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.

6.1 Feature Selection and class-imbalance

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

Out[2]:

	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 [3]:
```
pd.crosstab([data['Warm-blooded'],data['Gives Birth']],data['Class'])
```

Out[3]:

	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.

6.1.2 Decision Tree Classifier

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

In [4]:
```
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.

Plotting the resulting decision tree obtained after training the classifier, 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/)).

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

In [5]:
```
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[5]:

	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 [6]:
```
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
```

Out[6]:

	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 [7]:
```
from sklearn.metrics import accuracy_score

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

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

6.1.3 Feature Selection

特征选择(Feature Selection)也称特征子集选择(Feature Subset Selection , FSS)，或属性选择(Attribute Selection)。是指从已有的M个特征(Feature)中选择N个特征使得系统的特定指标最优化，是从原始特征中选择出一些最有效特征以降低数据集维度的过程,是提高学习算法性能的一个重要手段,也是模式识别中关键的数据预处理步骤。因此，当我们拿到数据集时，对数据进行预处理，会使我们之后建立的模型更精确。

6.1.3.1 方差过滤法

VarianceThreshold 是特征选择的一个简单基本方法,其原理在于–底方差的特征的预测效果往往不好。而VarianceThreshold会移除所有那些方差不满足一些阈值的特征。默认情况下，它将会移除所有的零方差特征，即那些在所有的样本上的取值均不变的特征。

sklearn中的VarianceThreshold类中重要参数 threshold（方差的阈值），表示删除所有方差小于threshold的特征 #不填默认为0——删除所有记录相同的特征。

In [8]:
```
import pandas as pd
import numpy as np
np.random.seed(1) #设置随机种子，实现每次生成的随机数矩阵都一样
a= np.random.randint(0, 200,10)
b= np.random.randint(0, 200,10)
c= np.random.randint(0, 200,10)
d= [9,9,9,9,9,9,9,9,9,9]
data=pd.DataFrame({"A" : a,"B" : b,"C" : c,"D" : d})
data
from sklearn.feature_selection import VarianceThreshold
sel_model = VarianceThreshold(threshold = 0)

#删除不合格特征之后的新矩阵
sel_model.fit_transform(data)
```

Out[8]:
```
array([[ 37, 134,  50],
       [140,  25,  68],
       [ 72, 178,  96],
       [137,  20,  86],
       [133, 101, 141],
       [ 79, 146, 137],
       [192, 139,   7],
       [144, 156,  63],
       [129, 157,  61],
       [ 71, 142,  22]], dtype=int64)
```

In [9]:
```
## 方差过滤法用于Vertebrate Dataset
data = pd.read_csv('vertebrate.csv',header='infer')
X = data.drop(['Name','Class'],axis=1)
sel_model.fit_transform(X)
```

Out[9]:
```
array([[1, 1, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 0, 0],
       [1, 1, 1, 0, 0, 0],
       [0, 0, 1, 0, 1, 1],
       [0, 0, 0, 0, 1, 0],
       [1, 1, 0, 1, 1, 1],
       [1, 0, 0, 1, 1, 0],
       [1, 1, 0, 0, 1, 0],
       [0, 1, 1, 0, 0, 0],
       [0, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0],
       [1, 1, 0, 0, 1, 1],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 1, 0, 1, 1]], dtype=int64)
```

6.1.3.2 相关性过滤法之k方检验

卡方检验就是统计样本的实际观测值与理论推断值之间的偏离程度，实际观测值与理论推断值之间的偏离程度就决定卡方值的大小，如果卡方值越大，二者偏差程度越大；反之，二者偏差越小；若两个值完全相等时，卡方值就为0，表明理论值完全符合。

原理：计算每个非负特征与标签之间的卡方值，每一个卡方值对应一个p值，用p值判断特征和标签之间的相关性。大众经验，p值选择0.05或0.1。如P<=0.05就说明两组数据相关。

适用范围：离散型标签（即适用分类问题），只能捕捉线性相关。

In [10]:
```
import pandas as pd
import numpy as np
np.random.seed(7)
a= np.random.randint(0, 200,10)
b= np.random.randint(0, 200,10)
c= np.random.randint(0, 200,10)
d= [9,9,9,19,9,9,10,9,9,9]
x=pd.DataFrame({"A" : a,"B" : b,"C" : c,"D" : d})
y=[0,0,1,0,1,1,0,1,0,1]
from sklearn.feature_selection import chi2
#卡方值
chivalue,pvalue = chi2(x,y)
#确保要保留几个特征
k=chivalue.shape[0]-((pvalue>0.05).sum())
# 结合SelectKBest筛选出来过滤特征后的数据集
from sklearn.feature_selection import SelectKBest
data_new=SelectKBest(chi2,k=k).fit_transform(x,y)
#删除不合格特征之后的新矩阵
data_new
```

Out[10]:
```
array([[175,  72, 172],
       [196,  89,   0],
       [ 25, 110,  75],
       [ 67,  42,  55],
       [151, 136,   6],
       [103, 167,  19],
       [ 92,  68, 188],
       [185, 176,  44],
       [142, 127, 191],
       [ 23, 135,  69]], dtype=int64)
```

In [11]:
```
## 相关性过滤法用于Vertebrate Dataset
data = pd.read_csv('vertebrate.csv',header='infer')
data['Class'] = data['Class'].replace(['fishes','birds','amphibians','reptiles'],'non-mammals')
Y = data['Class']
X = data.drop(['Name','Class'],axis=1)
chivalue,pvalue = chi2(X,Y)
#确保要保留几个特征
k=chivalue.shape[0]-((pvalue>0.05).sum())
# 结合SelectKBest筛选出来过滤特征后的数据集
from sklearn.feature_selection import SelectKBest
data_new=SelectKBest(chi2,k=k).fit_transform(X,Y)
#删除不合格特征之后的新矩阵
data_new
```

Out[11]:
```
array([[1, 1],
       [0, 0],
       [0, 0],
       [1, 1],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [1, 1],
       [0, 1],
       [0, 0],
       [1, 0],
       [1, 1],
       [0, 0],
       [0, 0]], dtype=int64)
```

In [12]:
```
X
```

Out[12]:

	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates
0	1	1	0	0	1	0
1	0	0	0	0	0	1
2	0	0	1	0	0	0
3	1	1	1	0	0	0
4	0	0	1	0	1	1
5	0	0	0	0	1	0
6	1	1	0	1	1	1
7	1	0	0	1	1	0
8	1	1	0	0	1	0
9	0	1	1	0	0	0
10	0	0	1	0	1	0
11	1	0	1	0	1	0
12	1	1	0	0	1	1
13	0	0	1	0	0	0
14	0	0	1	0	1	1

6.1.3.3 相关性过滤之互信息法

互信息法返回每个特征与标签之间的互信息量的统计，值越大越相关 。0表示特征和标签完全独立。

回归实现方式（连续型标签，互信息回归feature_selection.mutual_info_regression）

分类实现方式（离散型标签，互信息分类feature_selection.mutual_info_classif）

In [13]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif
data = pd.read_csv('vertebrate.csv',header='infer')
data['Class'] = data['Class'].replace(['fishes','birds','amphibians','reptiles'],'non-mammals')
Y = data['Class']
X = data.drop(['Name','Class'],axis=1)
#返回每个特征与标签的互信息统计量
x_y_result = mutual_info_classif(X,Y)
x_y_result
#筛选互信息量最大的2个特征
select = SelectKBest(score_func=chi2, k=2)
# 拟合数据
z = select.fit_transform(X,Y)
filter_1 = select.get_support()
print("所有的特征: ", X.columns.values)
print("筛选出来最优的特征是: ", X.columns.values[filter_1])
```

```
所有的特征:  ['Warm-blooded' 'Gives Birth' 'Aquatic Creature' 'Aerial Creature'
 'Has Legs' 'Hibernates']
筛选出来最优的特征是:  ['Warm-blooded' 'Gives Birth']
```

6.1.4 class-imbalance

样本（类别）样本不平衡（class-imbalance）指的是分类任务中不同类别的训练样例数目差别很大的情况，一般地，样本类别比例（多数类vs少数类）明显大于1:1（如4：1）就可以归为样本不均衡的问题。现实中，样本不平衡是一种常见的现象，如：金融欺诈交易检测，欺诈交易的订单样本通常是占总交易数量的极少部分，而且对于有些任务而言少数样本更为重要。

6.1.4.1 Naive random over-sampling

One way to fight this issue is to generate new samples in the classes which are under-represented. The most naive strategy is to generate new samples by randomly sampling with replacement the current available samples.

In [16]:
```
from collections import Counter
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=5000, n_features=2, n_informative=2,
                           n_redundant=0, n_repeated=0, n_classes=3,
                           n_clusters_per_class=1,
                           weights=[0.01, 0.05, 0.94],
                           class_sep=0.8, random_state=0)
print(sorted(Counter(y).items()))
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)
from collections import Counter
print(sorted(Counter(y_resampled).items()))
```

```
[(0, 64), (1, 262), (2, 4674)]
[(0, 4674), (1, 4674), (2, 4674)]
```

The augmented data set should be used instead of the original data set to train a classifier:

In addition, RandomOverSampler allows to sample heterogeneous data (e.g. containing some strings):

In [17]:
```
import numpy as np
X_hetero = np.array([['xxx', 1, 1.0], ['yyy', 2, 2.0], ['zzz', 3, 3.0]],
                    dtype=object)
y_hetero = np.array([0, 0, 1])
X_resampled, y_resampled = ros.fit_resample(X_hetero, y_hetero)
print(X_resampled)
print(y_resampled)
```

```
[['xxx' 1 1.0]
 ['yyy' 2 2.0]
 ['zzz' 3 3.0]
 ['zzz' 3 3.0]]
[0 0 1 1]
```

It would also work with pandas dataframe:

In [18]:
```
from sklearn.datasets import fetch_openml
df_adult, y_adult = fetch_openml('adult', version=2, as_frame=True, return_X_y=True)
df_adult.head()
df_resampled, y_resampled = ros.fit_resample(df_adult, y_adult)
df_resampled.head()
```

Out[18]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	25.0	Private	226802.0	11th	7.0	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	United-States
1	38.0	Private	89814.0	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	United-States
2	28.0	Local-gov	336951.0	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	United-States
3	44.0	Private	160323.0	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688.0	0.0	40.0	United-States
4	18.0	NaN	103497.0	Some-college	10.0	Never-married	NaN	Own-child	White	Female	0.0	0.0	30.0	United-States

6.1.4.2. From random over-sampling to SMOTE and ADASYN

Apart from the random sampling with replacement, there are two popular methods to over-sample minority classes: (i) the Synthetic Minority Oversampling Technique (SMOTE) [CBHK02] and (ii) the Adaptive Synthetic (ADASYN) [HBGL08] sampling method. These algorithms can be used in the same manner:

In [20]:
```
from imblearn.over_sampling import SMOTE, ADASYN
from sklearn.svm import LinearSVC
X_resampled, y_resampled = SMOTE().fit_resample(X, y)
print(sorted(Counter(y_resampled).items()))

clf_smote = LinearSVC().fit(X_resampled, y_resampled)
X_resampled, y_resampled = ADASYN().fit_resample(X, y)
print(sorted(Counter(y_resampled).items()))

clf_adasyn = LinearSVC().fit(X_resampled, y_resampled)
```

```
[(0, 4674), (1, 4674), (2, 4674)]
[(0, 4673), (1, 4662), (2, 4674)]
```

6.1.4.3 Under-sampling

In [21]:
```
from collections import Counter
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=5000, n_features=2, n_informative=2,
                           n_redundant=0, n_repeated=0, n_classes=3,
                           n_clusters_per_class=1,
                           weights=[0.01, 0.05, 0.94],
                           class_sep=0.8, random_state=0)
print(sorted(Counter(y).items()))
from imblearn.under_sampling import ClusterCentroids
cc = ClusterCentroids(random_state=0)
X_resampled, y_resampled = cc.fit_resample(X, y)
print(sorted(Counter(y_resampled).items()))
```

```
[(0, 64), (1, 262), (2, 4674)]

D:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans is known to have a memory leak on W
indows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_TH
READS=2.
  warnings.warn(

[(0, 64), (1, 64), (2, 64)]
```

RandomUnderSampler is a fast and easy way to balance the data by randomly selecting a subset of data for the targeted classes:

In [22]:
```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=0)
X_resampled, y_resampled = rus.fit_resample(X, y)
print(sorted(Counter(y_resampled).items()))
```

```
[(0, 64), (1, 64), (2, 64)]
```

RandomUnderSampler allows to bootstrap the data by setting replacement to True. The resampling with multiple classes is performed by considering independently each targeted class:

In [23]:
```
import numpy as np
print(np.vstack([tuple(row) for row in X_resampled]).shape)

rus = RandomUnderSampler(random_state=0, replacement=True)
X_resampled, y_resampled = rus.fit_resample(X, y)
print(np.vstack(np.unique([tuple(row) for row in X_resampled], axis=0)).shape)
```

```
(192, 2)
(181, 2)
```

In addition, RandomUnderSampler allows to sample heterogeneous data (e.g. containing some strings):

In [24]:
```
X_hetero = np.array([['xxx', 1, 1.0], ['yyy', 2, 2.0], ['zzz', 3, 3.0]],
                    dtype=object)
y_hetero = np.array([0, 0, 1])
X_resampled, y_resampled = rus.fit_resample(X_hetero, y_hetero)
print(X_resampled)


print(y_resampled)
```

```
[['xxx' 1 1.0]
 ['zzz' 3 3.0]]
[0 1]
```

It would also work with pandas dataframe:

In [25]:
```
from sklearn.datasets import fetch_openml
df_adult, y_adult = fetch_openml(
    'adult', version=2, as_frame=True, return_X_y=True)
df_adult.head()
df_resampled, y_resampled = rus.fit_resample(df_adult, y_adult)
df_resampled.head()
```

Out[25]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	29.0	Private	201101.0	HS-grad	9.0	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0.0	0.0	50.0	United-States
1	23.0	Private	188950.0	Assoc-voc	11.0	Never-married	Sales	Own-child	White	Male	0.0	0.0	40.0	United-States
2	24.0	Private	282604.0	Some-college	10.0	Married-civ-spouse	Protective-serv	Other-relative	White	Male	0.0	0.0	24.0	United-States
3	29.0	Private	174419.0	HS-grad	9.0	Never-married	Other-service	Unmarried	White	Female	0.0	0.0	30.0	United-States
4	20.0	Private	236592.0	12th	8.0	Never-married	Prof-specialty	Not-in-family	White	Female	0.0	0.0	35.0	Italy

6.2 Model building and evaluation

6.2.1 Model Overfitting

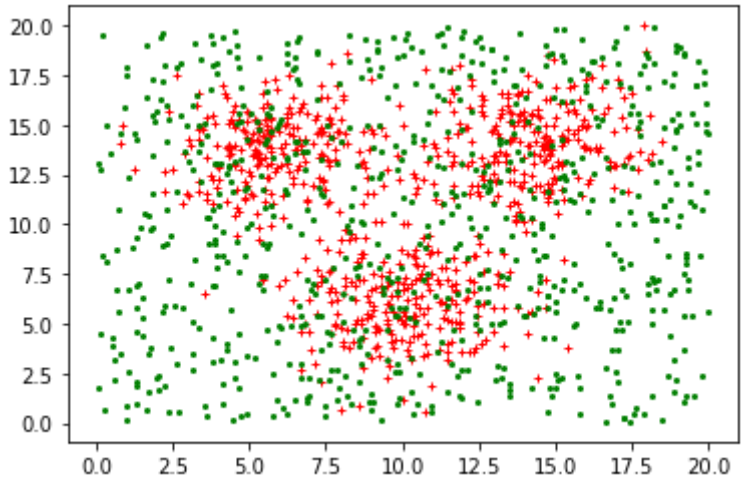
To illustrate the problem of model overfitting, we consider a two-dimensional dataset containing 1500 labeled instances, each of which is assigned to one of two classes, 0 or 1. Instances from each class are generated as follows:

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 [26]:
```
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],'g.',ms=4)
```

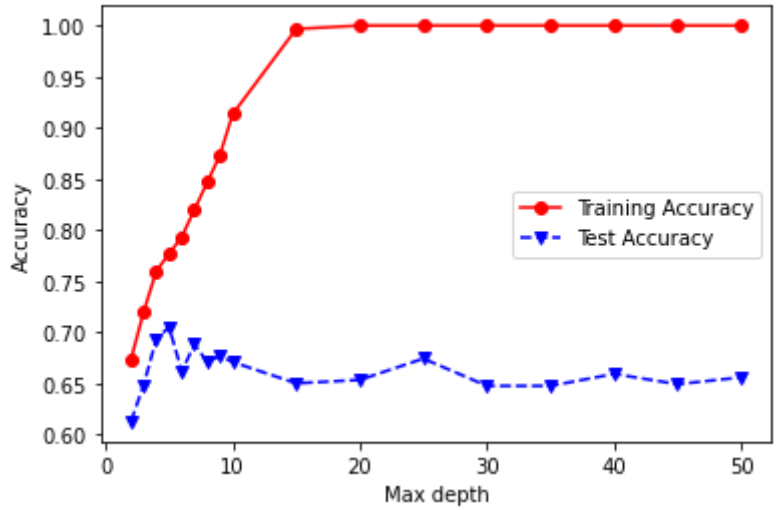
Out[26]:
```
[<matplotlib.lines.Line2D at 0x192161f4a90>,
 <matplotlib.lines.Line2D at 0x192161f4bb0>]
```



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 [27]:
```
#########################################
# 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 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[27]:
```
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.

6.2.2 Model evaluation

6.2.2.1 accuracy and confusion matrix

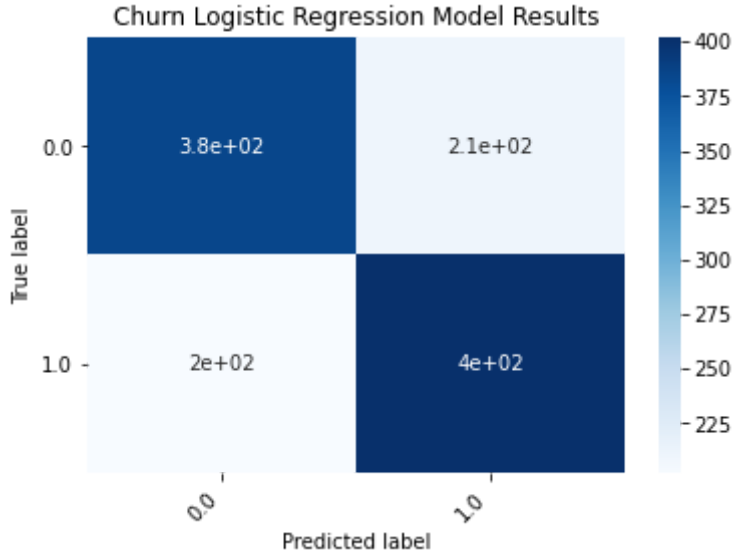
In [28]:
```
from sklearn.metrics import accuracy_score
print("Accuracy: ", accuracy_score(Y_test, Y_predTest))
```

```
Accuracy:  0.6558333333333334
```

In [29]:
```
from sklearn.metrics import confusion_matrix
conmat = confusion_matrix(Y_test, Y_predTest)
val = np.mat(conmat)
classnames = list(set(Y_train))
df_cm = pd.DataFrame(
val, index=classnames, columns=classnames,
)
print(df_cm)
```

```
     0.0  1.0
0.0  385  211
1.0  202  402
```

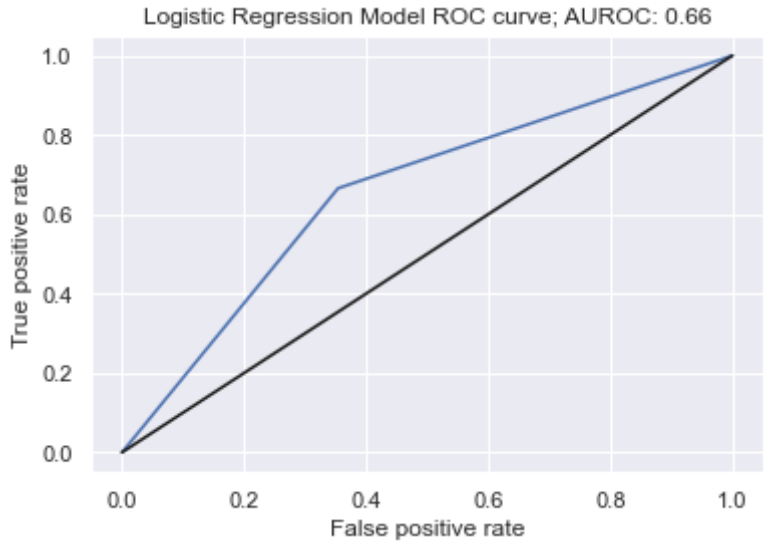
In [30]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure()
heatmap = sns.heatmap(df_cm, annot=True, cmap="Blues")
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right')
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title('Churn Logistic Regression Model Results')
plt.show()
```



6.2.2.2 ROC and AUROC

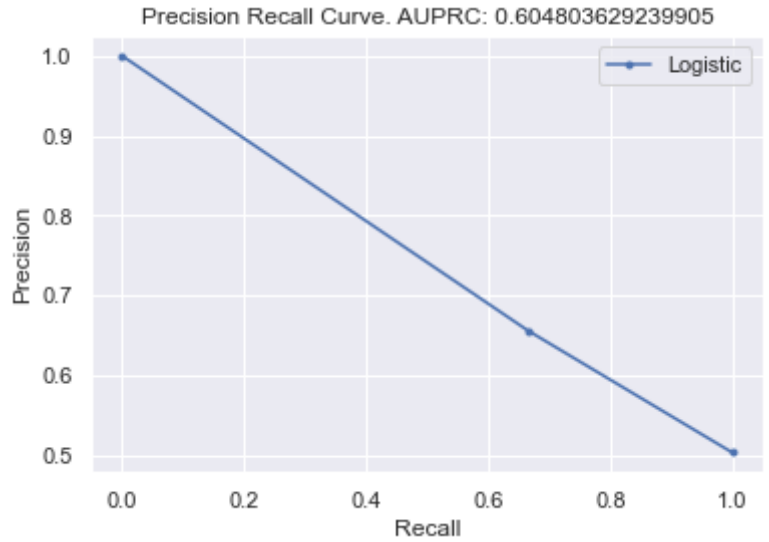
In [31]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = clf.predict_proba(np.array(X_test))[:,1]
fpr, tpr, thresholds = roc_curve(Y_test, y_pred_proba)
```

In [32]:
```
sns.set()
plt.plot(fpr, tpr)
plt.plot(fpr, fpr, linestyle = '-', color = 'k')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
AUROC = np.round(roc_auc_score(Y_test, y_pred_proba), 2)
plt.title(f'Logistic Regression Model ROC curve; AUROC: {AUROC}');
plt.show()
```



6.2.2.3 AUPRC

In [33]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
average_precision = average_precision_score(Y_test, y_pred_proba)
precision, recall, thresholds = precision_recall_curve(Y_test, y_pred_proba)
plt.plot(recall, precision, marker='.', label='Logistic')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
plt.title(f'Precision Recall Curve. AUPRC: {average_precision}')
plt.show()
```



6.3 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.

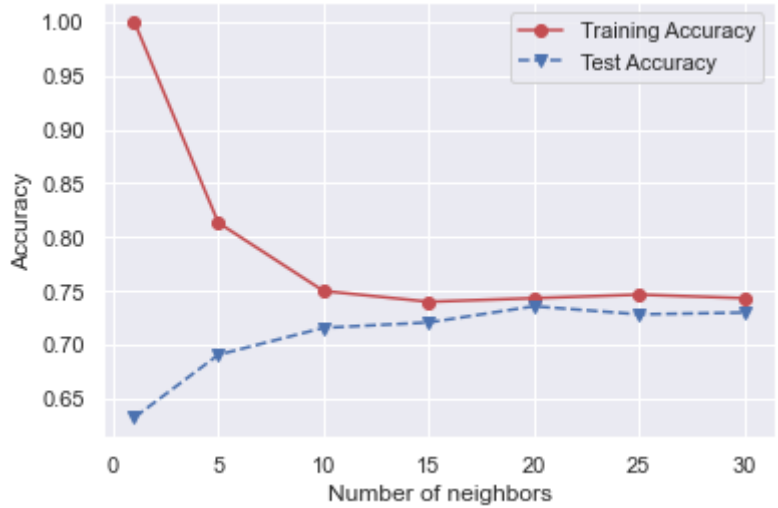
6.3.1 K-Nearest neighbor classifier

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 [34]:
```
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[34]:
```
Text(0, 0.5, 'Accuracy')
```



6.3.2 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.

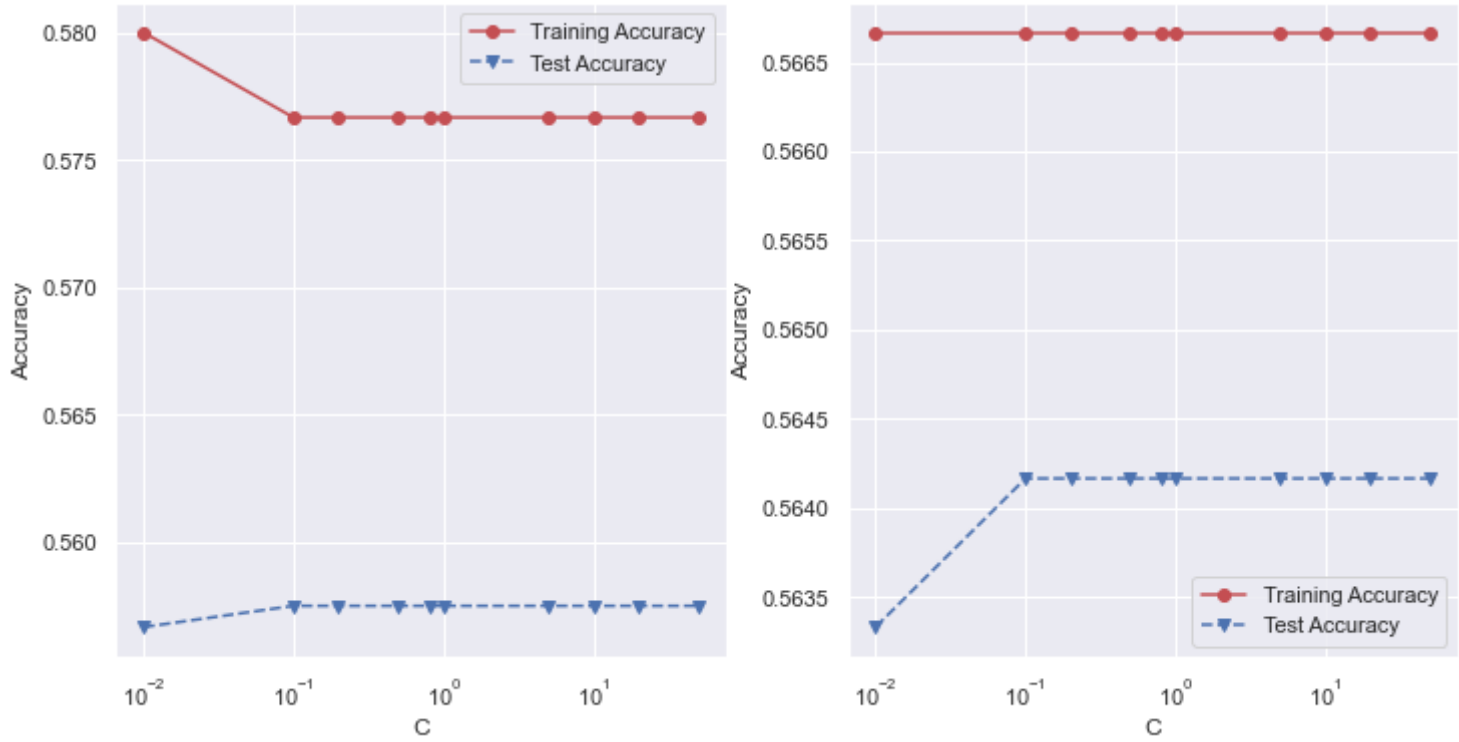
For support vector machine, the model parameters (w^*, b^*) are estimated by solving the following constrained optimization problem:

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

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

In [35]:
```
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[35]:
```
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.

6.3.3 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 [36]:
```
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[36]:
```
Text(0, 0.5, 'Accuracy')
```

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

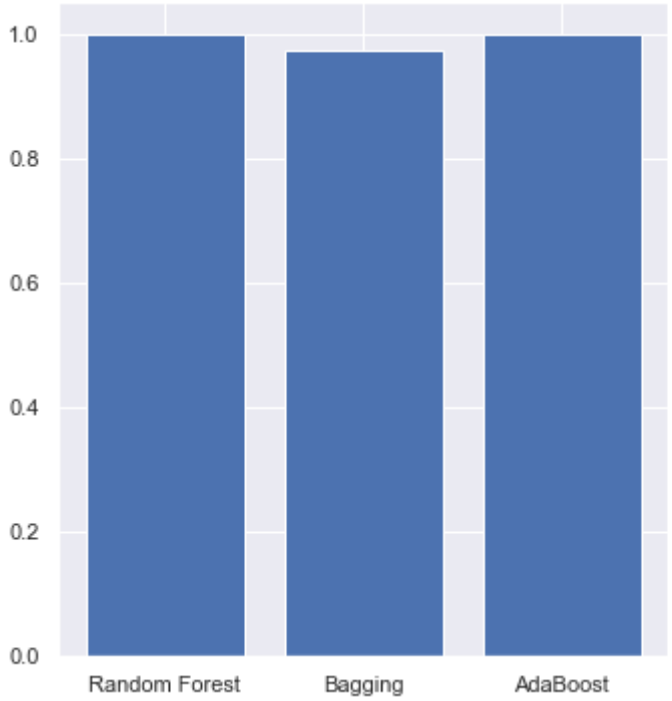
6.3.4 Ensemble Methods

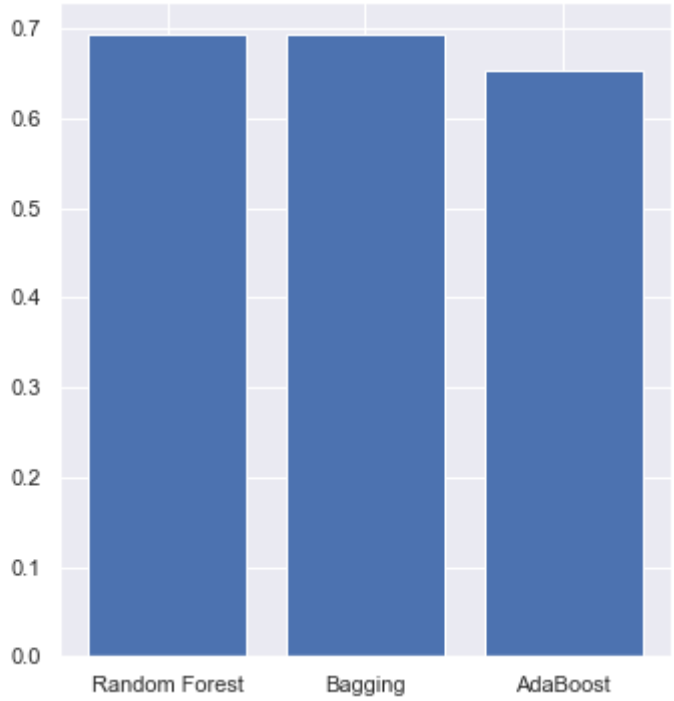
An ensemble classifier constructs a set of base classifiers from the training data and performs classification by taking a vote on the predictions made by each base classifier. We consider 3 types of ensemble classifiers in this example: bagging, boosting, and random forest. Detailed explanation about 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 [37]:
```
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)
```

Out[37]:
```
[Text(1.5, 0, 'Random Forest'),
 Text(2.5, 0, 'Bagging'),
 Text(3.5, 0, 'AdaBoost')]
```





6.4 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 [ ]: