Module 6: Classification

The following tutorial contains Python examples for solving classification problems. You should refer to the Chapters 3 and 4 of the "Introduction to Data Mining" book to understand some of the concepts introduced in this tutorial. The notebook can be downloaded from http://www.cse.msu.edu/~ptan/dmbook/tutorials/tutorial6/tutorial6.ipynb (http://www.cse.msu.edu/~ptan/dmbook/tutorials/tutorial6/tutorial6.ipynb).

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). The goals for this tutorial are as follows:

1. To provide examples of using different classification techniques from the scikit-learn library package.
2. To demonstrate the problem of model overfitting.

Read the step-by-step instructions below carefully. To execute the code, click on the corresponding cell and press the SHIFT-ENTER keys simultaneously.

Module 3 Extra Credits

Dingyi Duan ######## USD ########### ADS502

6.1 Vertebrate Dataset

We use a variation of the vertebrate data described in Example 3.1 of Chapter 3. Each vertebrate is classified into one of 5 categories: mammals, reptiles, birds, fishes, and amphibians, based on a set of explanatory attributes (predictor variables). Except for "name", the rest of the attributes have been converted into a *one hot encoding* binary representation. To illustrate this, we will first load the data into a Pandas DataFrame object and display its content.

In [1]:
```
import pandas as pd

data = pd.read_csv('M3 Vertebrate Data Set.csv',header='infer')
data
```

Out[1]:

	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

Given the limited number of training examples, suppose we convert the problem into a binary classification task (mammals versus non-mammals). We can do so by replacing the class labels of the instances to *non-mammals* except for those that belong to the *mammals* class.

In [2]:
```
# Use replace() to replace the varibales to 'non-mammals' for easier classificati
```

In [3]:
```
data['Class'] = data['Class'].replace(['fishes','birds','amphibians','reptiles'],
data
```

Out[3]:

	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

We can apply Pandas cross-tabulation to examine the relationship between the Warm-blooded and Gives Birth attributes with respect to the class.

In [4]:
```
# Use crosstab to make a contingency table
```

In [5]:
```
pd.crosstab([data['Warm-blooded'],data['Gives Birth']],data['Class'])
```

Out[5]:

	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 these 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

In this section, we apply a decision tree classifier to the vertebrate dataset described in the previous subsection.

In [6]:
```
# This is different from the lab code for running a DT model. Interesting.
```

In [7]:
```
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)
```

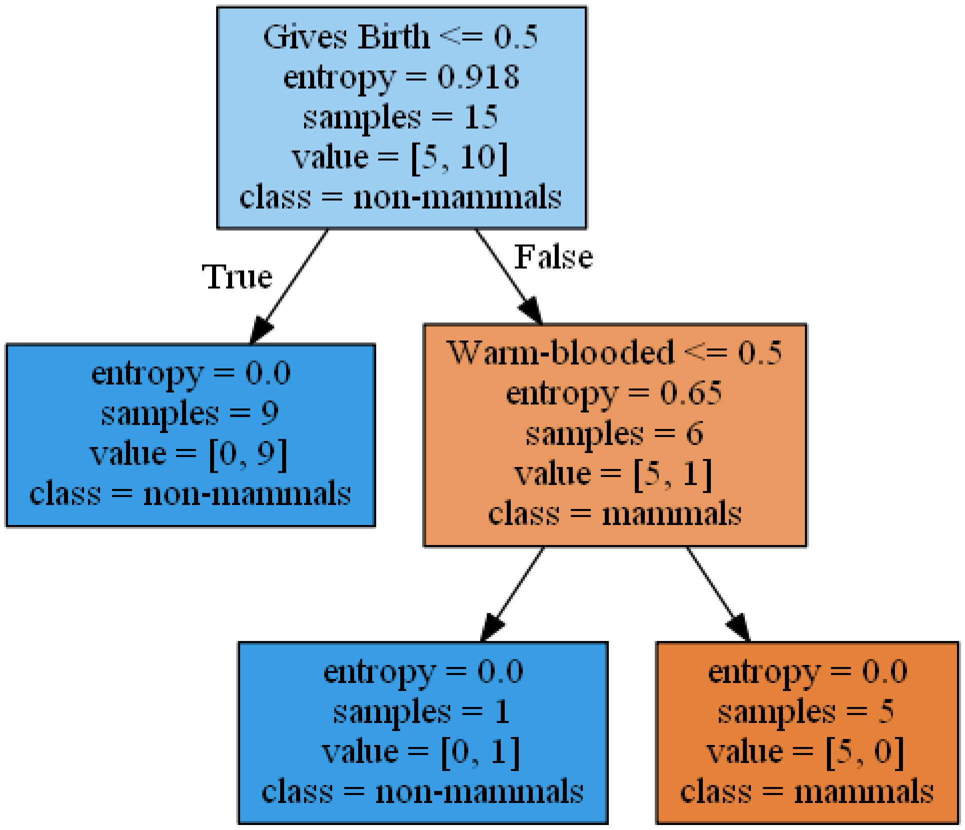
The preceding commands will extract the predictor (X) and target class (Y) attributes from the vertebrate dataset and create a decision tree classifier object using entropy as its impurity measure for splitting criterion. The decision tree class in Python sklearn library also supports using 'gini' as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

We can plot the resulting decision tree obtained after training the classifier. To do this, you must first install both graphviz (http://www.graphviz.org (http://www.graphviz.org)) and its Python interface called pydotplus (http://pydotplus.readthedocs.io/ (http://pydotplus.readthedocs.io/)).

In [18]:
```
# For the graphviz to work, I had to use pip install graphviz and conda install g
```

In [8]:
```
import pydotplus
from IPython.display import Image

dot_data = tree.export_graphviz(clf, feature_names=X.columns, class_names=['mamma
                                out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

Out[8]:



Next, suppose we apply the decision tree to classify the following test examples.

In [19]:
```
# Make a df from the tree.
```

In [20]:
```
testData = [['gila monster',0,0,0,0,1,1,'non-mammals'],
            ['platypus',1,0,0,0,1,1,'mammals'],
            ['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
```

Out[20]:

	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	mammals
2	owl	1	0	0	1	1	0	non-mammals
3	dolphin	1	1	1	0	0	0	mammals

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

In [21]:
```
# Predict using predictors from the test set.
```

In [10]:
```
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')
predictions
```

Out[10]:

	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 accuracy of the classifier on the test data as shown by the example given below.

In [22]:
```
# Accuracy = (TP + TN) / (TP + TN + FP + FN). But the accuracy.score() is new to
```

In [11]:
```
from sklearn.metrics import accuracy_score

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

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

3.3 Model Overfitting

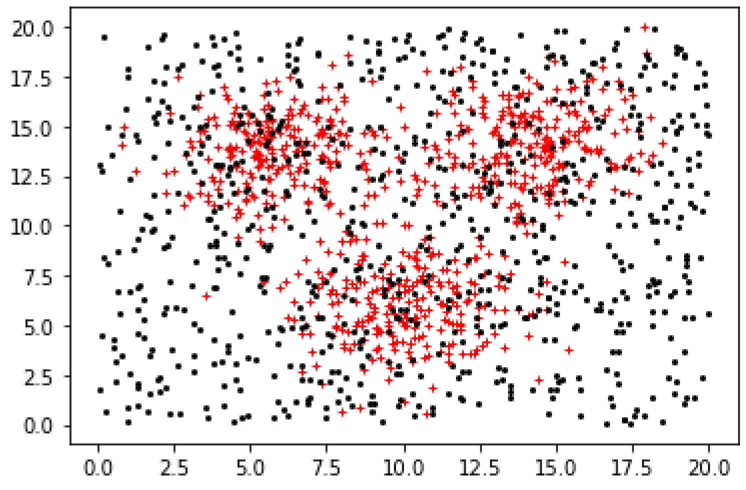
To illustrate the problem of model overfitting, we consider a two-dimensional dataset containing 1500 labeled instances, each of which is assigned to one of two classes, 0 or 1. Instances from each class are generated as follows:

1. Instances from class 1 are generated from a mixture of 3 Gaussian distributions, centered at [6,14], [10,6], and [14 14], respectively.
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.

In [12]:
```
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
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x203084fc340>,
 <matplotlib.lines.Line2D at 0x203084fc460>]
```

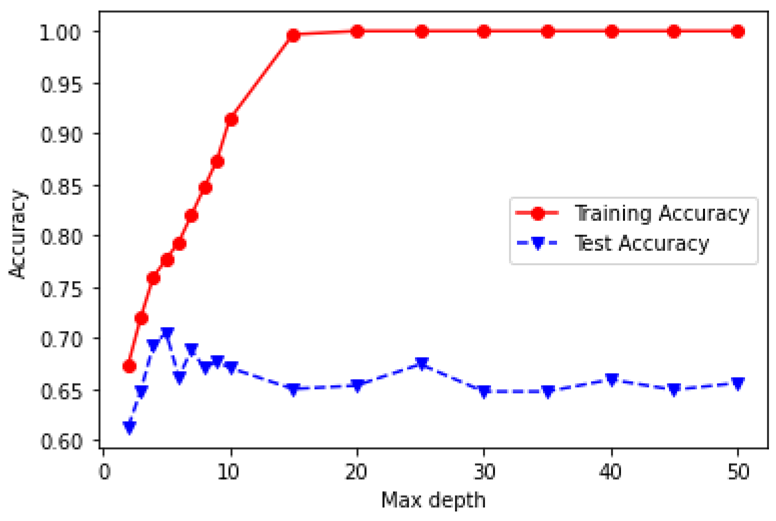


In this example, we reserve 80% of the labeled data for training and the remaining 20% for testing. We then fit decision trees of different maximum depths (from 2 to 50) to the training set and plot their respective accuracies when applied to the training and test sets.

In [23]:
```
# Very useful plot.
```

In [13]:
```
#########################################
# 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_s

from sklearn import tree
from sklearn.metrics import accuracy_score

#########################################
# Model fitting and evaluation
#########################################

maxdepths = [2,3,4,5,6,7,8,9,10,15,20,25,30,35,40,45,50]

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')
```

Out[13]:
```
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

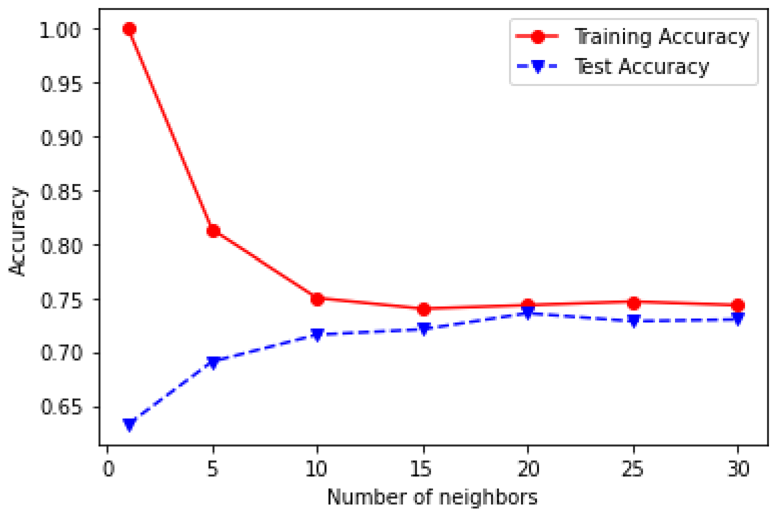
In this approach, the class label of a test instance is predicted based on the majority class of its *k* closest training instances. The number of nearest neighbors, *k*, is a hyperparameter that must be provided by the user, along with the distance metric. By default, we can use Euclidean distance (which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

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

In [24]:
```
# Same plot using KNN
```

In [14]:
```python
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')
```

Out[14]:
```
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:

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

where C is a hyperparameter that controls the inverse of model complexity (smaller values imply stronger regularization) while $\Omega(\cdot)$ 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:

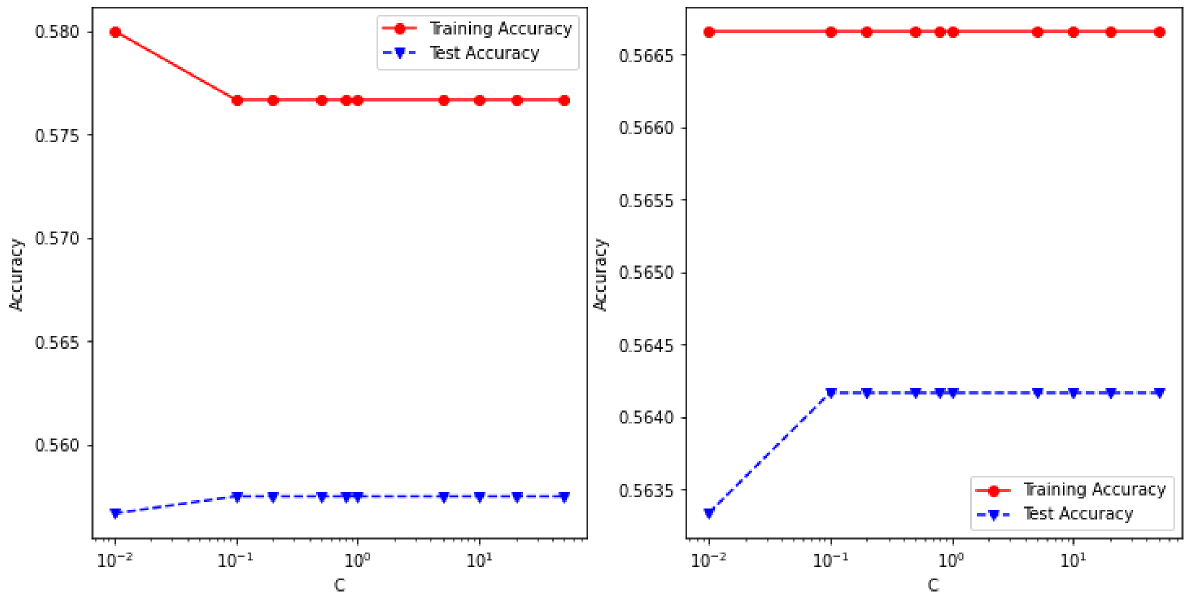
$$\min_{w^*, b^*, \{\xi_i\}} \frac{\|w\|^2}{2} + \frac{1}{C}\sum_i \xi_i$$

$$\text{s.t. } \forall i : y_i \left[w^T \phi(x_i) + b \right] \geq 1 - \xi_i, \;\; \xi_i \geq 0$$

In [25]:
```
# The plots seem complicated with the set-up of the parameters.
```

In [15]:
```
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')
```

Out[15]:
```
Text(0, 0.5, 'Accuracy')
```



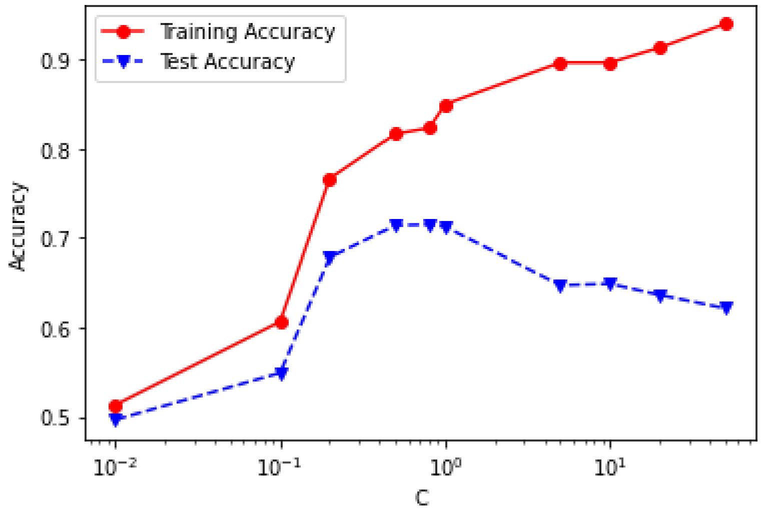
Note that linear classifiers perform poorly on the data since the true decision boundaries between classes are nonlinear for the given 2-dimensional dataset.

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.

In [16]:
```
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')
```

Out[16]:
```
Text(0, 0.5, 'Accuracy')
```



Observe that the nonlinear SVM can achieve a higher test accuracy compared to linear SVM.

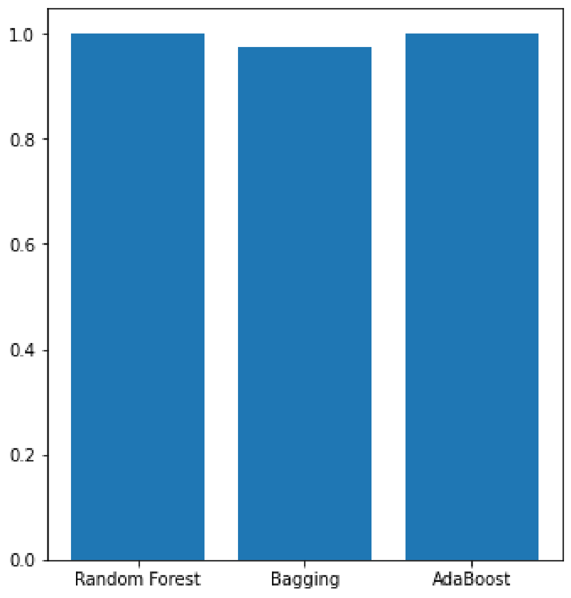
3.4.4 Ensemble Methods

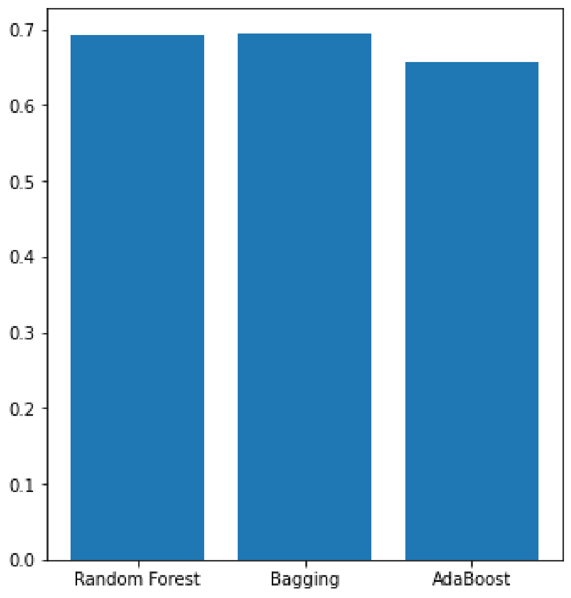
An ensemble classifier constructs a set of base classifiers from the training data and performs classification by taking a vote on the predictions made by each base classifier. We consider 3 types of ensemble classifiers in this example: bagging, boosting, and random forest. Detailed explanation about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

In [17]:
```
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_est
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_es
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)
```

Out[17]:
```
[Text(1.5, 0, 'Random Forest'),
 Text(2.5, 0, 'Bagging'),
 Text(3.5, 0, 'AdaBoost')]
```





3.5 Summary

This section provides several examples of using Python sklearn library to build classification models from a given input data. We also illustrate the problem of model overfitting and show how to apply different classification methods to the given dataset.

In [26]:
```
# It does seem like R has the natrual advantage of visualizations compared to Pyt
```

In [ ]:
```
```