In [1]:

```
import pandas as pd
import numpy as np


#*************************************************#
#
#    Decision Tree, Random Forest & Gradient Boosting
#
#*************************************************#


######################################
#>>> 1> Read file

titanic = pd.read_csv('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt')

print(titanic.head())#Pre-look at titanic as a dataframe
```

```
   row.names pclass  survived  \
0          1    1st         1
1          2    1st         0
2          3    1st         0
3          4    1st         0
4          5    1st         1

                                              name      age     embarked  \
0                     Allen, Miss Elisabeth Walton  29.0000  Southampton
1                      Allison, Miss Helen Loraine   2.0000  Southampton
2              Allison, Mr Hudson Joshua Creighton  30.0000  Southampton
3  Allison, Mrs Hudson J.C. (Bessie Waldo Daniels)  25.0000  Southampton
4                    Allison, Master Hudson Trevor   0.9167  Southampton

                         home.dest room      ticket   boat     sex
0                     St Louis, MO  B-5  24160 L221      2  female
1  Montreal, PQ / Chesterville, ON  C26         NaN    NaN  female
2  Montreal, PQ / Chesterville, ON  C26         NaN  (135)    male
3  Montreal, PQ / Chesterville, ON  C26         NaN    NaN  female
4  Montreal, PQ / Chesterville, ON  C22         NaN     11    male
```

In [2]:

```
print(titanic.info())#Look at titanic statistic information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1313 entries, 0 to 1312
Data columns (total 11 columns):
row.names    1313 non-null int64
pclass       1313 non-null object
survived     1313 non-null int64
name         1313 non-null object
age          633 non-null float64
embarked     821 non-null object
home.dest    754 non-null object
room         77 non-null object
ticket       69 non-null object
boat         347 non-null object
sex          1313 non-null object
dtypes: float64(1), int64(2), object(8)
memory usage: 112.9+ KB
None
```

In [3]:

```
######################################
#>>> 2> Select features and Vectorize

X = titanic[['pclass','age','sex']]
y = titanic['survived']


print(X.info()) # Check features information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1313 entries, 0 to 1312
Data columns (total 3 columns):
pclass    1313 non-null object
age       633 non-null float64
sex       1313 non-null object
dtypes: float64(1), object(2)
memory usage: 30.9+ KB
None
```

In [4]:

```
X['age'].fillna(X['age'].mean(),inplace=True)
print(X.info()) # Check again after filling out
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1313 entries, 0 to 1312
Data columns (total 3 columns):
pclass    1313 non-null object
age       1313 non-null float64
sex       1313 non-null object
dtypes: float64(1), object(2)
memory usage: 30.9+ KB
None
```

```
C:\Program Files\Anaconda3\lib\site-packages\pandas\core\generic.py:3191: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/in
dexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/i
ndexing.html#indexing-view-versus-copy)
  self._update_inplace(new_data)
```

In [5]:

```
from sklearn.cross_validation import train_test_split
from sklearn.feature_extraction import DictVectorizer

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=33) #Split data

vec = DictVectorizer(sparse=False)#Vectorize
X_train=vec.fit_transform(X_train.to_dict(orient='record'))
print(vec.feature_names_)
X_test=vec.transform(X_test.to_dict(orient='record'))
```

```
['age', 'pclass=1st', 'pclass=2nd', 'pclass=3rd', 'sex=female', 'sex=male']
```

In [7]:

```
######################################
#>>> 3> Build model and Predict

# Decision Tree
from sklearn.tree import DecisionTreeClassifier

dtc=DecisionTreeClassifier()
dtc.fit(X_train,y_train)
y_predict=dtc.predict(X_test)

# Random Forest
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(X_train,y_train)
rfc_y_pred=rfc.predict(X_test)

# Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
gbc=GradientBoostingClassifier()
gbc.fit(X_train,y_train)
gbc_y_pred=gbc.predict(X_test)

######################################
#>>> 4> Score
from sklearn.metrics import classification_report

# Decision Tree
print('The accuracy of Decision Tree is',dtc.score(X_test,y_test))
print(classification_report(y_predict,y_test))

# Random Forest
print('The accuracy of Random Forest Classifier is',rfc.score(X_test,y_test))
print(classification_report(rfc_y_pred,y_test))

# Gradient Boosting
print('The accuracy of Gradient Boosting Classifier is',gbc.score(X_test,y_test))
print(classification_report(gbc_y_pred,y_test))
```

```
The accuracy of Decision Tree is 0.781155015198
             precision    recall  f1-score   support

          0       0.91      0.78      0.84       236
          1       0.58      0.80      0.67        93

avg / total       0.81      0.78      0.79       329

The accuracy of Random Forest Classifier is 0.781155015198
             precision    recall  f1-score   support

          0       0.90      0.78      0.83       234
          1       0.59      0.79      0.68        95

avg / total       0.81      0.78      0.79       329

The accuracy of Gradient Boosting Classifier is 0.790273556231
             precision    recall  f1-score   support

          0       0.92      0.78      0.84       239
          1       0.58      0.82      0.68        90

avg / total       0.83      0.79      0.80       329
```

In [8]:

```
#*********************************************#
#
#    Logistic Regression & SGD Regression
#
#*********************************************#


#####################################
#>>> 1> Read file

from sklearn.datasets import load_boston
boston=load_boston()
print(boston.DESCR)
```

Boston House Prices dataset

Notes

Data Set Characteristics:

:Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive

:Median Value (attribute 14) is usually the target

:Attribute Information (in order):

```
    - CRIM     per capita crime rate by town
    - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
    - INDUS    proportion of non-retail business acres per town
    - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherw
ise)
    - NOX      nitric oxides concentration (parts per 10 million)
    - RM       average number of rooms per dwelling
    - AGE      proportion of owner-occupied units built prior to 1940
    - DIS      weighted distances to five Boston employment centres
    - RAD      index of accessibility to radial highways
    - TAX      full-value property-tax rate per $10,000
    - PTRATIO  pupil-teacher ratio by town
    - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
    - LSTAT    % lower status of the population
    - MEDV     Median value of owner-occupied homes in $1000's
```

:Missing Attribute Values: None

:Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
http://archive.ics.uci.edu/ml/datasets/Housing (http://archive.ics.uci.edu/ml/datasets/Housing)

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980. N.B. Various transformations are used in the table on pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

References

- Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.
- Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
- many more! (see http://archive.ics.uci.edu/ml/datasets/Housing) (http://archive.ics.uci.edu/ml/datasets/Housing))

In [9]:

```
#####################################
#>>> 2> Split data and preprocessing

# Split data
from sklearn.cross_validation import train_test_split

X=boston.data
y=boston.target

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=33) #Split data
print('The max target value is', np.max(boston.target))
print('The min target value is', np.min(boston.target))
print('The average target value is', np.mean(boston.target))

# preprocessing
from sklearn.preprocessing import StandardScaler

ss_X=StandardScaler()
ss_y=StandardScaler()

X_train=ss_X.fit_transform(X_train)
X_test=ss_X.fit_transform(X_test)

y_train=ss_y.fit_transform(y_train)
y_test=ss_y.fit_transform(y_test)
```

```
The max target value is 50.0
The min target value is 5.0
The average target value is 22.5328063241
```

```
C:\Program Files\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:583: Depr
ecationWarning: Passing 1d arrays as data is deprecated in 0.17 and will raise Value
Error in 0.19. Reshape your data either using X.reshape(-1, 1) if your data has a si
ngle feature or X.reshape(1, -1) if it contains a single sample.
  warnings.warn(DEPRECATION_MSG_1D, DeprecationWarning)
C:\Program Files\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:646: Depr
ecationWarning: Passing 1d arrays as data is deprecated in 0.17 and will raise Value
Error in 0.19. Reshape your data either using X.reshape(-1, 1) if your data has a si
ngle feature or X.reshape(1, -1) if it contains a single sample.
  warnings.warn(DEPRECATION_MSG_1D, DeprecationWarning)
C:\Program Files\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:583: Depr
ecationWarning: Passing 1d arrays as data is deprecated in 0.17 and will raise Value
Error in 0.19. Reshape your data either using X.reshape(-1, 1) if your data has a si
ngle feature or X.reshape(1, -1) if it contains a single sample.
  warnings.warn(DEPRECATION_MSG_1D, DeprecationWarning)
C:\Program Files\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:646: Depr
ecationWarning: Passing 1d arrays as data is deprecated in 0.17 and will raise Value
Error in 0.19. Reshape your data either using X.reshape(-1, 1) if your data has a si
ngle feature or X.reshape(1, -1) if it contains a single sample.
  warnings.warn(DEPRECATION_MSG_1D, DeprecationWarning)
```

In [10]:

```
######################################
#>>> 3> Build model and Predict

# Linear Regression
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(X_train,y_train)
lr_y_predict=lr.predict(X_test)

# SGDRegressor
from sklearn.linear_model import SGDRegressor
sgdr=SGDRegressor()
sgdr.fit(X_train,y_train)
sgdr_y_pred=sgdr.predict(X_test)

######################################
#>>> 4> Score

# Linear Regression
print('The value of default measurement of LinearRegression is',lr.score(X_test,y_test))

from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error

# R-squared
print('The value of R-squared of LinearRegression is',r2_score(y_test,lr_y_predict))

# Mean squared error
print('The value of mean squared error of LinearRegression is',mean_squared_error(ss_y.inverse_trans

# Mean absolute error
print('The value of mean absolute error of LinearRegression is',mean_absolute_error(ss_y.inverse_tra
```

```
The value of default measurement of LinearRegression is 0.676930350524
The value of R-squared of LinearRegression is 0.676930350524
The value of mean squared error of LinearRegression is 25.0512388542
The value of mean absolute error of LinearRegression is 3.51371562676
```

In [45]:

```
#**********************************************#
#
#          SVM
#
#**********************************************#



######################################
#>>> 1> Read file

titanic = pd.read_csv('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt')

######################################
#>>> 2> Vectorize

X = titanic.drop(['name','row.names','survived'],axis=1)
y = titanic['survived']

#fill missing value
X['age'].fillna(X['age'].mean(),inplace=True)
X.fillna('UNKNOWN',inplace=True)

from sklearn.cross_validation import train_test_split
from sklearn.feature_extraction import DictVectorizer

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=33) #Split data

vec = DictVectorizer()

X_train=vec.fit_transform(X_train.to_dict(orient='record'))
print(len(vec.feature_names_))
X_test=vec.transform(X_test.to_dict(orient='record'))
```

474

In [46]:

```
######################################
#>>> 3> Build model

# Decision Tree
from sklearn.tree import DecisionTreeClassifier

dt=DecisionTreeClassifier(criterion='entropy')
dt.fit(X_train,y_train)
print(dt.score(X_test,y_test))
```

0.823708206687

In [13]:

```
########################################
#>>> 4> Select feature and predict
from sklearn import feature_selection

fs=feature_selection.SelectPercentile(feature_selection.chi2,percentile=20)

X_train_fs=fs.fit_transform(X_train,y_train)
dt.fit(X_train_fs,y_train)

X_test_fs=fs.transform(X_test)
print(dt.score(X_test_fs,y_test))
```

```
0.817629179331
```

In [47]:

```
########################################
#>>> 5> Validation

from sklearn.cross_validation import cross_val_score
percentiles=range(1,100,2)
results=[]
for i in percentiles:
    fs=feature_selection.SelectPercentile(feature_selection.chi2,percentile=i)
    X_train_fs=fs.fit_transform(X_train,y_train)
    scores=cross_val_score(dt,X_train_fs,y_train,cv=5)
    results=np.append(results,scores.mean())
print(results)
```

```
[ 0.85063904  0.85673057  0.87501546  0.88622964  0.86590394  0.86998557
  0.86896516  0.87302618  0.86591424  0.87097506  0.86589363  0.86691404
  0.86795506  0.86386312  0.87097506  0.86590394  0.87199546  0.86691404
  0.86486291  0.87198516  0.8597918   0.86791383  0.86792414  0.87096475
  0.86894455  0.87199546  0.87097506  0.87404659  0.87198516  0.86893424
  0.86996496  0.86385281  0.87098536  0.86998557  0.87096475  0.86996496
  0.86688312  0.87201608  0.86995465  0.86795506  0.87098536  0.86691404
  0.86589363  0.86082251  0.86386312  0.8628221   0.86084313  0.8618223
  0.8598021   0.85675119]
```

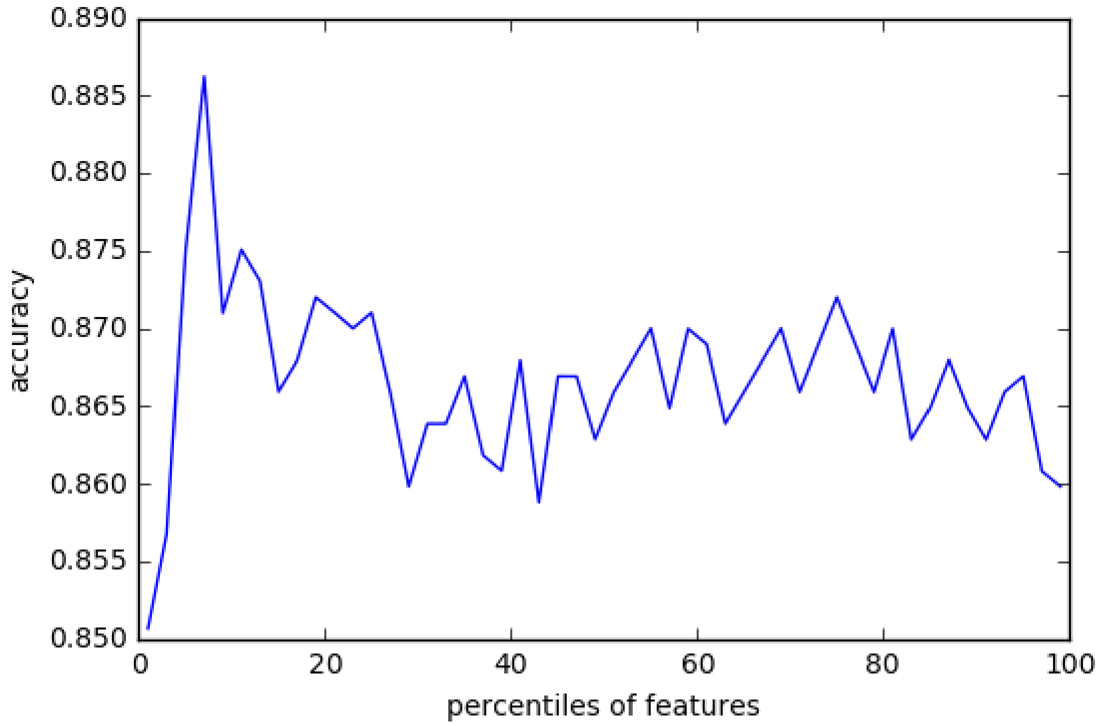
In [48]:

```
opt=np.where(results==results.max())[0]
print('Optimal number of features %d'%percentiles[opt])#I still couldn't find out the reason why it
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-48-3fa7ef434f1d> in <module>()
      1 opt=np.where(results==results.max())[0]
----> 2 print('Optimal number of features %d'%percentiles[opt])

TypeError: only integer scalar arrays can be converted to a scalar index
```

In [31]:

```
######################################
#>>> 6> Visualization

import pylab as pl
pl.plot(percentiles,results)
pl.xlabel('percentiles of features')
pl.ylabel('accuracy')
pl.show()
```



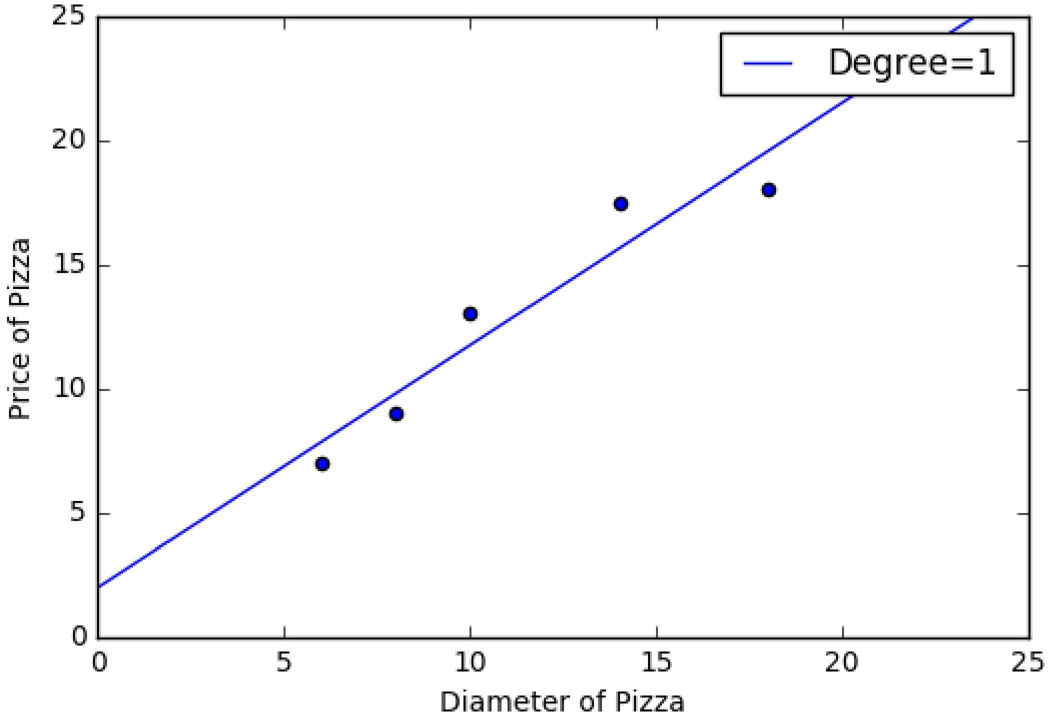
In [34]:

```
#Validation
from sklearn import feature_selection


fs=feature_selection.SelectPercentile(feature_selection.chi2,percentile=20)

X_train_fs=fs.fit_transform(X_train,y_train)
dt.fit(X_train_fs,y_train)

X_test_fs=fs.transform(X_test)
print(dt.score(X_test_fs,y_test))
```

0.823708206687

In [35]:

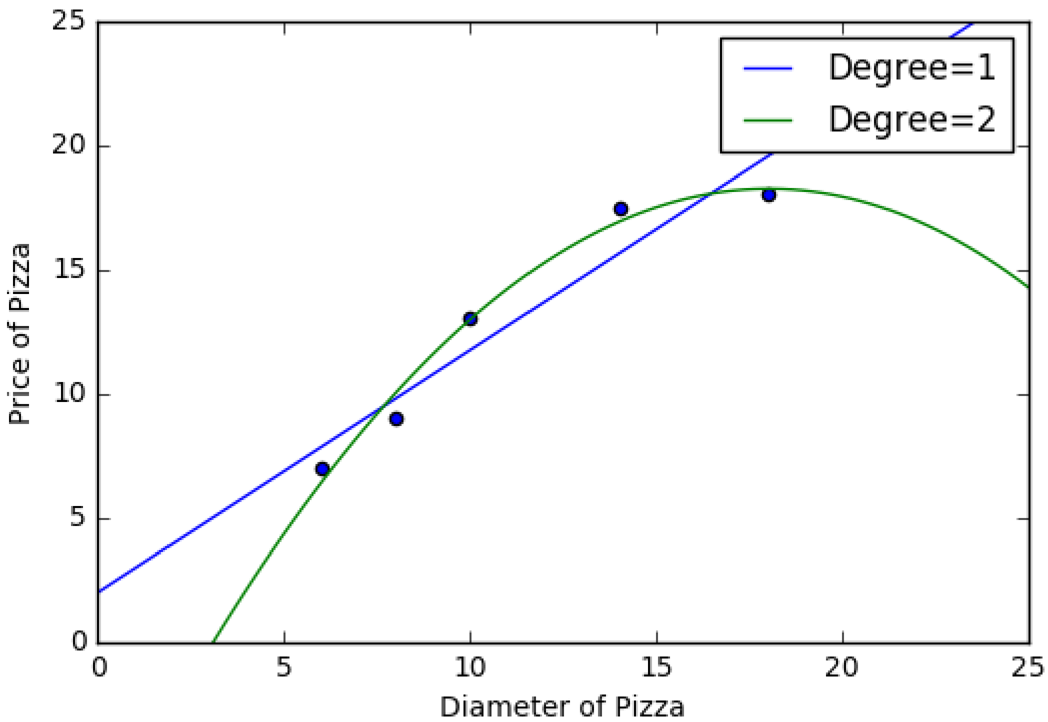
```
#*********************************************#
#
#          Regularization
#
#*********************************************#



#linearregression
#####################################
#>>> 1> Input dataset
X_train=[[6],[8],[10],[14],[18]]
y_train=[[7],[9],[13],[17.5],[18]]

#####################################
#>>> 2> Build linearregression model
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(X_train,y_train)

#####################################
#>>> 3> Predict
xx=np.linspace(0,26,100)
xx=xx.reshape(xx.shape[0],1)
yy=regressor.predict(xx)

#####################################
#>>> 4> Visualization and Output
import matplotlib.pyplot as plt

plt.scatter(X_train,y_train)
plt1,=plt.plot(xx,yy,label='Degree=1')
plt.axis([0,25,0,25])
plt.xlabel('Diameter of Pizza')
plt.ylabel('Price of Pizza')
plt.legend(handles=[plt1])
plt.show()

print('The R-squared value of Linear Regressor performing on the training data is',regressor.score(X
```



The R-squared value of Linear Regressor performing on the training data is 0.9100015
96424

In [36]:

```
#polynominal featrues in degree 2
########################################
#>>> 2&3>  Build model and Predict
from sklearn.preprocessing import PolynomialFeatures

poly2=PolynomialFeatures(degree=2)
X_train_poly2=poly2.fit_transform(X_train)

regressor_poly2=LinearRegression()
regressor_poly2.fit(X_train_poly2,y_train)

xx_poly2=poly2.transform(xx)
yy_poly2=regressor_poly2.predict(xx_poly2)

########################################
#>>> 4> Visualization and Output
plt.scatter(X_train,y_train)
plt1,=plt.plot(xx,yy,label='Degree=1')
plt2,=plt.plot(xx,yy_poly2,label='Degree=2')
plt.axis([0,25,0,25])
plt.xlabel('Diameter of Pizza')
plt.ylabel('Price of Pizza')
plt.legend(handles=[plt1,plt2])
plt.show()

print('The R-squared value of Polynominal Regressor(Degree=2) performing on the training data is',re
```

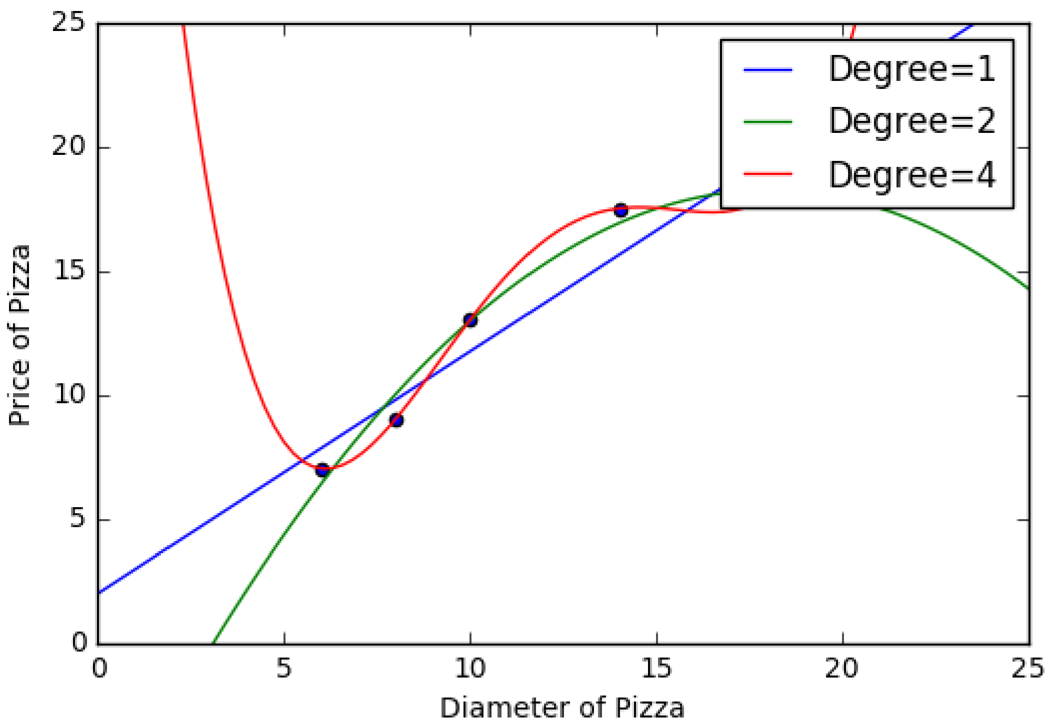



```
The R-squared value of Polynominal Regressor(Degree=2) performing on the training da
ta is 0.98164216396
```

In [39]:

```
#polynominal featrues in degree 4
########################################
#>>> 2&3> Build model and Predict
from sklearn.preprocessing import PolynomialFeatures

poly4=PolynomialFeatures(degree=4)
X_train_poly4=poly4.fit_transform(X_train)

regressor_poly4=LinearRegression()
regressor_poly4.fit(X_train_poly4,y_train)

xx_poly4=poly4.transform(xx)
yy_poly4=regressor_poly4.predict(xx_poly4)
print(regressor_poly4.coef_)
print(np.sum(regressor_poly4.coef_**2))

########################################
#>>> 4> Visualization and Output
plt.scatter(X_train,y_train)
plt1,=plt.plot(xx,yy,label='Degree=1')
plt2,=plt.plot(xx,yy_poly2,label='Degree=2')
plt4,=plt.plot(xx,yy_poly4,label='Degree=4')
plt.axis([0,25,0,25])
plt.xlabel('Diameter of Pizza')
plt.ylabel('Price of Pizza')
plt.legend(handles=[plt1,plt2,plt4])
plt.show()

print('The R-squared value of Polynominal Regressor(Degree=4) performing on the training data is',re
```

```
[[  0.00000000e+00  -2.51739583e+01   3.68906250e+00  -2.12760417e-01
    4.29687500e-03]]
647.382645737
```




```
The R-squared value of Polynominal Regressor(Degree=4) performing on the training da
ta is 1.0
```

In [40]:

```
######################################
#>>> 5> Validation

#Prepare test dataset
X_test=[[6],[8],[11],[16]]
y_test=[[8],[12],[15],[18]]

#linear regression
print('The R-squared value of Linear Regressor performing on the test data is',regressor.score(X_tes

#polynominal 2
X_test_poly2=poly2.fit_transform(X_test)
print('The R-squared value of Polynominal Regressor(Degree=2) performing on the test data is',regres

#polynominal 4
X_test_poly4=poly4.fit_transform(X_test)
print('The R-squared value of Polynominal Regressor(Degree=4) performing on the test data is',regres
```

```
The R-squared value of Linear Regressor performing on the test data is 0.80972683246
7
The R-squared value of Polynominal Regressor(Degree=2) performing on the test data i
s 0.867544365635
The R-squared value of Polynominal Regressor(Degree=4) performing on the test data i
s 0.809588079577
```

In [42]:

```
#lasso
######################################
#>>> 2&3>  Build model and Predict
from sklearn.linear_model import Lasso

lasso_poly4=Lasso()
lasso_poly4.fit(X_train_poly4,y_train)
print(lasso_poly4.score(X_test_poly4,y_test))
print(lasso_poly4.coef_)
```

```
0.83889268736
[  0.00000000e+00   0.00000000e+00   1.17900534e-01   5.42646770e-05
  -2.23027128e-04]
```

```
C:\Program Files\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_descen
t.py:466: ConvergenceWarning: Objective did not converge. You might want to increase
 the number of iterations
  ConvergenceWarning)
```