensemble classifiers in this example: bagging, boosting, and random forest. Detailed explanation about these classifiers can be found in Section 4.10 of the book.

In the example below, we fit 500 base classifiers to the 2-dimensional dataset using each ensemble method.The base classifier corresponds to a decision tree with maximum depth equals to 10

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

numBaseClassifiers = 500
maxdepth = 10
trainAcc = []
testAcc = []

clf = ensemble.RandomForestClassifier(n_estimators=numBaseClassifiers)
clf.fit(X_train, Y_train)
Y_predTrain = clf.predict(X_train)
Y_predTest = clf.predict(X_test)
trainAcc.append(accuracy_score(Y_train, Y_predTrain))
testAcc.append(accuracy_score(Y_test, Y_predTest))

clf = ensemble.BaggingClassifier(DecisionTreeClassifier(max_depth=maxdepth),n_estimators=numBaseClassifiers)
clf.fit(X_train, Y_train)
Y_predTrain = clf.predict(X_train)
Y_predTest = clf.predict(X_test)
trainAcc.append(accuracy_score(Y_train, Y_predTrain))
testAcc.append(accuracy_score(Y_test, Y_predTest))

clf = ensemble.AdaBoostClassifier(DecisionTreeClassifier(max_depth=maxdepth),n_estimators=numBaseClassifiers)
clf.fit(X_train, Y_train)
Y_predTrain = clf.predict(X_train)
```

```
Y_predTest = clf.predict(X_test)
trainAcc.append(accuracy_score(Y_train, Y_predTrain))
testAcc.append(accuracy_score(Y_test, Y_predTest))

methods = ['Random Forest', 'Bagging', 'AdaBoost']
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,6))
ax1.bar([1.5,2.5,3.5], trainAcc)
ax1.set_xticks([1.5,2.5,3.5])
ax1.set_xticklabels(methods)
ax2.bar([1.5,2.5,3.5], testAcc)
ax2.set_xticks([1.5,2.5,3.5])
ax2.set_xticklabels(methods)
```

```
[Text(1.5, 0, 'Random Forest'),
 Text(2.5, 0, 'Bagging'),
 Text(3.5, 0, 'AdaBoost')]
```