Machine Learning in Python

Supervised Learning: Explaining Titanic Hypothesis with Decision Trees

In [1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Preprocessing

First, we must load the dataset. We assume it is located in the data/titanic.csv file

In [16]:
```
import graphviz
##don't forget to install graphvis
from graphviz import Graph
import csv
import numpy as np
with open('data/titanic.csv', 'rb') as csvfile:
    titanic_reader = csv.reader(csvfile, delimiter=',', quotechar='"')

    # Header contains feature names
    row = titanic_reader.next()
    feature_names = np.array(row)

    # Load dataset, and target classes
    titanic_X, titanic_y = [], []
    for row in titanic_reader:
        titanic_X.append(row)
        titanic_y.append(row[2]) # The target value is "survived"

    titanic_X = np.array(titanic_X)
    titanic_y = np.array(titanic_y)
```

In [17]:
```
print titanic_X[:,4]
```

```
['29.0000' ' 2.0000' '30.0000' ..., 'NA' 'NA' 'NA']
```

In [18]:
```
print feature_names, titanic_X[0], titanic_y[0]
```

```
 'ticket' 'boat' 'sex'] ['1' '1st' '1' 'Allen, Miss Elisabeth Walton' '29.0000'
'Southampton'
 'St Louis, MO' 'B-5' '24160 L221' '2' 'female'] 1
```

Keep only class (1st,2nd,3rd), age (float), and sex (masc, fem)

In [19]:
```
# we keep the class, the age and the sex
titanic_X = titanic_X[:, [1, 4, 10]]
feature_names = feature_names[[1, 4, 10]]
```

In [20]:
```
print feature_names
print titanic_X[12], titanic_y[12]
```

```
['pclass' 'age' 'sex']
['1st' 'NA' 'female'] 1
```

Solve missing values ('NA') for the 'age' feature. Solution: use the mean value.

In [21]:
```
# We have missing values for age
# Assign the mean value
ages = titanic_X[:, 1]
mean_age = np.mean(titanic_X[ages != 'NA', 1].astype(np.float))
titanic_X[titanic_X[:, 1] == 'NA', 1] = mean_age
```

In [22]:
```
print feature_names
print titanic_X[12], titanic_y[12]
```

```
['pclass' 'age' 'sex']
['1st' '31.1941810427' 'female'] 1
```

Class and sex are categorical classes. Sex can be converted to a binary value (0=female,1=male):

In [23]:
```
# Encode sex
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
label_encoder = enc.fit(titanic_X[:, 2])
print "Categorical classes:", label_encoder.classes_
integer_classes = label_encoder.transform(label_encoder.classes_)
print "Integer classes:", integer_classes
t = label_encoder.transform(titanic_X[:, 2])
titanic_X[:, 2] = t
```

In [24]:
```
print feature_names
print titanic_X[12], titanic_y[12]
```

```
['pclass' 'age' 'sex']
['1st' '31.1941810427' '0'] 1
```

Now, we have to convert the class. Since we have three different classes, we cannot convert to binary values (and using 0/1/2 values would imply an order, something we do not want). We use OneHotEncoder to get three different attributes

In [25]:
```
from sklearn.preprocessing import OneHotEncoder

enc = LabelEncoder()
label_encoder = enc.fit(titanic_X[:, 0])
print "Categorical classes:", label_encoder.classes_
integer_classes = label_encoder.transform(label_encoder.classes_).reshape(3, 1)
##reshape into three different classes first,second and third
print "Integer classes:", integer_classes
enc = OneHotEncoder()
one_hot_encoder = enc.fit(integer_classes)
# First, convert clases to 0-(N-1) integers using label_encoder
num_of_rows = titanic_X.shape[0]
t = label_encoder.transform(titanic_X[:, 0]).reshape(num_of_rows, 1)
# Second, create a sparse matrix with three columns, each one indicating if the i
new_features = one_hot_encoder.transform(t)
# Add the new features to titanix_X
titanic_X = np.concatenate([titanic_X, new_features.toarray()], axis = 1)
#Eliminate converted columns
titanic_X = np.delete(titanic_X, [0], 1)
# Update feature names
feature_names = ['age', 'sex', 'first_class', 'second_class', 'third_class']
# Convert to numerical values
titanic_X = titanic_X.astype(float)
titanic_y = titanic_y.astype(float)
print titanic_X
```

```
Categorical classes: ['1st' '2nd' '3rd']
Integer classes: [[0]
 [1]
 [2]]
[[ 29.          0.          1.          0.          0.        ]
 [  2.          0.          1.          0.          0.        ]
 [ 30.          1.          1.          0.          0.        ]
 ....
```

```
[ 31.19418104   1.            0.            0.            1.        ]]
```

In [26]:
```
print titanic_X.shape
```

```
(1313, 5)
```

In [27]:
```
##class sklearn.preprocessing.StandardScaler(copy=True, with_mean=True, with_std=
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
titanic_X_ScaledAge=titanic_X

numpy.copyto(titanic_X_ScaledAge, titanic_X)
titanic_X_MaxMinAge=titanic_X
numpy.copyto(titanic_X_MaxMinAge, titanic_X)
```

In [27]:

In [28]:
```
scaler = StandardScaler()
ages = titanic_X[:, 0]

scaler=scaler.fit(titanic_X_ScaledAge[:,0].astype(float))

scaled_values=scaler.transform(titanic_X_ScaledAge[:,0])

titanic_X_ScaledAge[:,0]=scaled_values

print titanic_X_ScaledAge
```

```
[[ -2.14450535e-01   0.00000000e+00   1.00000000e+00   0.00000000e+00
    0.00000000e+00]
 [ -2.85332323e+00   0.00000000e+00   1.00000000e+00   0.00000000e+00
    0.00000000e+00]
 [ -1.16714509e-01   1.00000000e+00   1.00000000e+00   0.00000000e+00
    0.00000000e+00]
 ...,
 [  2.10559129e-12   1.00000000e+00   0.00000000e+00   0.00000000e+00
    1.00000000e+00]
 [  2.10559129e-12   0.00000000e+00   0.00000000e+00   0.00000000e+00
    1.00000000e+00]
 [  2.10559129e-12   1.00000000e+00   0.00000000e+00   0.00000000e+00
    1.00000000e+00]]
```

In [29]:
```
mscaler = MinMaxScaler()
```

```
maxminscaler=mscaler.fit(titanic_X_MaxMinAge[:,0].astype(float))

maxminscaled_values=maxminscaler.transform(titanic_X_MaxMinAge[:,0])

titanic_X_MaxMinAge[:,0]=maxminscaled_values

print titanic_X_MaxMinAge
```

```
[[ 0.40705854  0.          1.          0.          0.        ]
 [ 0.02588189  0.          1.          0.          0.        ]
 [ 0.4211762   1.          1.          0.          0.        ]
 ...,
 [ 0.43803523  1.          0.          0.          1.        ]
 [ 0.43803523  0.          0.          0.          1.        ]
 [ 0.43803523  1.          0.          0.          1.        ]]
```

In [30]:
```
print titanic_X
```

```
[[ 0.40705854  0.          1.          0.          0.        ]
 [ 0.02588189  0.          1.          0.          0.        ]
 [ 0.4211762   1.          1.          0.          0.        ]
 ...,
 [ 0.43803523  1.          0.          0.          1.        ]
 [ 0.43803523  0.          0.          0.          1.        ]
 [ 0.43803523  1.          0.          0.          1.        ]]
```

In [31]:
```
print titanic_X_ScaledAge
```

```
[[ 0.40705854  0.          1.          0.          0.        ]
 [ 0.02588189  0.          1.          0.          0.        ]
 [ 0.4211762   1.          1.          0.          0.        ]
 ...,
 [ 0.43803523  1.          0.          0.          1.        ]
 [ 0.43803523  0.          0.          0.          1.        ]
 [ 0.43803523  1.          0.          0.          1.        ]]
```

Separate training and test sets

In [32]:
```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(titanic_X, titanic_y, test_si
```

```
[[ 0.43803523  1.          0.          0.          1.        ]
 [ 0.43803523  0.          1.          0.          0.        ]
 [ 0.43803523  1.          0.          0.          1.        ]
 ...,
 [ 0.16705843  0.          0.          1.          0.        ]
 [ 0.25176435  1.          0.          1.          0.        ]
 [ 0.43803523  0.          0.          0.          1.        ]]
[[ 0.25176435  1.          0.          1.          0.        ]
 [ 0.67529396  0.          1.          0.          0.        ]
 [ 0.43803523  1.          1.          0.          0.        ]
 ...,
 [ 0.27999966  1.          0.          0.          1.        ]
 [ 0.75999989  1.          1.          0.          0.        ]
 [ 0.43803523  0.          0.          0.          1.        ]]
```

Seperate trainig and test sets for scales ages

In [33]:

In [34]:

In [35]:

```
print X_train_scaled
print X_test_scaled
```

```
[[ 0.43803523  1.          0.          0.          1.        ]
 [ 0.43803523  0.          1.          0.          0.        ]
 [ 0.43803523  1.          0.          0.          1.        ]
 ...,
 [ 0.16705843  0.          0.          1.          0.        ]
 [ 0.25176435  1.          0.          1.          0.        ]
 [ 0.43803523  0.          0.          0.          1.        ]]
[[ 0.25176435  1.          0.          1.          0.        ]
 [ 0.67529396  0.          1.          0.          0.        ]
 [ 0.43803523  1.          1.          0.          0.        ]
 ...,
 [ 0.27999966  1.          0.          0.          1.        ]
 [ 0.75999989  1.          1.          0.          0.        ]
 [ 0.43803523  0.          0.          0.          1.        ]]
```

Create a training and test set for maxminscaled variables

In [36]:

```
[[ 0.43803523  1.          0.          0.          1.        ]
 [ 0.43803523  0.          1.          0.          0.        ]
 [ 0.43803523  1.          0.          0.          1.        ]
 ...,
 [ 0.16705843  0.          0.          1.          0.        ]
 [ 0.25176435  1.          0.          1.          0.        ]
 [ 0.43803523  0.          0.          0.          1.        ]]
```

In [38]:
```
print X_test_maxmin
```

```
[[ 0.25176435  1.          0.          1.          0.        ]
 [ 0.67529396  0.          1.          0.          0.        ]
 [ 0.43803523  1.          1.          0.          0.        ]
 ...,
 [ 0.27999966  1.          0.          0.          1.        ]
 [ 0.75999989  1.          1.          0.          0.        ]
 [ 0.43803523  0.          0.          0.          1.        ]]
```

Data exploration

In [39]:
```
#going to create a dummy nparray just for visualization
titanic_y_shp=titanic_y.reshape(1313,1)
print titanic_y_shp
print titanic_y_shp.shape
titanic_explore = np.append(titanic_X, titanic_y_shp, 1)
```

```
[[ 1.]
 [ 0.]
 [ 0.]
 ...,
 [ 0.]
 [ 0.]
 [ 0.]]
(1313, 1)
```

```
import pandas and convert to dataframe
```

this needs to be tidied up

In [40]:
```
import pandas as pd

df1=pd.DataFrame({feature_names[0]:titanic_explore[:,0],feature_names[1]:titanic_
                 ,feature_names[3]:titanic_explore[:,3],feature_names[4]:titanic_
```

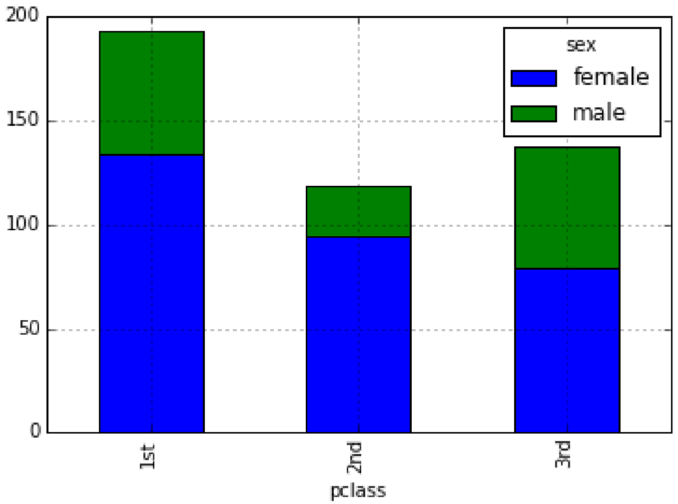
```
C:\Python27\lib\site-packages\pytz\__init__.py:35: UserWarning: Module argparse
was already imported from C:\Python27\lib\argparse.pyc, but c:\python27\lib\site
-packages is being added to sys.path
  from pkg_resources import resource_stream
```

Explore if there is a gender balance between the number of survivors

In [41]:
```
import matplotlib.pyplot as plt
#print summed_first_class.iat[0,0]

titanic_invest = pd.read_csv('data/titanic.csv', sep=',')

print titanic_invest.columns
##transform the data reshape its using pandas
grouped_by_t=(titanic_invest['survived']).groupby([titanic_invest['pclass'],titan
summed_by_class_and_gender_t=grouped_by_t.sum()
summed_first_class_t=pd.DataFrame(summed_by_class_and_gender_t)
print summed_first_class_t


test5 = titanic_invest.groupby(['pclass', 'sex'])['survived'].sum().unstack('sex'
print test5
%pylab inline
test5.plot(kind='bar', stacked=True)
```

```
Index([u'row.names', u'pclass', u'survived', u'name', u'age', u'embarked', u'hom
e.dest', u'room', u'ticket', u'boat', u'sex'], dtype='object')
               survived
pclass sex
1st    female       134
       male          59
2nd    female        94
       male          25
3rd    female        79
       male          58
sex     female  male
pclass
1st        134    59
2nd         94    25
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0xd1f8d50>



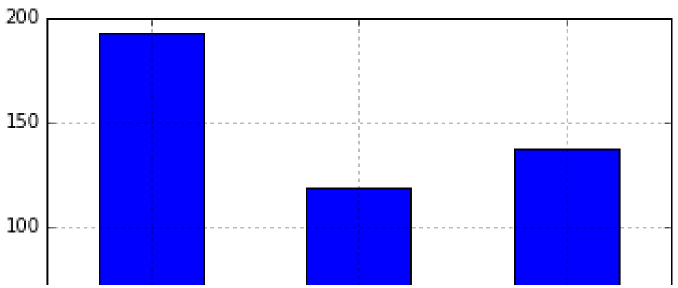
There appears to be a Gender Imbalance, more of the survivors appear to be females in all berths

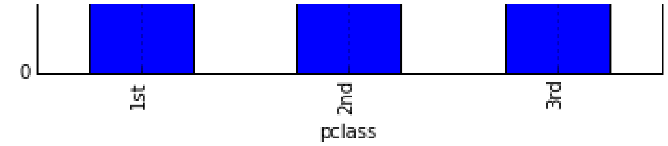
Explore if there is a (socio-economic) Class imbalance

In [42]:
```
test6 = titanic_invest.groupby(['pclass'])['survived'].sum().fillna(0)
print test6
%pylab inline
test6.plot(kind='bar', stacked=True)
```

```
pclass
1st       193
2nd       119
3rd       137
Name: survived, dtype: int64
Populating the interactive namespace from numpy and matplotlib
```

Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0xd44fb70>





again there appears to be a class (socio economic) imablance in the survival rates

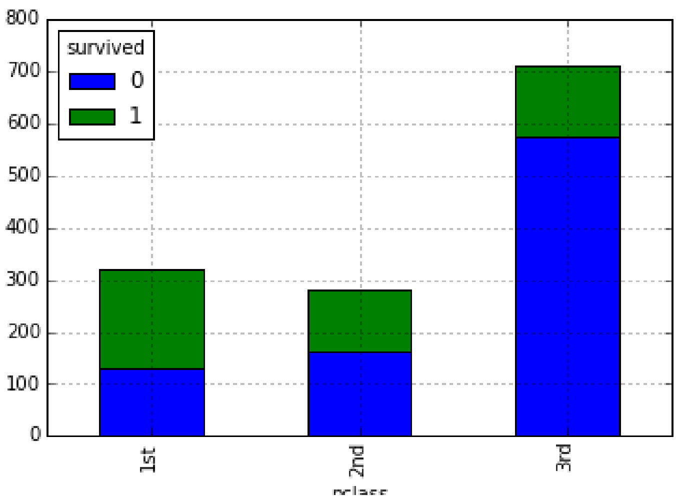
Did (socio economic) Class play a role in peoples survival was there a difference?

was there a difference in the number of survivors to those who perished by socio economic class?

In [43]:
```
print titanic_invest.columns
test7 = titanic_invest.groupby(['pclass','survived'])['name'].count().unstack('su
print test7
%pylab inline
test7.plot(kind='bar', stacked=True)
```

```
Index([u'row.names', u'pclass', u'survived', u'name', u'age', u'embarked', u'hom
e.dest', u'room', u'ticket', u'boat', u'sex'], dtype='object')
survived    0    1
pclass
1st       129  193
2nd       161  119
3rd       574  137
Populating the interactive namespace from numpy and matplotlib
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0xd563ab0>

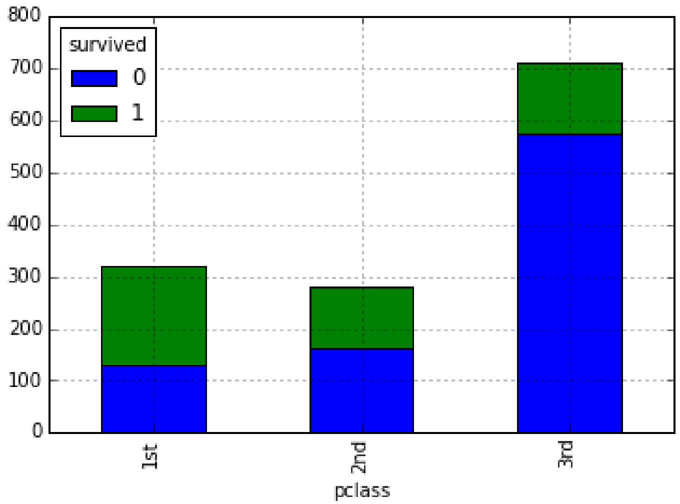


What effect does (socio economic) class and gender have on survival

In [44]:
```
test7 = titanic_invest.groupby(['pclass','survived'])['name'].count().unstack('su
print test7
%pylab inline
test7.plot(kind='bar', stacked=True)
```

```
survived    0    1
pclass
1st       129  193
2nd       161  119
3rd       574  137
Populating the interactive namespace from numpy and matplotlib
```

Out[44]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xd69e170>
```

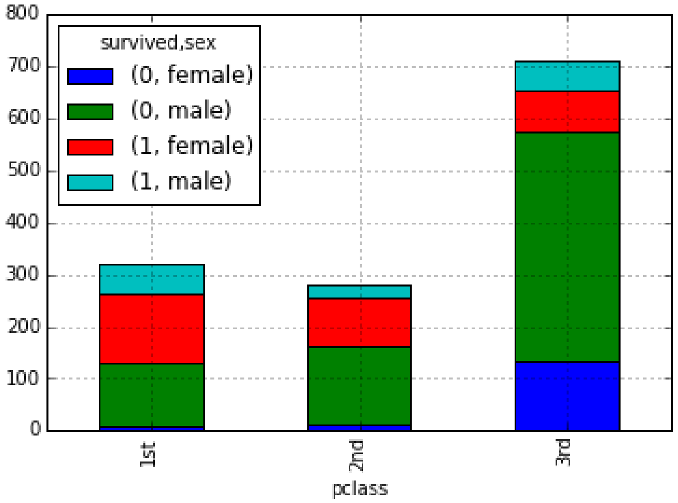


In [45]:
```
test8 = titanic_invest.groupby(['pclass','survived','sex'])['name'].count().unsta
print test8
print test8.columns
%pylab inline
test8.plot(kind='bar', stacked=True)
```

```
survived       0           1
sex       female  male  female  male
pclass
1st            9   120     134    59
```

```
survived  sex
0         female
          male
1         female
          male
Populating the interactive namespace from numpy and matplotlib
```

Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0xd878790>

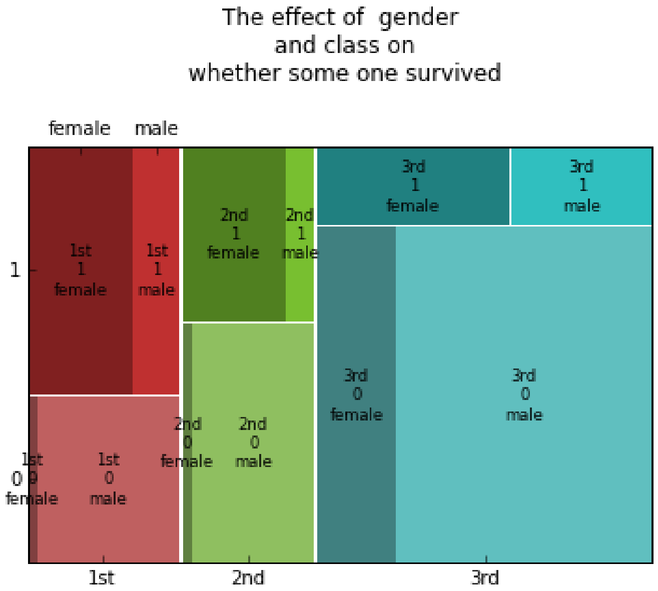


In [46]:
```
test9 = titanic_invest.groupby(['pclass','survived','sex'])['name'].count().filln
print test9
print 'test 9 class',type(test9)
%pylab inline
import statsmodels
from statsmodels.graphics.mosaicplot import mosaic
mosaic(test9, title='The effect of  gender\n and class on\n whether some one surv
```

```
pclass  survived  sex
1st     0         female      9
                  male      120
        1         female    134
                  male       59
2nd     0         female     13
                  male      148
        1         female     94
                  male       25
3rd     0         female    134
                  male      440
        1         female     79
                  male       58
```

```
Populating the interactive namespace from numpy and matplotlib
```

Out[46]:
```
(<matplotlib.figure.Figure at 0x103c8710>,
 OrderedDict([(('1st', '0', 'female'), (0.0, 0.0, 0.016902795722373597, 0.399290
15084294583)), (('1st', '0', 'male'), (0.017441181067604755, 0.0, 0.225370609631
64796, 0.39929015084294583)), (('1st', '1', 'female'), (0.0, 0.4026124099791585,
 0.16821055086755904, 0.5973875900208414)), (('1st', '1', 'male'), (0.1687489362
1279019, 0.4026124099791585, 0.074062854486462548, 0.5973875900208414)), (('2nd'
, '0', 'female'), (0.24776228574875767, 0.0, 0.017010825112623713, 0.57308970099
66777)), (('2nd', '0', 'male'), (0.26524127203114761, 0.0, 0.19366170128217766,
0.5730897009966777)), (('2nd', '1', 'female'), (0.24776228574875767, 0.576411960
1328902, 0.16641359227824645, 0.42358803986710963)), (('2nd', '1', 'male'), (0.4
1464403919677034, 0.5764119601328902, 0.044258934116554918, 0.42358803986710963)
), (('3rd', '0', 'female'), (0.4638534683628302, 0.0, 0.12488560403022837, 0.804
6315376312432)), (('3rd', '0', 'male'), (0.58992786736342917, 0.0, 0.41007213263
657083, 0.8046315376312432)), (('3rd', '1', 'female'), (0.4638534683628302, 0.80
79537967674557, 0.30847927880786224, 0.19204620323254412)), (('3rd', '1', 'male'
), (0.77352154214106317, 0.8079537967674557, 0.22647845785893689, 0.192046203232
54412))]))
```



Again the stacked bar chart and the mosaic plots tell a very sad story, that although gender did play a part on whether someone survived, (socio - economic) class would appear to have played a bigger part.

In [47]:
```
import pygal
pivot_gender=titanic_invest.pivot_table('age',index='sex',cols=['pclass','survive
```

```
pclass          1st                   2nd                   3rd
survived          0          1          0          1          0          1
sex
female    35.200000  37.906250  31.400000  26.853333  23.379310  21.720239
male      44.841463  34.253877  31.698113  14.841267  26.219444  19.379628
<class 'pandas.core.frame.DataFrame'>
```

```
C:\Python27\lib\site-packages\pandas\util\decorators.py:53: FutureWarning: cols
is deprecated, use columns instead
  warnings.warn(msg, FutureWarning)
```

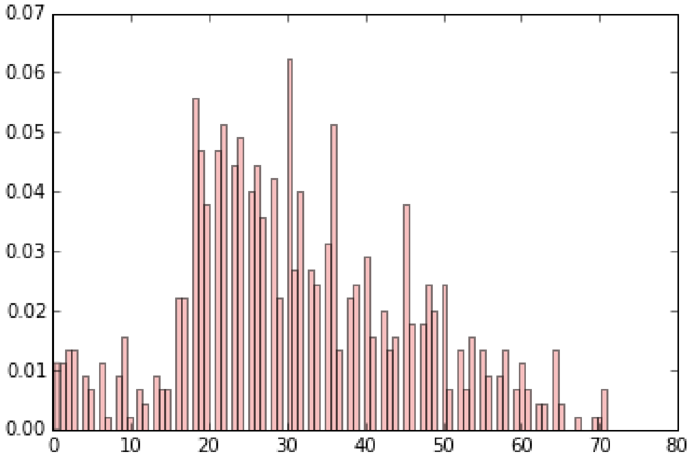
using the pivot table above it can be seen that the average age of the male survivors compared to males who persihedd was considerably less. Also the average age of both male an female survivors decreases

Again this looks to show that age, (socio economic) class and gender do pay a significant role in someones survival.

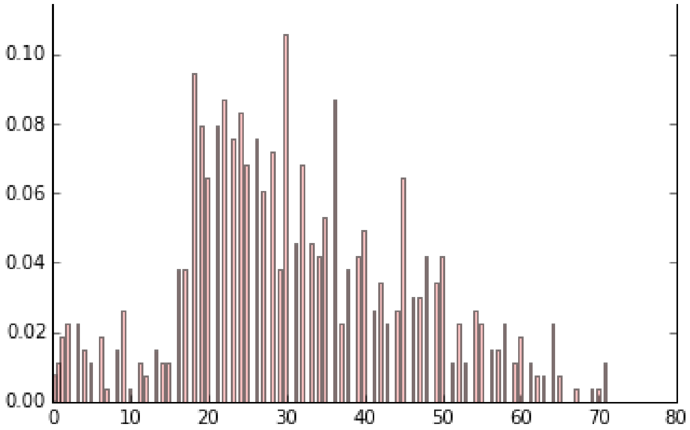
Lets take a closer look at the distribution of ages

lets look at the distributions of age

In [48]:
```
age=titanic_invest['age'][(titanic_invest['age'] >= 0)].values
bins=100
plt.hist(age, bins, normed=True, color="#F08080", alpha=.5);
```



In [49]:
```
max_data = np.r_[age].max()
bins = np.linspace(0, max_data, max_data + 100)
```



From the above age dsitribution, lets perform a series of tests

Look at the age distribution, by gender

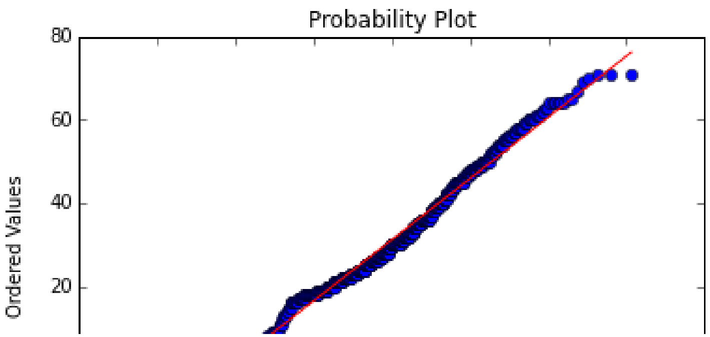
In [50]:
```
from  scipy.stats import shapiro
shapiro(age)
##This means that if your p-value <= 0.05, then you would reject the NULL hypothe
```

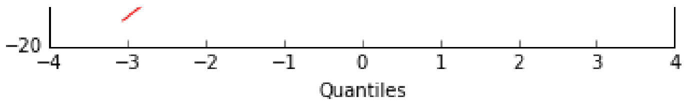
Out[50]:
```
(0.9826399087905884, 7.770732963763294e-07)
```

P value is lees than 0.05 and we can reject the null hypothesis, lets look at the q-q plot

In [51]:
```
from scipy.stats import probplot
from scipy.stats import norm

probplot(age, dist=norm,plot=plt)
plt.show
```

Out[51]:
```
<function matplotlib.pyplot.show>
```





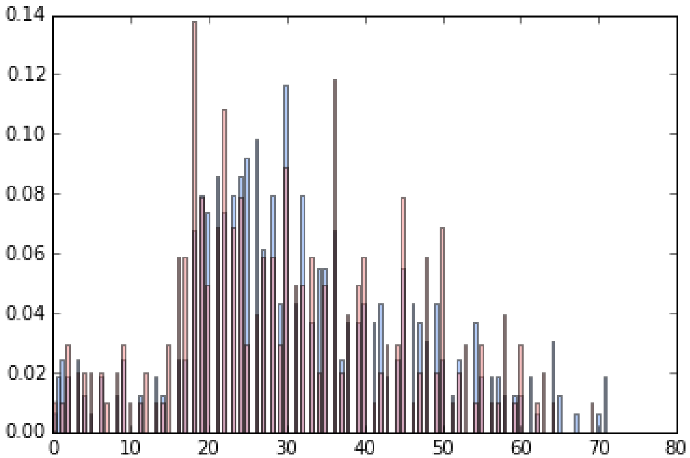
Deviates from normal

it backs up the histogram above. Skew test, tests the null hypothesis that the skewness of the population that the sample was drawn from is the same as that of a corresponding normal distribution. So far males at the 0.05 level it can be rejected, but for females it cannot, but at the 0.05 level it can. Lets test for normality.(3.0750140208828016, 0.0021049265805137183) In [ ]:

The distribution of ages by gender

In [52]:

```
#tips[['total_bill', 'tip', 'smoker', 'time']].head(5)
female_age=titanic_invest['age'][(titanic_invest['sex']=='female') & (titanic_inv
##ignore nulls maybe you should replace with avg's
male_age=titanic_invest['age'][(titanic_invest['sex']=='male') & (titanic_invest[


max_data = np.r_[female_age, male_age].max()

bins = np.linspace(0, max_data, max_data + 100)
plt.hist(male_age, bins, normed=True, color="#6495ED", alpha=.5)
plt.hist(female_age, bins, normed=True, color="#F08080", alpha=.5);
```



So in the overall population there is a skew in age bygender , there is a less positive skew for females when compared to the skew for males. Best to calculate some measure

```
from scipy.stats import skewtest
from scipy.stats import skew

print kurtosis(age)
print skew(age)
print skewtest(age)

#A flatter distribution (when compared to a gaussian distribution) has a negative
##If the skewness is greater than 1.0 (or less than -1.0), the skewness is substa
```

```
-0.229521383691
0.302265425212
(3.0750140208828016, 0.0021049265805137183)
```

lets put some measures on the distribution

use scipy stats to calcualte some measures

lets take a look at waht this is telling us , it backs up the histogram above. Skew test, tests the null hypothesis that the skewness of the population that the sample was drawn from is the same as that of a corresponding normal distribution. So far males at the 0.05 level it can be rejected, but for females it cannot, but at the 0.10 it can. Lets test for normality.

In [54]:
```
from scipy.stats import kurtosis
from scipy.stats import skewtest
from scipy.stats import skew
kurtskewmale=titanic_invest['age'][(titanic_invest['sex']=='male') & (titanic_inv

print 'males kutosis,skew,skewtest'
print kurtosis(kurtskewmale)
print skew(kurtskewmale)
print skewtest(kurtskewmale)
#A flatter distribution (when compared to a gaussian distribution) has a negative
##If the skewness is greater than 1.0 (or less than -1.0), the skewness is substa

print 'males kutosis,skew,skewtest'
kurtskewfemale=titanic_invest['age'][(titanic_invest['sex']=='female') & (titanic

print 'females kutosis,skew,skewtest'
print kurtosis(kurtskewfemale)
print skew(kurtskewfemale)
print skewtest(kurtskewfemale)

##use reshape
```

```
males kutosis,skew,skewtest
-0.0353121047863
0.330741998691
(2.6498981644934689, 0.0080516035916345305)
males kutosis,skew,skewtest
females kutosis,skew,skewtest
-0.538240159127
0.262501158122
(1.6926953783388221, 0.090513465441597482)
```

In [55]:

```
import pygal
```

In [56]:

```

radar_chart = pygal.Radar()
radar_chart.title = 'Difference in av.ages between males and Females'
radar_chart.x_labels=['1stclass_notsurv','1stclass_surv','2ndclass_notsurv','2ndc
female_age=pivot_gender.loc['female'].values
female_age=female_age.tolist()
male_age=pivot_gender.loc['male'].values ##write out the values and convert to ar
male_age=male_age.tolist()

radar_chart.add('female_average_age',female_age)
radar_chart.add('male_average_age',male_age)
pivot_gender_std=titanic_invest.pivot_table('age',index='sex',cols=['pclass','sur
female_age_std=pivot_gender_std.loc['female'].values.tolist()
male_age_std=pivot_gender_std.loc['male'].values.tolist()
##radar_chart.add('female_std_age',female_age_std)
#radar_chart.add('male_std_age',male_age_std)
pivot_gender_median=titanic_invest.pivot_table('age',index='sex',cols=['pclass','
female_age_median=pivot_gender_std.loc['female'].values.tolist()
male_age_median=pivot_gender_std.loc['male'].values.tolist()
radar_chart.add('female_median_age',female_age_median)
radar_chart.add('male_median_age',male_age_median)
radar_chart.render_to_file('radar_chart.svg')
from IPython.display import SVG
SVG(filename='radar_chart.svg')
```

Out[56]: <IPython.core.display.SVG at 0xd870f50>

In [60]:

```
import pygal

#tips[['total_bill', 'tip', 'smoker', 'time']].head(5)
#female_age=titanic_invest['age'][(titanic_invest['sex']=='female') & (titanic_in
```

```
box_plot = pygal.Box()
box_plot.title = 'female age and Male age boxplot'
box_plot.add('Female Age', female_age)
box_plot.add('Male Age', male_age)

box_plot.render_to_file('box_plot.svg')
SVG(filename='box_plot.svg')
```

Out[60]: <IPython.core.display.SVG at 0x10fedbf0>

What about the boxplot for male and female survivors and non survivors.

In [68]:

```
import pygal

#tips[['total_bill', 'tip', 'smoker', 'time']].head(5)
female_surv_age=titanic_invest['age'][(titanic_invest['sex']=='female') & (titani
female_notsurv_age=titanic_invest['age'][(titanic_invest['sex']=='female') & (tit

male_surv_age=titanic_invest['age'][(titanic_invest['sex']=='male') & (titanic_in
male_notsurv_age=titanic_invest['age'][(titanic_invest['sex']=='male') & (titanic

##ignore nulls maybe you should replace with avg's
#male_age=titanic_invest['age'][(titanic_invest['sex']=='male') & (titanic_invest

box_plot = pygal.Box()
box_plot.title = 'female age and Male survived and non survived age boxplot'
box_plot.add('Female Age Survivors', female_surv_age)
box_plot.add('Female Age Non Survivors', female_notsurv_age)
box_plot.add('Male Age Survivors', male_surv_age)
box_plot.add('Male Age Non Survivors', female_notsurv_age)
box_plot.render_to_file('box_plot_gender_age_survivors.svg')
```

Out[68]: <IPython.core.display.SVG at 0x10f48390>

In [71]:

```
import pygal

#tips[['total_bill', 'tip', 'smoker', 'time']].head(5)
female_surv_age_1st=titanic_invest['age'][(titanic_invest['sex']=='female') & (ti
female_notsurv_age_1st=titanic_invest['age'][(titanic_invest['sex']=='female') &
male_surv_age_1st=titanic_invest['age'][(titanic_invest['sex']=='male') & (titani
male_notsurv_age_1st=titanic_invest['age'][(titanic_invest['sex']=='male') & (tit
#
```

```
male_surv_age_2nd=titanic_invest['age'][(titanic_invest['sex']=='male') & (titani
male_notsurv_age_2nd=titanic_invest['age'][(titanic_invest['sex']=='male') & (tit
#
female_surv_age_3rd=titanic_invest['age'][(titanic_invest['sex']=='female') & (ti
female_notsurv_age_3rd=titanic_invest['age'][(titanic_invest['sex']=='female') &
male_surv_age_3rd=titanic_invest['age'][(titanic_invest['sex']=='male') & (titani
male_notsurv_age_3rd=titanic_invest['age'][(titanic_invest['sex']=='male') & (tit


female_surv_age_1st
female_notsurv_age_1st
male_surv_age_1st
male_notsurv_age_1st
#
female_surv_age_2nd
female_notsurv_age_2nd
male_surv_age_2nd
male_notsurv_age_2nd
#
female_surv_age_3rd
female_notsurv_age_3rd
male_surv_age_3rd
male_notsurv_age_3rd

##ignore nulls maybe you should replace with avg's
#male_age=titanic_invest['age'][(titanic_invest['sex']=='male') & (titanic_invest

box_plot = pygal.Box()
box_plot.title = 'female age and Male survived and non survived age boxplot'
box_plot.add('Female Age 1st Class Survivors', female_surv_age_1st)
box_plot.add('Female Age 1st Class Non Survivors', female_notsurv_age_1st)
box_plot.add('Male Age Survivors 1st Class', male_surv_age_1st)
box_plot.add('Male Age Non Survivors 1st Class', male_notsurv_age_1st)

box_plot.add('Female Age 2nd Class Survivors', female_surv_age_2nd)
box_plot.add('Female Age 2nd Class Non Survivors', female_notsurv_age_2nd)
box_plot.add('Male Age  2nd Class Survivors', male_surv_age_2nd)
box_plot.add('Male Age  2nd Class Non Survivors', male_notsurv_age_2nd)

box_plot.add('Female Age 3rd Class Survivors', female_surv_age_3rd)
box_plot.add('Female Age 3rd Class Non Survivors', female_notsurv_age_3rd)
box_plot.add('Male 3rd Class Age Survivors', male_surv_age_3rd)
box_plot.add('Male 3rd Class Age Non Survivors', male_notsurv_age_3rd)
```

Out[71]: <IPython.core.display.SVG at 0x10f72ef0>

In [72]:
```
from scipy.stats import mode
pivot_gender_2=titanic_invest.pivot_table('age',index='sex',cols=['pclass','survi
print pivot_gender_2
```

```
        pclass  survived  sex
mean    1st     0         female              35.2
                          male            44.84146
                1         female          37.90625
                          male            34.25388
        2nd     0         female              31.4
                          male            31.69811
                1         female          26.85333
                          male            14.84127
        3rd     0         female          23.37931
                          male            26.21944
                1         female          21.72024
                          male            19.37963
mode    1st     0         female      ([2.0], [1.0])
                          male       ([46.0], [6.0])
                1         female     ([19.0], [4.0])
                          male       ([36.0], [4.0])
        2nd     0         female     ([22.0], [2.0])
                          male      ([30.0], [10.0])
                1         female     ([36.0], [5.0])
                          male        ([2.0], [3.0])
        3rd     0         female      ([9.0], [3.0])
                          male      ([26.0], [10.0])
                1         female     ([18.0], [4.0])
                          male        ([3.0], [2.0])
std     1st     0         female          23.44568
                          male            13.50123
                1         female          14.65396
                          male            14.58941
        2nd     0         female          11.86217
                          male            11.37514
                1         female          12.60061
                          male            13.81641
        3rd     0         female          12.62202
                          male             10.5125
                1         female          11.24558
                          male            12.83478
median  1st     0         female                36
```

```
                     male                   36
        2nd     0    female               28.5
                     male                 29.5
                1    female                 28
                     male                    8
        3rd     0    female                 21
                     male                   25
                1    female               18.5
                     male                 20.5
dtype: object
```

Build a simple logistic regression model

In [73]:
```
from sklearn import linear_model
clf_lm = linear_model.LogisticRegression()
clf_lm=clf_lm.fit(X_train,y_train)
print feature_names
print clf_lm.coef_
print clf_lm.intercept_
```

```
['age', 'sex', 'first_class', 'second_class', 'third_class']
[[-1.6535738  -2.45390771  1.47383216  0.51898878 -0.72940027]]
[ 1.26342066]
```

The coefficients back up what we saw earlier gender does have the most significant role in survival, folowed by class (first and third class has a highly significant role to play).

In [74]:
```
print clf_lm.score(X_test,y_test) #Returns the coefficient of determination R^2 o

y_pred=clf_lm.predict(X_test)
```

```
0.790273556231
```

In [75]:
```
clf_lm.coef_
```

Out[75]:
```
array([[-1.6535738 , -2.45390771,  1.47383216,  0.51898878, -0.72940027]])
```

regression model gives pretty good accuarcy try on the scaled age model

In [79]:
```
clf_lm = linear_model.LogisticRegression()

clflmScaledage = clf_lm.fit(X_train_scaled,y_train_scaled)
print clflmScaledage.score(X_test,y_test)
```

```
y_pred=clflmScaledage.predict(X_test)
confusion_matrix(y_test,y_pred)
# test on data that was not used for fitting
```

```
0.790273556231
[[-1.6535738  -2.45390771  1.47383216  0.51898878 -0.72940027]]
[ 1.26342066]
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-79-32db27e6474f> in <module>()
      6 print clflmScaledage.intercept_
      7 y_pred=clflmScaledage.predict(X_test)
----> 8 confusion_matrix(y_test,y_pred)
      9 # test on data that was not used for fitting

NameError: name 'confusion_matrix' is not defined
```

very poor try maxmin transform

In [86]:

```
clf_lm = linear_model.LogisticRegression()

clflmMaxMinage = clf_lm.fit(X_train_maxmin,y_train_maxmin)
print feature_names
print clflmMaxMinage.coef_
from sklearn import metrics
def measure_performance(X,y,clf, show_accuracy=True, show_classification_report=T
    y_pred=clf.predict(X)
    if show_accuracy:
        print "Accuracy:{0:.3f}".format(metrics.accuracy_score(y,y_pred)),"\n"

    if show_classification_report:
        print "Classification report"
        print metrics.classification_report(y,y_pred),"\n"

    if show_confusion_matrix:
        print "Confusion matrix"
        print metrics.confusion_matrix(y,y_pred),"\n"

measure_performance(X_test_maxmin,y_test_maxmin,clflmMaxMinage, show_classificati


# test on data that was not used for fitting
```

```
Accuracy:0.790

Classification report
             precision    recall  f1-score   support

        0.0       0.77      0.93      0.84       202
        1.0       0.84      0.57      0.68       127

avg / total       0.80      0.79      0.78       329


Confusion matrix
[[188  14]
 [ 55  72]]
```

For every one unit change in age, the log odds of survival (versus non-survival) decreases by 1.65. For males compared to females, the log odds of survival (versus non-survival) decreases by 2.45. For First class passengers, the log odds of survival (versus non-survival) increases by 1.47. For Second class passengers, the log odds of survival (versus non-survival) increases by 0.518. For third class passengers, the log odds of survival (versus non-survival) decreases by 0.518.

Decision Trees -explain the effeects of class, gender and age with a decision tree

Let build a decision tree and examine the output of it to explain the Titanic hypothesis

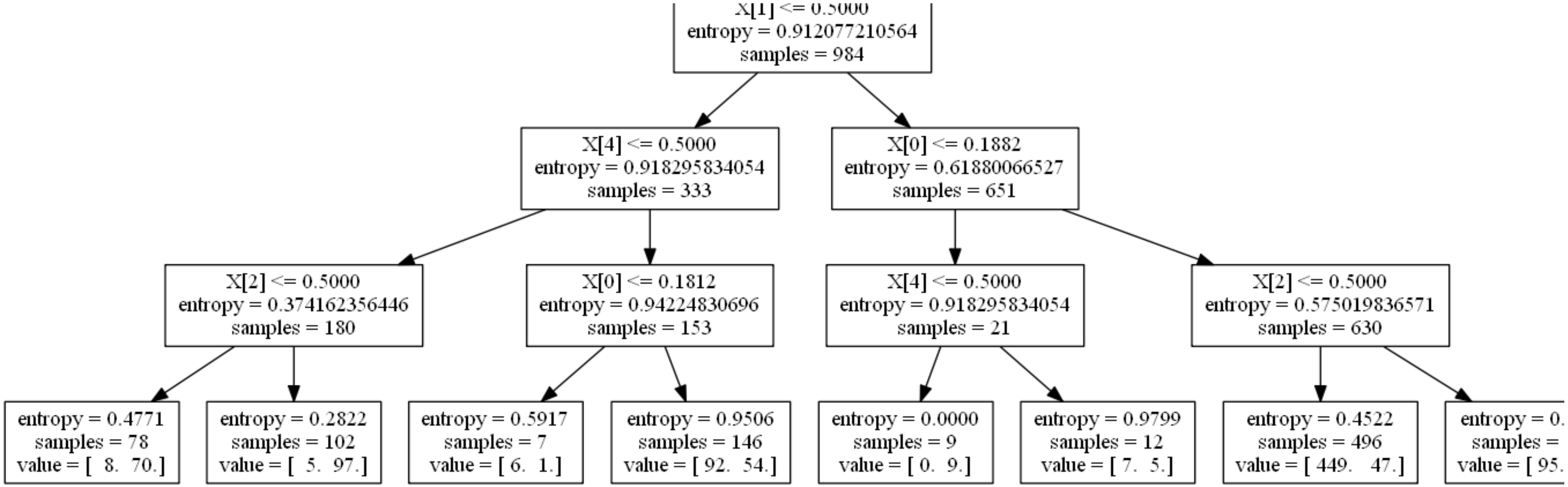
Fit a decision tree with the data.

In [81]:
```
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3,min_samples_le
clf = clf.fit(X_train,y_train)
```

Show the built tree, using pydot

In [82]:
```
import pydot,StringIO
dot_data = StringIO.StringIO()
tree.export_graphviz(clf, out_file=dot_data)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
graph.write_png('titanic.png')
from IPython.core.display import Image
```

Out[82]:



Measure Accuracy, precision, recall, f1 in the training set

In [83]:
```
from sklearn import metrics
def measure_performance(X,y,clf, show_accuracy=True, show_classification_report=T
    y_pred=clf.predict(X)
    if show_accuracy:
        print "Accuracy:{0:.3f}".format(metrics.accuracy_score(y,y_pred)),"\n"

    if show_classification_report:
        print "Classification report"
        print metrics.classification_report(y,y_pred),"\n"

    if show_confusion_matrix:
        print "Confusion matrix"
        print metrics.confusion_matrix(y,y_pred),"\n"

measure_performance(X_test,y_test,clf, show_classification_report=False, show_con
print 'foo',X_test
```

```
Accuracy:0.793

foo [[ 0.25176435  1.          0.          1.          0.        ]
 [ 0.67529396  0.          1.          0.          0.        ]
 [ 0.43803523  1.          1.          0.          0.        ]
 ...,
 [ 0.27999966  1.          0.          0.          1.        ]
```

Decision trees with scaled values

In [87]:
```
##X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled
clf = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3,min_samples_le

clfScaledage = clf.fit(X_train_scaled,y_train_scaled)

measure_performance(X_test_scaled,y_test_scaled,clfScaledage, show_classification
```

```
Accuracy:0.793

Classification report
             precision    recall  f1-score   support

        0.0       0.77      0.96      0.85       202
        1.0       0.88      0.54      0.67       127

avg / total       0.81      0.79      0.78       329


Confusion matrix
[[193   9]
 [ 59  68]]
```

Decision trees with MaxMin Scaling

In [88]:
```
#X_train_maxmin, X_test_maxmin, y_train_maxmin, y_test_maxmin
clf = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3,min_samples_le


clfMaxMinage = clf.fit(X_train_maxmin,y_train_maxmin)

measure_performance(X_test_maxmin,y_test_maxmin,clfMaxMinage, show_classification
```

```
Accuracy:0.793

Classification report
             precision    recall  f1-score   support

        0.0       0.77      0.96      0.85       202
        1.0       0.88      0.54      0.67       127

avg / total       0.81      0.79      0.78       329
```

```
Confusion matrix
[[193   9]
 [ 59  68]]
```

why is scaling the age making no difference, it is essentially done internally

Ue Gini instead of entropy

In [89]:

```
clf = tree.DecisionTreeClassifier(criterion='gini', max_depth=3,min_samples_leaf=
clfv1 = clf.fit(X_train,y_train)
measure_performance(X_test,y_test,clfv1, show_classification_report=True, show_co
```

```
Accuracy:0.793

Classification report
             precision    recall  f1-score   support

        0.0       0.77      0.96      0.85       202
        1.0       0.88      0.54      0.67       127

avg / total       0.81      0.79      0.78       329


Confusion matrix
[[193   9]
 [ 59  68]]
```

Use Extra tree Classifier

In [90]:

```
from sklearn.ensemble import ExtraTreesClassifier
extclf = ExtraTreesClassifier(n_estimators=10, max_depth=None,min_samples_split=1
clfv1fit = extclf.fit(X_train,y_train)
measure_performance(X_test,y_test,clfv1fit, show_classification_report=True, show
```

```
Accuracy:0.778

Classification report
             precision    recall  f1-score   support

        0.0       0.77      0.91      0.83       202
        1.0       0.79      0.57      0.67       127
```

```
Confusion matrix
[[183  19]
 [ 54  73]]
```

Leave one out cross validation

In [91]:

```
from sklearn.cross_validation import cross_val_score, LeaveOneOut
from scipy.stats import sem

def loo_cv(X_train,y_train,clf):
    # Perform Leave-One-Out cross validation
    # We are preforming 1313 classifications!
    loo = LeaveOneOut(X_train[:].shape[0])
    scores=np.zeros(X_train[:].shape[0])
    for train_index,test_index in loo:
        X_train_cv, X_test_cv= X_train[train_index], X_train[test_index]
        y_train_cv, y_test_cv= y_train[train_index], y_train[test_index]
        clf = clf.fit(X_train_cv,y_train_cv)
        y_pred=clf.predict(X_test_cv)
        scores[test_index]=metrics.accuracy_score(y_test_cv.astype(int), y_pred.a
    print ("Mean score: {0:.3f} (+/-{1:.3f})").format(np.mean(scores), sem(scores
```

Perform leave-one-out cross validation to better measure performance, reducing variance. LeaveOneOut (or LOO) is a simple cross-validation. Each learning set is created by taking all the samples except one, the test set being the sample left out. Thus, for n samples, we have n different training sets and n different tests set. This cross-validation procedure does not waste much data as only one sample is removed from the training set:

In [92]:

```
loo_cv(titanic_X, titanic_y,clf)
```

```
Mean score: 0.811 (+/-0.011)
```

Random Forests - can we improve predictive accuracy by using a different learner

Try to improve performance using Random Forests

In [93]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [94]:
```
loo_cv(titanic_X,titanic_y,clfRF)
```

```
Mean score: 0.813 (+/-0.011)
```

with split validation

In [95]:
```
clfRFIT = clfRF.fit(X_train,y_train)
```

In [96]:
```
measure_performance(X_test,y_test,clfRFIT, show_classification_report=False, show
```

```
Accuracy:0.778
```

a small drop in observed performance

with 10 fold cross validation

In [97]:
```
from sklearn.cross_validation import KFold


def ten_fold_Cross_Validation_train_and_evaluate(clf, X_train, y_train):
   # Perform Leave-One-Out cross validation
    # We are preforming 1313 classifications!
    # create a k-fold croos validation iterator of k=10 folds
    cv = KFold(X_train.shape[0], 10, shuffle=True, random_state=33)
    scores=np.zeros(X_train[:].shape[0])
    for train_index,test_index in cv:
        X_train_cv, X_test_cv= X_train[train_index], X_train[test_index]
        y_train_cv, y_test_cv= y_train[train_index], y_train[test_index]
        clf = clf.fit(X_train_cv,y_train_cv)
        y_pred=clf.predict(X_test_cv)
        scores[test_index]=metrics.accuracy_score(y_test_cv.astype(int), y_pred.a


    print "Predictive accuracy using 10-fold crossvalidation:",np.mean(scores)
```

In [98]:
```
ten_fold_Cross_Validation_train_and_evaluate(clfRF,titanic_X,titanic_y)
```

```
Predictive accuracy using 10-fold crossvalidation: 0.805728271135
```

fold cross validation

Using other learners, bagging, knn

In [99]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
bagging = BaggingClassifier(KNeighborsClassifier(),
                            max_samples=0.5, max_features=0.5)
```

In [100]:
```
bagging.fit(X_train,y_train)
```

Out[100]:
```
BaggingClassifier(base_estimator=KNeighborsClassifier(algorithm='auto', leaf_siz
e=30, metric='minkowski',
           metric_params=None, n_neighbors=5, p=2, weights='uniform'),
         bootstrap=True, bootstrap_features=False, max_features=0.5,
         max_samples=0.5, n_estimators=10, n_jobs=1, oob_score=False,
         random_state=None, verbose=0)
```

In [101]:
```
measure_performance(X_test,y_test,bagging)
```

```
Accuracy:0.793

Classification report
             precision    recall  f1-score   support

        0.0       0.77      0.95      0.85       202
        1.0       0.86      0.55      0.67       127

avg / total       0.81      0.79      0.78       329


Confusion matrix
[[191  11]
 [ 57  70]]
```

so lets try K-nn on its own

In [102]:
```
n_neighbors = 5
clfKNN=KNeighborsClassifier(n_neighbors)
```

In [103]:
```
clfKNN.fit(X_train,y_train)
```

Out[103]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
```

In [104]:
```
measure_performance(X_test,y_test,clfKNN)
```

```
Accuracy:0.793

Classification report
             precision    recall  f1-score   support

        0.0       0.78      0.93      0.85       202
        1.0       0.84      0.57      0.68       127

avg / total       0.80      0.79      0.78       329


Confusion matrix
[[188  14]
 [ 54  73]]
```

In [105]:
```
X_train, X_test, y_train, y_test = train_test_split(titanic_X, titanic_y, test_si
```

10 fold Cross Validation

In [106]:
```
clfKNNCV=KNeighborsClassifier(n_neighbors)
```

n k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k − 1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used. This is what we have implemented in above 10 fold cross validation.

In [106]:
```

```

ten_fold_Cross_Validation_train_and_evaluate(clfKNNCV,titanic_X,titanic_y)

Using Pandas with Bokeh

In [107]:
```
import mpld3
```

BokehJS successfully loaded.

In [109]:
```
from bokeh.plotting import figure, HBox, output_file, show, VBox
from bokeh.models import Range1d
```

In [110]:
```

from bokeh.plotting import *
```

In [111]:
```
test7 = titanic_invest.groupby(['pclass','survived'])['name'].count().unstack('su
```

In [112]:
```
print test7
```

```
survived    0    1
pclass
1st       129  193
2nd       161  119
3rd       574  137
```

In [113]:
```
cabin_classes=test7.index.tolist() #get a list of index
```

In [114]:
```
test7.columns.tolist() #get a list of columns
```

Out[114]:
```
[0, 1]
```

In [115]:
```
Not_Surv=test7.loc[:,0].values
Surv=test7.loc[:,1].values
```

In [117]:
```
# create a figure()
p1 = figure(title="Survivors and Non Survivors BY cabin class (stacked)", tools="
            x_range=cabin_classes, y_range=[0, max(Surv+Not_Surv)],
            background_fill='#59636C', plot_width=800
    )

# use the `rect` renderer to display stacked bars of the medal results. Note
# that we set y_range explicitly on the first renderer
p1.rect(x=cabin_classes, y=Surv/2, width=0.8, height=Surv, color="blue", alpha=0.
p1.rect(x=cabin_classes, y=Surv+Not_Surv/2, width=0.8, height=Not_Surv, color="re
```

Out[117]:
```
<bokeh.plotting.Figure at 0x1180cab0>
```

In [119]:
```
show(p1)
```

In [119]:
```

```

In [138]:
```
import vincent
```

In [139]:
```
from vincent import AxisProperties, PropertySet, ValueRef
```

In [140]:
```
titanic_invest.columns.tolist()
```

Out[140]:
```
['row.names',
 'pclass',
 'survived',
 'name',
 'age',
 'embarked',
 'home.dest',
 'room',
 'ticket',
 'boat',
 'sex',
 'surv_class']
```

In [141]:
```
titanic_invest.ix[:,4].fillna(titanic_invest.ix[:,4].mean(),inplace=True) ##repla
```

In [142]:
```
titanic_invest["surv_class"] = titanic_invest["pclass"] + titanic_invest["survive
```

In [143]:
```
data=titanic_invest[['age','pclass','surv_class']]
```

In [144]:
```
test7
```

Out[144]:

survived	0	1
pclass		
1st	129	193
2nd	161	119
3rd	574	137

In [162]:
```
vincent.core.initialize_notebook()
```

```
grouped = vincent.GroupedBar(test7)
grouped.axis_titles(x='pclass', y='survived')
grouped.legend(title='class survival')
grouped.to_json('vega.json')
grouped.display()
```

In [164]:

```
```

In [133]:

```
```

In [ ]:

```
```