In [1]:
```
import numpy
import urllib
import scipy.optimize
import random
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn import svm
```

In [2]:
```
#read bear information file on website
def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

print "Reading data..."
data = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.js
on"))
print "done"
```

```
Reading data...
done
```

In [3]:
```
#an overview of features
print(data[0])
```


```
{'beer/style': 'Hefeweizen', 'beer/ABV': 5.0, 'beer/beerId': '47986', 'revie
w/timeStruct': {'wday': 0, 'isdst': 0, 'mday': 16, 'hour': 20, 'min': 57, 'se
c': 3, 'year': 2009, 'yday': 47, 'mon': 2}, 'review/aroma': 2.0, 'review/appe
arance': 2.5, 'review/timeUnix': 1234817823, 'review/palate': 1.5, 'review/ta
ste': 1.5, 'beer/name': 'Sausa Weizen', 'beer/brewerId': '10325', 'review/ove
rall': 1.5, 'review/text': 'A lot of foam. But a lot.\tIn the smell some bana
na, and then lactic and tart. Not a good start.\tQuite dark orange in color,
with a lively carbonation (now visible, under the foam).\tAgain tending to la
ctic sourness.\tSame for the taste. With some yeast and banana.', 'user/profi
leName': 'stcules'}
```

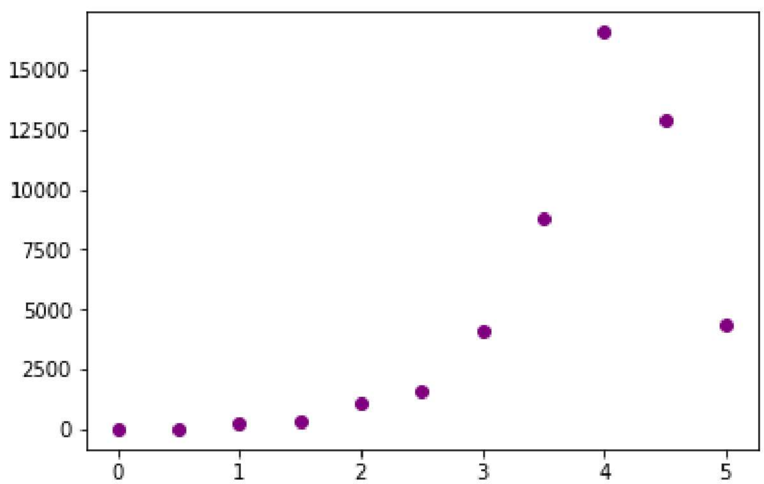

1.

In [4]:
```
#count reviews with different number of stars
rev=[d['review/taste'] for d in data if d.has_key('review/taste')]
def starnu(st,rev):
    nu = 0
    for a in rev:
        if a==st:
            nu=nu+1
    return nu
```

In [5]:
```
#there are 11 different numbers of stars include 0, 0.5, 1,...,4.5,5
stars = numpy.zeros(11)
for i in range(11):
    stars[i]=int(starnu(i/2.0,rev))
stars =(numpy.array(stars))
xaxis=numpy.linspace(0.0,5.0,num=11)
#print how many number of reviews for each number of stars
print('There are ')
for i in range(len(xaxis)):
    print stars[i],' for ',xaxis[i],' Stars, '
print('ratings in the dataset (for review/taste)')
```

```
There are
0.0  for  0.0  Stars,
0.0  for  0.5  Stars,
211.0  for  1.0  Stars,
343.0  for  1.5  Stars,
1099.0  for  2.0  Stars,
1624.0  for  2.5  Stars,
4137.0  for  3.0  Stars,
8797.0  for  3.5  Stars,
16575.0  for  4.0  Stars,
12883.0  for  4.5  Stars,
4331.0  for  5.0  Stars,
ratings in the dataset (for review/taste)
```

In [6]:
```
#show how many reviews with each number of stars in plot
fig=plt.scatter(xaxis,stars,color='purple')
plt.show()
```



2.

In [7]:
```
#raw beer name list
beerna = [d['beer/name'] for d in data if d.has_key('review/overall')]
```

In [9]:
```
nalist=[]
count=1
pre=beerna[0]
#count each beer name's numbers
for i in range(1,len(beerna)):
    if beerna[i]==pre:
        count=count+1
    else:
        nalist.append((pre,count))
        count=1
    pre=beerna[i]
```

In [10]:
```
#sum up the number of beer whose names are same but appear sparsely/not cluste
r together
#store name as key and count as value in a dictionary
nasum={}
for i in range(len(nalist)):
    bi=nalist[i]
    if nasum.has_key(bi[0])==False:
        nasum[bi[0]]=bi[1]
    else:
        nasum[bi[0]]=nasum[bi[0]]+bi[1]
```

In [11]:
```
#create a dictionary to store beer names with >=5 reviews and their counts
nalist5={}
for key,count in nasum.iteritems():
    if count>=5:
        nalist5[key]=count
```

In [12]:
```
narev = [(d['beer/name'],d['review/overall']) for d in data]
```

In [69]:
```
#get the average stars of review/overall for each beer with >= 5 reviews
avg=0
prena=0
avgstar=[]
for index in range(len(narev)):
    i=narev[index]
    if i[0] in nalist5:
        if i[0]==prena:
            avg=avg+i[1]
            continue
    if prena in nalist5:
        count=nalist5[prena]
        avg=avg/count
        avgstar.append((prena,avg))
    prena=i[0]
    avg=i[1]
```

In [70]:
```
#sort the review list from high rating to low rating
avgstar.sort(key=lambda tup: tup[1], reverse=True)
```

In [15]:
```
#all of information about the highest reviewed beer
#'Wobbly Bob APA' with its 4.714285714285714 ratings
hi=filter(lambda beer: beer['beer/name'] == 'Wobbly Bob APA', data)
```

3.

In [16]:
```
#make sure that all of the 50000 items have abv and style properties
print(len([d for d in data if d.has_key('beer/ABV')]))
print(len([d for d in data if d.has_key('beer/style')]))
```

```
50000
50000
```

In [17]:
```
#Contruct a X matrix with a column filled with 1, a column with one hot encodi
ng for whether beer is a Hefeweizen
#and a collumn of ABV score
hef = numpy.zeros(shape=(len(data),3))
for i in range(len(data)):
    hef[i][0]=1
    if data[i]['beer/style']=='Hefeweizen':
        hef[i][1]= 1
    hef[i][2] = data[i]['beer/ABV']
print(hef.shape)
#Construct the y array of review/taste
revtaste = numpy.array([d['review/taste'] for d in data]).reshape(len(data),1)
print(revtaste.shape)
```

```
(50000L, 3L)
(50000L, 1L)
```

In [19]:
```
#Use linear regression to train model
reg1 = LinearRegression()
reg1.fit(hef,revtaste)
#get the value of theta
print(reg1.intercept_,reg1.coef_)
#test model
pred1 = reg1.predict(hef)
mse1=metrics.mean_squared_error(revtaste,pred1)
print(mse1)
```

```
(array([3.11795084]), array([[ 0.        , -0.05637406,  0.10877902]]))
0.4496582550241641
```

$\theta_0 = 3.11795084, \theta_1 = -0.05637406, \theta_2 = 0.10877902$ θ_0 is a constant, θ_1 is the weight assigned to [beer is a Hefeweizen] and θ_2 is the weight assigned to beer/ABV. θ is the feature weights vector.

4.

In [20]:
```
#split dataset into training and testing
Xtrain,Xtest=numpy.split(hef,2)
Ytrain,Ytest=numpy.split(revtaste,2)
#train model on the training set
reg2 = LinearRegression()
reg2.fit(Xtrain,Ytrain)
#theta_0 is 2.99691466, theta_1 is -0.03573098, theta_2 is 0.11672256
print(reg2.intercept_,reg2.coef_)
#test model
pred2 = reg2.predict(Xtest)
#model's mse on testing set is 0.4237065211985226
mse2=metrics.mean_squared_error(Ytest,pred2)
pred3 = reg2.predict(Xtrain)
#mse on training set is 0.4839680560134243
mse3=metrics.mean_squared_error(Ytrain,pred3)
print(mse2,mse3)
```

```
(array([2.99691466]), array([[ 0.        , -0.03573098,  0.11672256]]))
(0.4237065211985226, 0.4839680560134243)
```

5.

In [65]:
```
#shuffle the dataset so that training and testing dataset can be randomly sele
cted
mix = numpy.concatenate((revtaste,hef),axis=1)
numpy.random.shuffle(mix)
Xstrain,Xstest=numpy.split(mix[:,1:],2)
Ystrain,Ystest=numpy.split(mix[:,0],2)
Ystrain = Ystrain.reshape(len(Ystrain),1)
Ystest = Ystest.reshape(len(Ystest),1)
#train model on the training set
reg3 = LinearRegression()
reg3.fit(Xstrain,Ystrain)
#theta_0 is 3.11403742, theta_1 is -0.08077046, theta_2 is 0.10983618
print(reg3.intercept_,reg3.coef_)
#test model
pred4 = reg3.predict(Xstest)
#mse on testing set is 0.4506784358424855
mse4=metrics.mean_squared_error(Ystest,pred4)
pred5 = reg3.predict(Xstrain)
#mse on training set is 0.44869336329655113
mse5=metrics.mean_squared_error(Ystrain,pred5)
print(mse4,mse5)
```

```
(array([3.11403742]), array([[ 0.        , -0.08077046,  0.10983618]]))
(0.4506784358424855, 0.44869336329655113)
```

We observe that in problem 4, mse on testing set is much smaller than that on training set. In problem 5, mse on testing set is a bit larger than mse on training set. This difference probably because the dataset is shuffled in problem 5, and thus training sets and testing sets become more balanced. Without randomly selecting, splitting the original dataset directly into two parts in problem 4 may lead to biased training and testing sets, and thus cause abnormal mse.

6.

In [22]:
```
#make sure that all of the 50000 items have these features
print(len([d for d in data if d.has_key('review/taste')]))
print(len([d for d in data if d.has_key('review/appearance')]))
print(len([d for d in data if d.has_key('review/aroma')]))
print(len([d for d in data if d.has_key('review/palate')]))
print(len([d for d in data if d.has_key('review/overall')]))
```

```
50000
50000
50000
50000
50000
```

In [66]:
```
#random split train and test dataset
Xfive=numpy.array([[d['review/taste'],d['review/appearance'],d['review/aroma'
],d['review/palate'],d['review/overall']] for d in data])
hefy=hef[:,1]
hefy=hefy.reshape(len(hefy),1)
mix2 = numpy.concatenate((hefy,Xfive),axis=1)
numpy.random.shuffle(mix2)
Xstrain,Xstest=numpy.split(mix2[:,1:],2)
Ystrain,Ystest=numpy.split(mix2[:,0],2)
Ystrain = Ystrain.reshape(len(Ystrain))
Ystest = Ystest.reshape(len(Ystest))
```

In [24]:
```
#get the accuracy of prediction
def acc(Ypred,Y):
    prob=0
    for i in range(len(Y)):
        if Ypred[i]==Y[i]:
            prob=prob+1
    prob=1.0*prob/len(Y)
    return prob
```

In [25]:
```
#Train SVM classifier
clf1 = svm.SVC(C=1000,kernel='linear')
clf1.fit(Xstrain,Ystrain)
```

Out[25]:
```
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [26]:
```
#The testing accuracy is 0.98752 and the training accuracy is 0.98766
predict=clf1.predict(Xstest)
print(acc(predict,Ystest))
predict=clf1.predict(Xstrain)
print(acc(predict,Ystrain))
```

```
0.98752
0.98776
```

7.

In [30]:
```
#We want to compare the accurancy under different kernals
#To speed up, we use C=100 instead of 1000
clf11 = svm.SVC(C=100,kernel='linear')
clf11.fit(Xstrain,Ystrain)
predict=clf11.predict(Xstest)
print(acc(predict,Ystest))
predict=clf11.predict(Xstrain)
print(acc(predict,Ystrain))
```

```
0.98752
0.98776
```

In [25]:
```
clf2 = svm.SVC(C=100,kernel='sigmoid')
clf2.fit(Xstrain,Ystrain)
```

Out[25]:
```
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='sigmoid',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [26]:
```
predict=clf2.predict(Xstest)
print(acc(predict,Ystest))
predict=clf2.predict(Xstrain)
print(acc(predict,Ystrain))
```

```
0.976
0.97444
```

In [27]:
```
clf3 = svm.SVC(C=100)
clf3.fit(Xstrain,Ystrain)
```

Out[27]:
```
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [28]:
```
#We observe that rbf model has higher training accuracy, so we choose rbf as o
ur kernal
predict=clf3.predict(Xstest)
print(acc(predict,Ystest))
predict=clf3.predict(Xstrain)
print(acc(predict,Ystrain))
```

```
0.98816
0.98712
```

In [43]:
```
#To boost the model performance, we search the most frequently appeared words
 in Hefeweizen-style-beer
from collections import Counter
words=numpy.array([d['review/text'] for d in data if d['beer/style']=='Hefewei
zen'])
words=' '.join(words)
spl = words.split()
Counter = Counter(spl)
most_occur = Counter.most_common(30)
print(most_occur)
```

```
[('a', 3245), ('the', 2333), ('and', 2210), ('of', 1807), ('with', 1262), ('i
s', 1248), ('to', 922), ('I', 777), ('but', 709), ('in', 686), ('The', 664),
('this', 611), ('that', 577), ('it', 486), ('on', 471), ('was', 446), ('as',
408), ('very', 402), ('wheat', 391), ('for', 372), ('not', 360), ('beer', 35
7), ('head', 353), ('banana', 338), ('some', 337), ('light', 330), ('A', 31
0), ('white', 287), ('good', 280), ('from', 270)]
```

In [51]:
```
#Ignoring words that are not helpful, we choose 'wheat','head','banana','ligh
t','white' as our keywords
#Construct X matrix by one hot encoding
key = numpy.zeros(shape=(len(data),5))
for i in range(len(data)):
    if 'wheat' in data[i]['review/text']:
        key[i][0]=1
    if 'head' in data[i]['review/text']:
        key[i][1]=1
    if 'banana' in data[i]['review/text']:
        key[i][2]=1
    if 'light' in data[i]['review/text']:
        key[i][3]=1
    if 'white' in data[i]['review/text']:
        key[i][4]=1
#key is the feature vector of which the 1st, 2nd, 3rd, 4th, 5th columns corres
pond to
#the boolean of the existence of word 'wheat','head','banana','light','white'
 respectively in every review text
key=numpy.array(key)
mix3 = numpy.concatenate((hefy,key),axis=1)
#shuffle the dataset
numpy.random.shuffle(mix3)
Xsstrain,Xsstest=numpy.split(mix3[:,1:],2)
Ysstrain,Ysstest=numpy.split(mix3[:,0],2)
Ysstrain = Ysstrain.reshape(len(Ysstrain))
Ysstest = Ysstest.reshape(len(Ysstest))
```

In [57]:
```
#With this method we not only improve the accuracy, but also make the speed mu
ch faster ~ ^ O ^ ~
clf8 = svm.SVC(C=100)
clf8.fit(Xsstrain,Ysstrain)
#Compared with testing accuracy 0.98816, training accuracy 0.98712 with the or
iginal feature vector and rbf kernal
#Both training and testing accuracys are improved with the new feature vector
predict=clf8.predict(Xsstest)
#testing accuracy=0.98852
print(acc(predict,Ysstest))
#training accuracy=0.98972
predict=clf8.predict(Xsstrain)
print(acc(predict,Ysstrain))
```

```
0.98852
0.98972
```

Now let's try C with [0.1,10,1000,100000]

In [67]:
```
clf99 = svm.SVC(C=0.1)
clf99.fit(Xsstrain,Ysstrain)
predict=clf99.predict(Xsstest)
print(acc(predict,Ysstest))
predict=clf99.predict(Xsstrain)
print(acc(predict,Ysstrain))
```

```
0.98744
0.98784
```

In [59]:
```
clf10 = svm.SVC(C=10)
clf10.fit(Xsstrain,Ysstrain)
predict=clf10.predict(Xsstest)
print(acc(predict,Ysstest))
predict=clf10.predict(Xsstrain)
print(acc(predict,Ysstrain))
```

```
0.9886
0.98968
```

In [60]:
```
clf11 = svm.SVC(C=1000)
clf11.fit(Xsstrain,Ysstrain)
predict=clf11.predict(Xsstest)
print(acc(predict,Ysstest))
predict=clf11.predict(Xsstrain)
print(acc(predict,Ysstrain))
```

```
0.98852
0.98976
```

In [61]:
```
clf12 = svm.SVC(C=10000)
clf12.fit(Xsstrain,Ysstrain)
predict=clf12.predict(Xsstest)
print(acc(predict,Ysstest))
predict=clf12.predict(Xsstrain)
print(acc(predict,Ysstrain))
```

```
0.98852
0.98976
```

There are little difference when C varies. Let's try more C values to see if the difference will be more obvious

In [63]:
```
#Seems like no matter how C large, the accuracy converges to (0.98852,0.98976)
clf13 = svm.SVC(C=1000000)
clf13.fit(Xsstrain,Ysstrain)
predict=clf13.predict(Xsstest)
print(acc(predict,Ysstest))
predict=clf13.predict(Xsstrain)
print(acc(predict,Ysstrain))
```

```
0.98852
0.98976
```

In [68]:
```
#Is there any lower bound when C is very small?
#The training and testing accuracy are same as when C=0.1
clf14 = svm.SVC(C=0.00000001)
clf14.fit(Xsstrain,Ysstrain)
predict=clf14.predict(Xsstest)
print(acc(predict,Ysstest))
predict=clf14.predict(Xsstrain)
print(acc(predict,Ysstrain))
```

```
0.98744
0.98784
```

For this model, we observe that the lowest training accuracy is 0.98784, the lowest testing accuracy is 0.98744; the highest training accuracy is 0.98976, the highest testing accuracy is 0.98852. A high C helps us train a classifier to correctly classify as many training sample as possible, but the generalization ability of model decreases. A low C neglects some misclassified outliers and looks for a large-margin seperating hyper plane, which may lead to larger error. For this dataset, training and testing accuracy increases with the growth of C.