1 a). $\arg\min_{f(x)} E_{Y|x} \exp(-\frac{1}{K}(Y_1 f_1 + \cdots Y_K f_K))$ subject to $f_1 + \cdots f_K = 0$

The lagrange of this constrained optimization problem can be written as:

$$\exp(-\frac{f_1(x)}{K-1}) Prob(C=1|x) + \cdots + \exp(-\frac{f_K(x)}{K-1}) Prob(C=K|x) - \lambda(f_1(x) + \cdots + f_K(x)).$$

Where λ is the lagrange multiplier. Taking derivative with respect to $f(k)$ and λ:

$$-\frac{1}{K-1}\exp(-\frac{f_1(x)}{K-1}) Prob(C=1|x) - \lambda = 0$$

$$\vdots$$

$$-\frac{1}{K-1}\exp(-\frac{f_K(x)}{K-1}) Prob(C=K|x) - \lambda = 0$$

$$f_1(x) + \cdots f_K(x) = 0$$

$\therefore$ $f_k^*(x) = (K-1)\left(\log Prob(C=k|x) - \frac{1}{K}\sum \log Prob(C=k'|x)\right)$ $k = 1 \cdots K$

$$\begin{cases} \arg\max_k f_k^*(x) = \arg\max_k Prob(C=k|x) \\ Prob(C=k|x) = \dfrac{e^{\frac{1}{K-1}f_k^*(x)}}{e^{\frac{1}{K-1}f_1^*(x)} + \cdots e^{\frac{1}{K-1}f_K^*(x)}} \quad k = 1 \cdots K. \end{cases}$$

1. b). Initialize the observation weights $w_i = 1/n$

from 1 to M:

$$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}(C_i \neq T^{(m)}(x_i)) / \sum_{i=1}^{n} w_i$$

$$\alpha^{(m)} = \log\frac{1-err^m}{err^m} + \log(k-1)$$

$$w_i = w_i \cdot \exp(\alpha^m \mathbb{I}(C_i \neq T^{(m)}(x_i)))$$

Output: $C(x) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(x) = k)$

The major difference between Adaboost and this new algorithm is the $\alpha^{(m)}$, where in New Algorithm, the $\alpha^{(m)} = \log\frac{1-err^m}{err^m} + \log(k-1)$

In AdaBoost: the $\alpha^m = \frac{1}{2}\log\frac{1-Err^m}{Err^m}$

HW3#2

Function to Import train data and do pre-processing

In [42]:

```
import numpy as np
import pandas as pd
from sklearn import tree
from sklearn import model_selection
import matplotlib.pyplot as plt
from sklearn.externals.six import StringIO
#import pydotplus
from IPython.display import Image
from sklearn import metrics
%matplotlib inline



def data_processing(file):
    train=file.readlines()

    train_list=[]

    for line in train:
        line_replace=line.replace('?','mode')
        line_list=line_replace.split(",")
        #line_list=line_list.split(" ")
        for i in range(len(line_list)):
            if line_list[i].isdigit()==True:
                line_list[i]=int(line_list[i])

        train_list.append(line_list)
    train_list.pop()
    train_array=np.array(train_list)

    classVotes={}
    for i in range(len(train_list[0])):
        if type(train_list[0][i])==int:
            data=train_array[:,i]
            s=0
            num=1
            for j in data:
                s+=int(j)
                num+=1
        classVotes[i]=s/num
    for i in range(len(train_list[0])):
```

```
        if type(train_list[0][i])==int:
            for j in range(len(train_list)):
                if train_list[j][i]=='mode':
                    train_list[j][i].replace('classVotes[i]','mode')

    return train_list

file1=open('/Users/wendy/Documents/2017 Fall/CS 534/HW3/adult-test.csv')
file2=open('/Users/wendy/Documents/2017 Fall/CS 534/HW3/adult-data.csv')

def data_processing1(file):
    train=file.readlines()

    train_list=[]

    for line in train:
        line_replace=line.replace('?','mode')
        line_list=line_replace.split(", ")
        #line_list=line_list.split(" ")
        for i in range(len(line_list)):
            if line_list[i].isdigit()==True:
                line_list[i]=int(line_list[i])

        train_list.append(line_list)
    train_list.pop()
    train_array=np.array(train_list)

    classVotes={}
    for i in range(len(train_list[0])):
        if type(train_list[0][i])==int:
            data=train_array[:,i]
            s=0
            num=1
            for j in data:
                s+=int(j)
                num+=1
        classVotes[i]=s/num
    for i in range(len(train_list[0])):
        if type(train_list[0][i])==int:
            for j in range(len(train_list)):
                if train_list[j][i]=='mode':
                    train_list[j][i].replace('classVotes[i]','mode')

    return train_list


train=data_processing(file2)
test=data_processing1(file1)
print train[0]
print test[0]
import csv
t=open("train.csv","wb")
writer=csv.writer(t)
writer.writerows(train)
```

```
te=open("test.csv","wb")
writer=csv.writer(te)
writer.writerows(test)
cname=['a','b','c','d',"e",'f','g','h','i','j','k','l','m','n','o']
train_data = pd.read_csv('/Users/wendy/Documents/2017 Fall/CS 534/HW3/train.csv',nar
test_data=pd.read_csv('/Users/wendy/Documents/2017 Fall/CS 534/HW3/test.csv',names=c
#print test_data
```

```
[39, ' State-gov', 77516, ' Bachelors', 13, ' Never-married', ' Adm-cl
erical', ' Not-in-family', ' White', ' Male', 2174, 0, 40, ' United-St
ates', ' <=50K\r\n']
[25, 'Private', 226802, '11th', 7, 'Never-married', 'Machine-op-inspct
', 'Own-child', 'Black', 'Male', 0, 0, 40, 'United-States', '<=50K.\n'
]
```

Data transformation and convert training_y into numeric categories

In [31]:

```
bDummies = pd.get_dummies(train_data.b, prefix='b').iloc[:, 1:]
dDummies = pd.get_dummies(train_data.d, prefix='d').iloc[:, 1:]
fDummies = pd.get_dummies(train_data.f, prefix='f').iloc[:, 1:]
gDummies = pd.get_dummies(train_data.g, prefix='g').iloc[:, 1:]
hDummies = pd.get_dummies(train_data.g, prefix='h').iloc[:, 1:]
iDummies = pd.get_dummies(train_data.g, prefix='i').iloc[:, 1:]
jDummies = pd.get_dummies(train_data.g, prefix='j').iloc[:, 1:]
nDummies = pd.get_dummies(train_data.g, prefix='n').iloc[:, 1:]

trainDF = pd.concat([train_data, bDummies,dDummies,fDummies,gDummies,hDummies,iDumm
print trainDF.columns

y = trainDF.o
trainx = trainDF.drop(trainDF.columns[[1,3,5,6,7,8,9,13,14]], axis=1)
trainy=[]
for i in y:
    if i==' <=50K\r\n':
        trainy.append(0)
    else:
        trainy.append(1)
```

```
Index([u'a', u'b', u'c', u'd', u'e', u'f', u'g', u'h', u'i', u'j',
       ...
       u'n_ Handlers-cleaners', u'n_ Machine-op-inspct', u'n_ Other-se
rvice',
       u'n_ Priv-house-serv', u'n_ Prof-specialty', u'n_ Protective-se
rv',
       u'n_ Sales', u'n_ Tech-support', u'n_ Transport-moving', u'n_ m
ode'],
      dtype='object', length=114)
```

Decision_tree Function

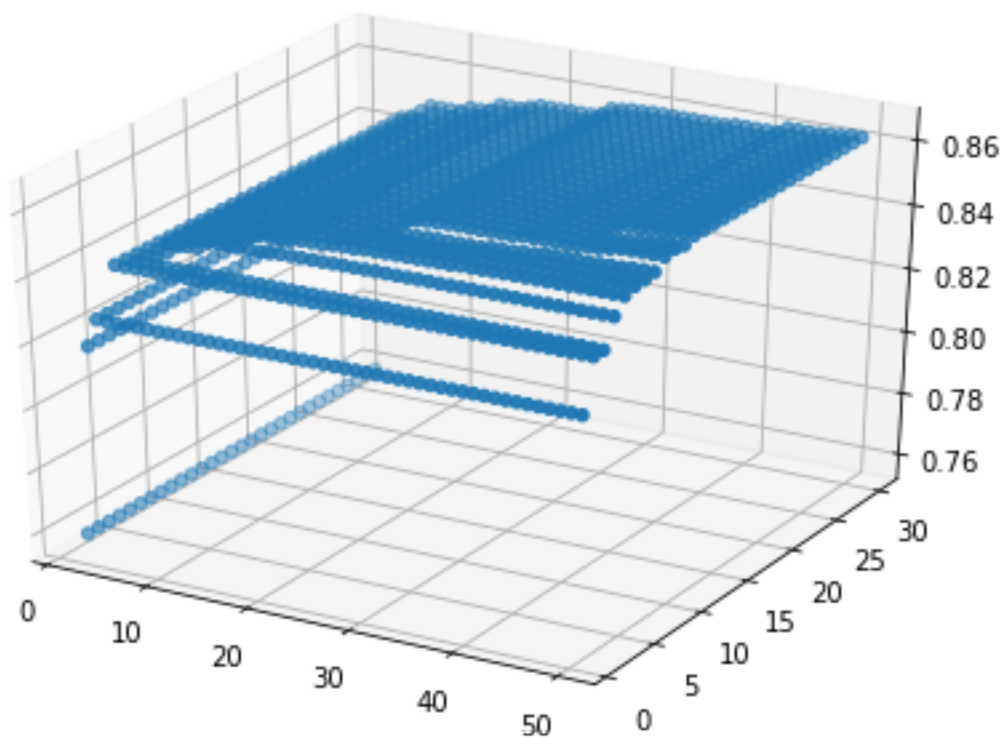
In [34]:

```
def D_Tree(i,j):
    clf = tree.DecisionTreeClassifier(max_depth=i,max_leaf_nodes=j)
    clf.fit(trainx, trainy)
    #score = model_selection.cross_val_score(clf,trainx, trainy, cv=5,scoring='accu

    trainyHat = clf.predict(trainx)
    err=0
    for a in range(len(trainyHat)):
        if trainyHat[a]!=trainy[a]:
            err+=1
    return float ((len(trainy)-float(err))/len(trainy))

#print "Train", metrics.classification_report(trainy, trainyHat)
```

plot the accuracy as a function of depth and num of leaves

In [43]:

```
accuracy_list=[]
accuracy=[]
depth_list=[]
#depth_list=list(range(1,3))
leaf_num_list=[]
for i in range(1,31):
    for j in range (2,51):
        depth_list.append(i)
        leaf_num_list.append(j)
        accuracy.append(D_Tree(i,j) )
        accuracy_list.append((i,j,D_Tree(i,j)))

from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig=plt.figure()
ax=Axes3D(fig)
ax.scatter(leaf_num_list,depth_list,accuracy)
plt.show()
```



Find the optimal depth and leaves

In [44]:

```
from operator import itemgetter
sorted_accuracy=sorted(accuracy_list,key=itemgetter(2),reverse=True)
print "the optimal depth is:",sorted_accuracy[0][0],"the optimal #of leaves is:",sor
```

```
the optimal depth is: 12 the optimal #of leaves is: 49 with the accura
cy: 0.862223587224
```

Test data transformation

In [51]:

```
bDummies = pd.get_dummies(test_data.b, prefix='b').iloc[:, 1:]
dDummies = pd.get_dummies(test_data.d, prefix='d').iloc[:, 1:]
fDummies = pd.get_dummies(test_data.f, prefix='f').iloc[:, 1:]
gDummies = pd.get_dummies(test_data.g, prefix='g').iloc[:, 1:]
hDummies = pd.get_dummies(test_data.g, prefix='h').iloc[:, 1:]
iDummies = pd.get_dummies(test_data.g, prefix='i').iloc[:, 1:]
jDummies = pd.get_dummies(test_data.g, prefix='j').iloc[:, 1:]
nDummies = pd.get_dummies(test_data.g, prefix='n').iloc[:, 1:]

testDF = pd.concat([test_data, bDummies,dDummies,fDummies,gDummies,hDummies,iDummies
y = testDF.o
testx = testDF.drop(testDF.columns[[1,3,5,6,7,8,9,13,14]], axis=1)

testy=[]
for i in y:
    if i=='<=50K.\n':
        testy.append(0)
    else:
        testy.append(1)
```

Accuracy on test data

In [53]:

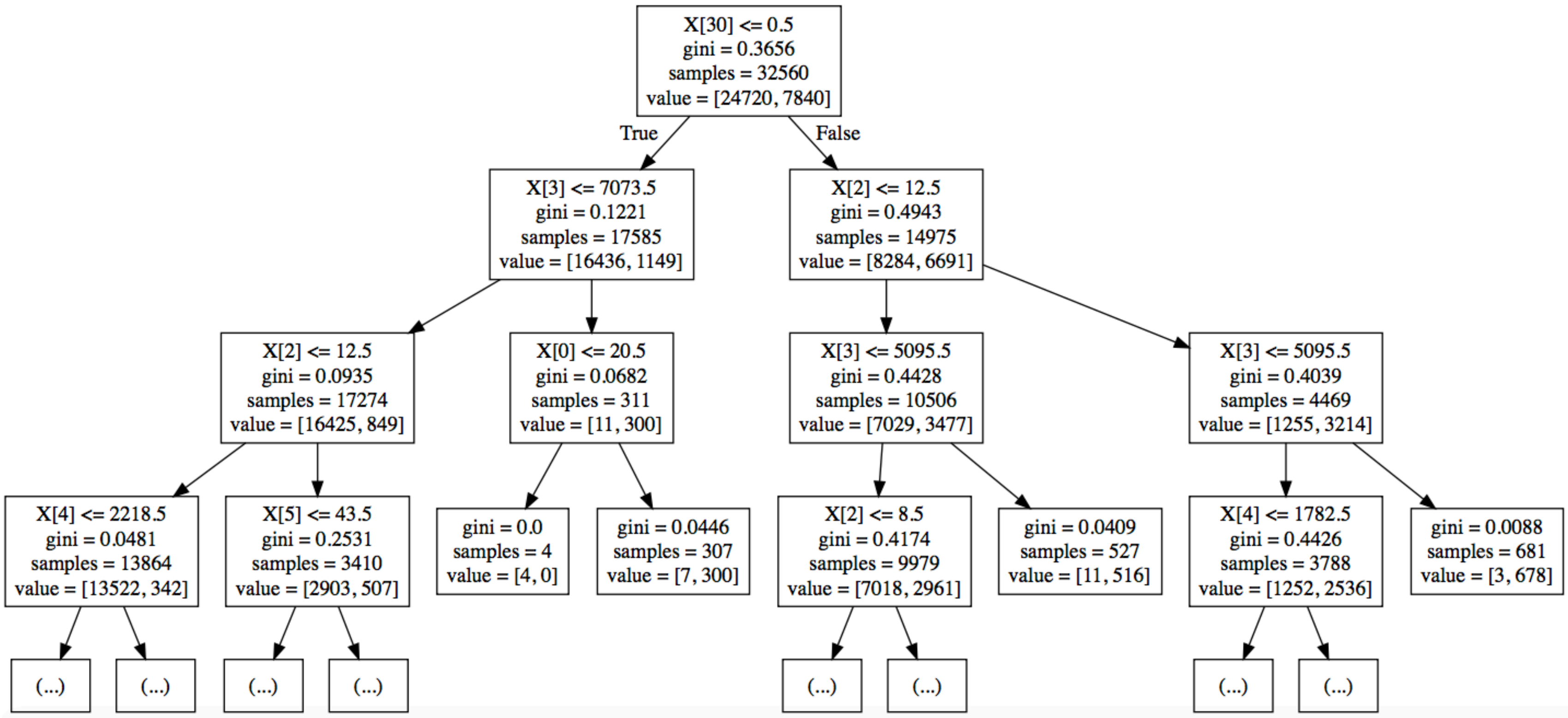
```
clf = tree.DecisionTreeClassifier(max_depth=12,max_leaf_nodes=49)
clf.fit(trainx, trainy)
testyHat = clf.predict(testx)
err=0
for i in range(len(testyHat)):
        if testyHat[i]!=testy[i]:
            err+=1
print "the accuracy is:",float ((len(testy)-float(err))/len(testy))
print "test", metrics.classification_report(testy, testyHat)
```

```
the accuracy is: 0.861249309011
test                precision    recall  f1-score   support

          0       0.89      0.94      0.91     12435
          1       0.76      0.61      0.67      3846

avg / total       0.86      0.86      0.86     16281
```

Plot the tree

In [133]:

```
from sklearn.tree import export_graphviz
tree.export_graphviz(clf, out_file='tree.dot',max_depth=3)
```


X[30] <= 0.5
gini = 0.3656
samples = 32560
value = [24720, 7840]
True
False
X[3] <= 7073.5
gini = 0.1221
samples = 17585
value = [16436, 1149]
X[2] <= 12.5
gini = 0.4943
samples = 14975
value = [8284, 6691]
X[2] <= 12.5
gini = 0.0935
samples = 17274
value = [16425, 849]
X[0] <= 20.5
gini = 0.0682
samples = 311
value = [11, 300]
X[3] <= 5095.5
gini = 0.4428
samples = 10506
value = [7029, 3477]
X[3] <= 5095.5
gini = 0.4039
samples = 4469
value = [1255, 3214]
X[4] <= 2218.5
gini = 0.0481
samples = 13864
value = [13522, 342]
X[5] <= 43.5
gini = 0.2531
samples = 3410
value = [2903, 507]
gini = 0.0
samples = 4
value = [4, 0]
gini = 0.0446
samples = 307
value = [7, 300]
X[2] <= 8.5
gini = 0.4174
samples = 9979
value = [7018, 2961]
gini = 0.0409
samples = 527
value = [11, 516]
X[4] <= 1782.5
gini = 0.4426
samples = 3788
value = [1252, 2536]
gini = 0.0088
samples = 681
value = [3, 678]
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)

HW3#3 Gradient Boosting

Import Data

In [1]:

```
import csv
import numpy
import scipy
from sklearn import linear_model
from math import sqrt
import matplotlib.pyplot as plt
training_list=[]
validate_list=[]
testdata_list=[]

tradata=csv.reader(open('/Users/wendy/Documents/2017 Fall/CS 534/homework1/BlogFeedb
for row in tradata:
    training_list.append(row)

valdata=csv.reader(open('/Users/wendy/Documents/2017 Fall/CS 534/homework1/BlogFeedb
for row in valdata:
    validate_list.append(row)

testdata=csv.reader(open('/Users/wendy/Documents/2017 Fall/CS 534/homework1/BlogFeed
for row in testdata:
    testdata_list.append(row)

training_list=numpy.array(training_list,dtype=float)
validate_list=numpy.array(validate_list,dtype=float)
testdate_list=numpy.array(validate_list,dtype=float)

ytrain=training_list[:,280]
xtrain=training_list[:, 0:279]
yval=validate_list[:,280]
xval=validate_list[:,0:279]
xtest=testdate_list[:,0:279]
ytest=testdate_list[:,280]
```

Gradient Boosting with mean square loss

In [73]:

```
def GB_squareLoss(v,m,xtrain,ytrain,xval,yval):
    from sklearn import tree
    clf = tree.DecisionTreeRegressor()
    clf = clf.fit(xtrain,ytrain)
    ytrain_hat=clf.predict(xtrain)
    loss=clf.predict(xval)-yval
    #loss_train=clf.predict(xtrain)-ytrain
    for i in range(m):
        r=ytrain_hat-ytrain
        h=tree.DecisionTreeRegressor()
        h = h.fit(xtrain,r)
        ytrain=ytrain_hat
        ytrain_hat=v*h.predict(xtrain)+ytrain_hat
        loss+=v*h.predict(xval)
        #loss_train+=v*h.predict(xtrain)
    return sum(0.5*loss**2)/len(yval)
```

Gradient Boosting with mean absolute loss

In [74]:

```
def GB_absLoss(v,m,xtrain,ytrain,xval,yval):
    from sklearn import tree
    clf = tree.DecisionTreeRegressor()
    clf = clf.fit(xtrain,ytrain)
    ytrain_hat=clf.predict(xtrain)
    loss=clf.predict(xval)-yval
    #loss_train=clf.predict(xtrain)-ytrain
    for i in range(m):
        r=[]
        for i in range(len(ytrain)):
            if ytrain_hat[i]!=ytrain[i]:
                r.append(1)
            else:
                r.append(0)
        h=tree.DecisionTreeRegressor()
        h = h.fit(xtrain,r)
        ytrain=ytrain_hat
        ytrain_hat=v*h.predict(xtrain)+ytrain_hat
        loss+=v*h.predict(xval)
        #loss_train+=v*h.predict(xtrain)
    return sum(0.5*loss**2)/len(yval)
```

For number of boosting iterations between 5, 10, 15, 25, plot the validation error as a function of the parameter v ∈ [0,1] for square loss.

In [95]:

```
b=[5,10,15,25]
a=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
loss_list=[]
m_list=[]
v_list=[]
final_list=[]
for j in range(len(b)):
    for i in range(len(a)):
        loss=GB_squareLoss(a[i],b[j],xtrain,ytrain,xval,yval)
        loss_list.append(loss)
        m_list.append(b[j])
        v_list.append(a[i])
        final_list.append((a[i],b[j],loss))
        print "iteration=",b[j],"shrinkage parameter=",a[i],"loss=",loss
    #plt.plot(a,loss_list)
    #plt.show()
```

```
iteration= 5 shrinkage parameter= 0.1 loss= 580.144855996
iteration= 5 shrinkage parameter= 0.2 loss= 567.586909633
iteration= 5 shrinkage parameter= 0.3 loss= 594.463417269
iteration= 5 shrinkage parameter= 0.4 loss= 597.615686554
iteration= 5 shrinkage parameter= 0.5 loss= 563.962425064
iteration= 5 shrinkage parameter= 0.6 loss= 586.20157722
iteration= 5 shrinkage parameter= 0.7 loss= 490.082786183
iteration= 5 shrinkage parameter= 0.8 loss= 523.969169283
iteration= 5 shrinkage parameter= 0.9 loss= 530.721345989
iteration= 10 shrinkage parameter= 0.1 loss= 550.576336355
iteration= 10 shrinkage parameter= 0.2 loss= 590.216992578
iteration= 10 shrinkage parameter= 0.3 loss= 553.161433057
iteration= 10 shrinkage parameter= 0.4 loss= 531.571050211
iteration= 10 shrinkage parameter= 0.5 loss= 553.90037111
iteration= 10 shrinkage parameter= 0.6 loss= 542.758901101
iteration= 10 shrinkage parameter= 0.7 loss= 528.496395946
iteration= 10 shrinkage parameter= 0.8 loss= 586.699159649
iteration= 10 shrinkage parameter= 0.9 loss= 590.445409649
iteration= 15 shrinkage parameter= 0.1 loss= 565.734049386
iteration= 15 shrinkage parameter= 0.2 loss= 578.293558894
iteration= 15 shrinkage parameter= 0.3 loss= 607.514487582
iteration= 15 shrinkage parameter= 0.4 loss= 560.120289752
iteration= 15 shrinkage parameter= 0.5 loss= 549.031369937
iteration= 15 shrinkage parameter= 0.6 loss= 593.372466457
iteration= 15 shrinkage parameter= 0.7 loss= 630.716220422
iteration= 15 shrinkage parameter= 0.8 loss= 598.597916072
iteration= 15 shrinkage parameter= 0.9 loss= 523.800064231
```

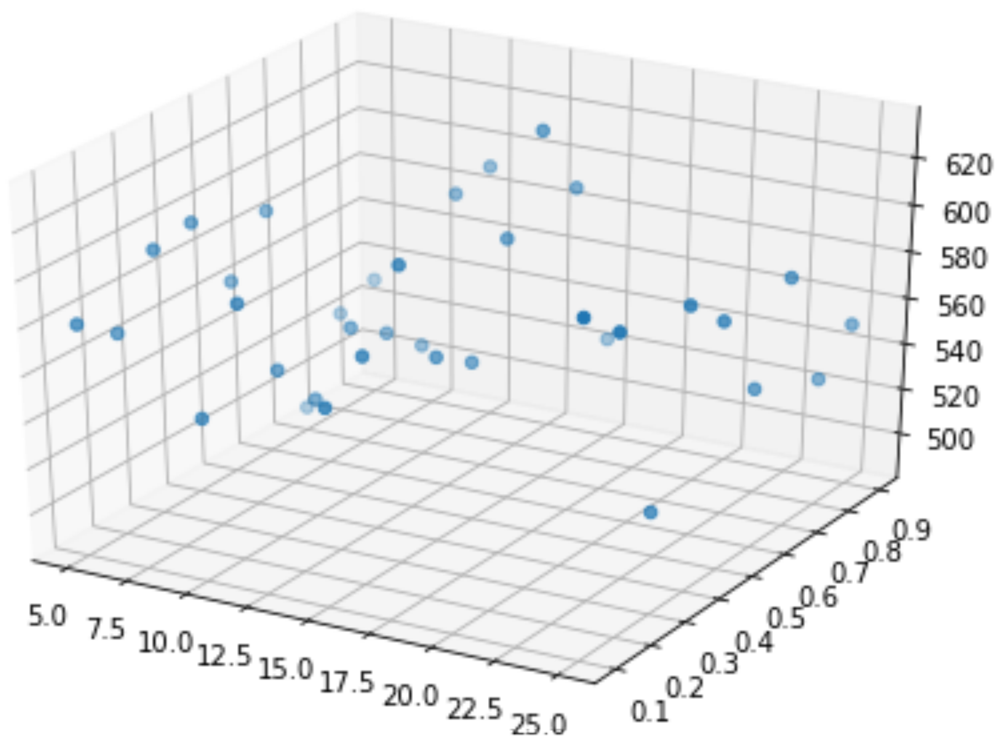
```
iteration= 25 shrinkage parameter= 0.1 loss= 623.161183271
iteration= 25 shrinkage parameter= 0.2 loss= 608.58082358
iteration= 25 shrinkage parameter= 0.3 loss= 524.319803026
iteration= 25 shrinkage parameter= 0.4 loss= 602.050391093
iteration= 25 shrinkage parameter= 0.5 loss= 586.816371681
iteration= 25 shrinkage parameter= 0.6 loss= 549.268983728
iteration= 25 shrinkage parameter= 0.7 loss= 587.886924065
iteration= 25 shrinkage parameter= 0.8 loss= 535.439746091
iteration= 25 shrinkage parameter= 0.9 loss= 550.612633616
```

Find out the optimal num of iteration and optimal shrinkage parameter

In [96]:

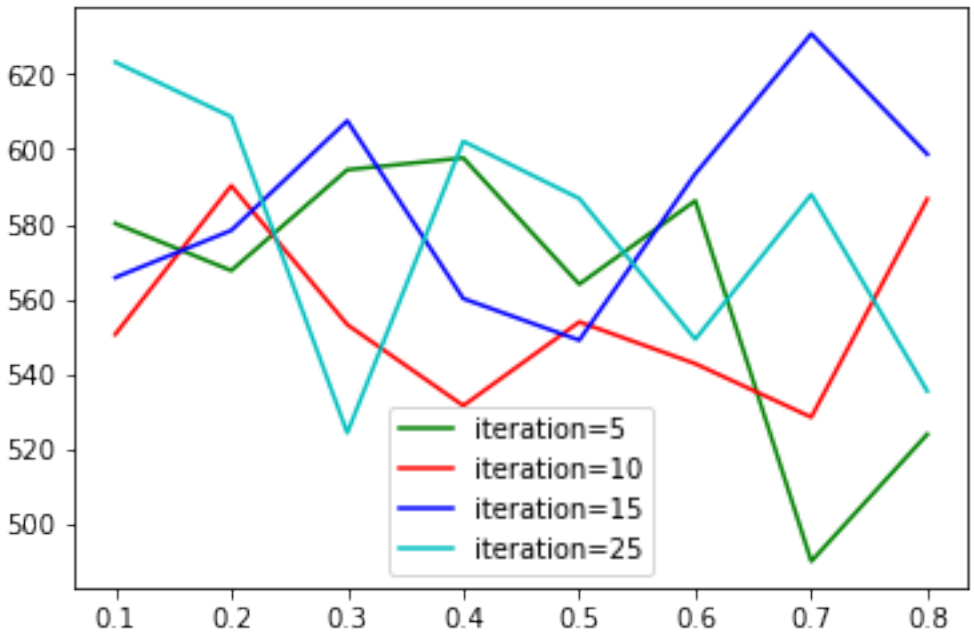
```
from operator import itemgetter
sorted_loss=sorted(final_list,key=itemgetter(2))
print "the optimal #of iteration is:",sorted_loss[0][1],"the optimal v is:",sorted_l
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig=plt.figure()
ax=Axes3D(fig)
ax.scatter(m_list,v_list,loss_list)
plt.show()
```

```
the optimal #of iteration is: 5 the optimal v is: 0.7 with the loss on
validation set: 490.082786183
```



In [97]:

```
ax=plt.gca()
ax.plot(v_list[0:8],loss_list[0:8],"g",label="iteration=5")
ax.plot(v_list[9:17],loss_list[9:17],"r",label="iteration=10")
ax.plot(v_list[18:26],loss_list[18:26],"b",label="iteration=15")
ax.plot(v_list[27:35],loss_list[27:35],"c",label="iteration=25")
plt.legend()
plt.show()
```



For number of boosting iterations between 5, 10, 15, 25, plot the validation error as a function of the parameter v ∈ [0,1] for absolute loss.

In [78]:

```
b=[5,10,15,25]
a=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
loss_list=[]
m_list=[]
v_list=[]
final_list=[]
for j in range(len(b)):
    for i in range(len(a)):
        loss=GB_absLoss(a[i],b[j],xtrain,ytrain,xval,yval)
        loss_list.append(loss)
        m_list.append(b[j])
        v_list.append(a[i])
        final_list.append((a[i],b[j],loss))
        print "iteration=",b[j],"shrinkage parameter=",a[i],"loss=",loss
    #plt.plot(a,loss_list)
    #plt.show()
```

```
iteration= 5 shrinkage parameter= 0.1 loss= 597.500071367
iteration= 5 shrinkage parameter= 0.2 loss= 517.212567799
```

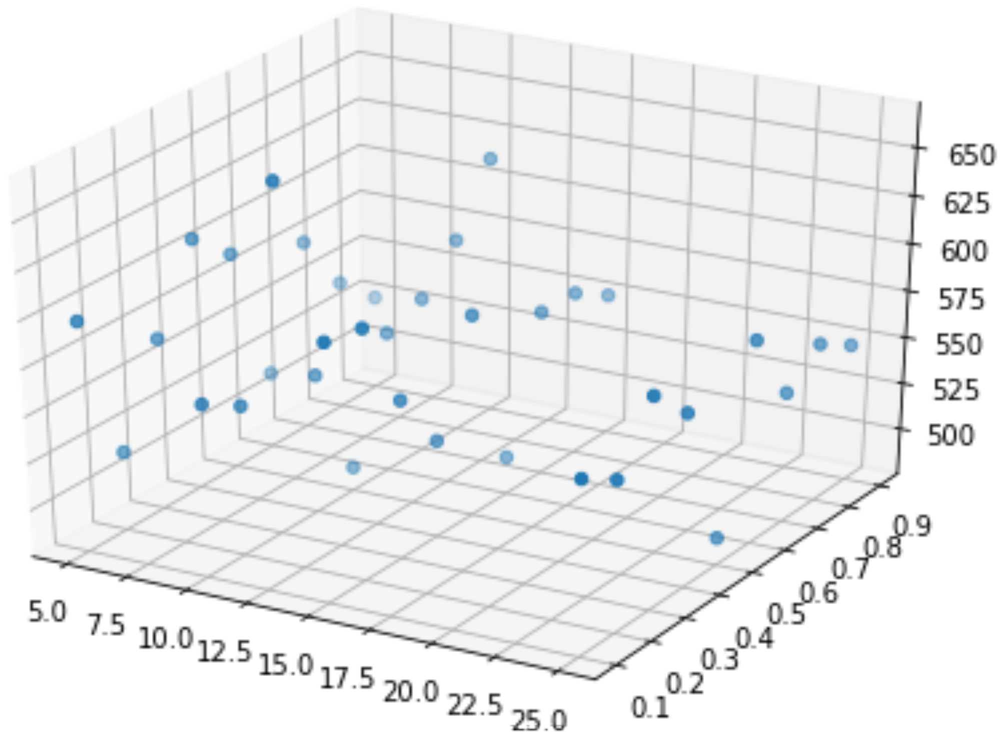
```
iteration= 5 shrinkage parameter= 0.3 loss= 566.787421298
iteration= 5 shrinkage parameter= 0.4 loss= 608.758676848
iteration= 5 shrinkage parameter= 0.5 loss= 590.609834428
iteration= 5 shrinkage parameter= 0.6 loss= 515.337353697
iteration= 5 shrinkage parameter= 0.7 loss= 576.240517057
iteration= 5 shrinkage parameter= 0.8 loss= 543.795755424
iteration= 5 shrinkage parameter= 0.9 loss= 525.110690836
iteration= 10 shrinkage parameter= 0.1 loss= 567.112733728
iteration= 10 shrinkage parameter= 0.2 loss= 555.032511815
iteration= 10 shrinkage parameter= 0.3 loss= 660.225750944
iteration= 10 shrinkage parameter= 0.4 loss= 548.999327719
iteration= 10 shrinkage parameter= 0.5 loss= 487.615766843
iteration= 10 shrinkage parameter= 0.6 loss= 549.928133029
iteration= 10 shrinkage parameter= 0.7 loss= 557.734013703
iteration= 10 shrinkage parameter= 0.8 loss= 579.0470668
iteration= 10 shrinkage parameter= 0.9 loss= 612.6331002
iteration= 15 shrinkage parameter= 0.1 loss= 611.798922352
iteration= 15 shrinkage parameter= 0.2 loss= 608.007743363
iteration= 15 shrinkage parameter= 0.3 loss= 559.73668998
iteration= 15 shrinkage parameter= 0.4 loss= 526.886077806
iteration= 15 shrinkage parameter= 0.5 loss= 582.628996574
iteration= 15 shrinkage parameter= 0.6 loss= 494.993683985
iteration= 15 shrinkage parameter= 0.7 loss= 562.86429489
iteration= 15 shrinkage parameter= 0.8 loss= 562.509745615
iteration= 15 shrinkage parameter= 0.9 loss= 550.752114259
iteration= 25 shrinkage parameter= 0.1 loss= 568.370075649
iteration= 25 shrinkage parameter= 0.2 loss= 556.160255495
iteration= 25 shrinkage parameter= 0.3 loss= 587.977412218
iteration= 25 shrinkage parameter= 0.4 loss= 567.814337711
iteration= 25 shrinkage parameter= 0.5 loss= 489.695586323
iteration= 25 shrinkage parameter= 0.6 loss= 583.562232372
iteration= 25 shrinkage parameter= 0.7 loss= 544.836220739
iteration= 25 shrinkage parameter= 0.8 loss= 559.896354712
iteration= 25 shrinkage parameter= 0.9 loss= 548.429025121
```

Find out the optimal num of iteration and optimal shrinkage parameter

In [86]:

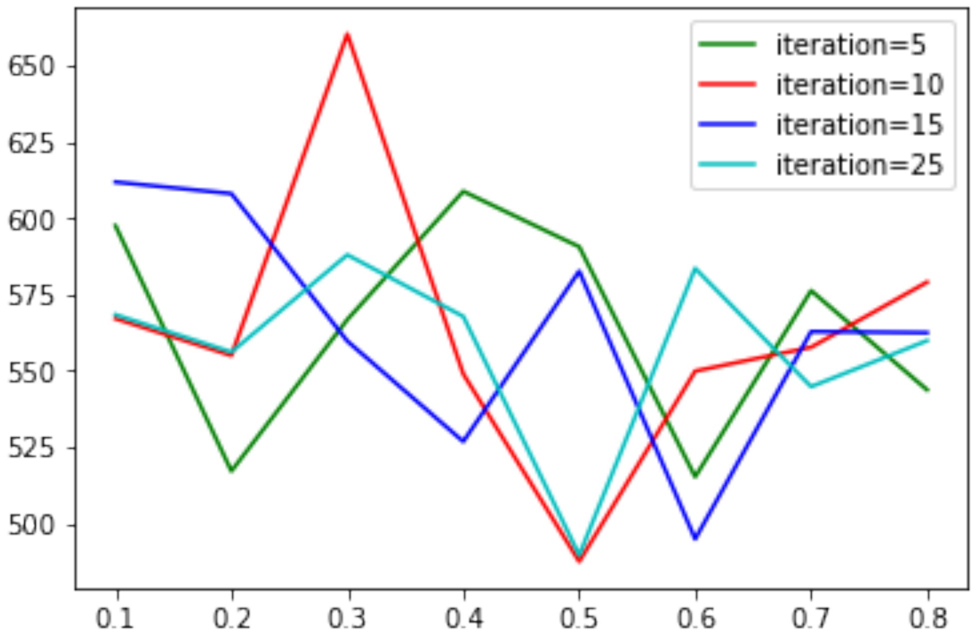
```
from operator import itemgetter
sorted_loss=sorted(final_list,key=itemgetter(2))
print "the optimal #of iteration is:",sorted_loss[0][1],"the optimal v is:",sorted_l
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig=plt.figure()
ax=Axes3D(fig)
ax.scatter(m_list,v_list,loss_list)
plt.show()
```

```
the optimal #of iteration is: 10 the optimal v is: 0.5 with the loss o
n validation set: 487.615766843
```



In [90]:

```
ax=plt.gca()
ax.plot(v_list[0:8],loss_list[0:8],"g",label="iteration=5")
ax.plot(v_list[9:17],loss_list[9:17],"r",label="iteration=10")
ax.plot(v_list[18:26],loss_list[18:26],"b",label="iteration=15")
ax.plot(v_list[27:35],loss_list[27:35],"c",label="iteration=25")
plt.legend()
plt.show()
```



from the plots in terms of optimal parameters and the use of the shrinkage parameter, the increase of num of iteration does not affect the optimal shrinkage parameter

Calculate the loss on test data with the optimal parameters v and boosting iterations for each of the loss function

In [98]:

```
test_loss1=GB_squareLoss(0.7,5,xtrain,ytrain,xtest,ytest)
print "loss on test data with square loss is :",test_loss1
test_loss2=GB_absLoss(0.5,10,xtrain,ytrain,xtest,ytest)
print "loss on test data with absolute loss is :",test_loss2
```

```
loss on test data with square loss is : 591.876784185
loss on test data with absolute loss is : 597.463644019
```

```
The result on test data is worse than the result on training data and validation
data. this is because gradient boosting may bring about the problem of
overfitting, so the model cannot fit well on testing data
```

In [ ]:

```

```