1. a) $x=[x_1, x_2] \quad z=[z_1, z_2]$

$$k_\beta(x\cdot z) = 1+\beta^2(x_1z_1+x_2z_2)^2+2\beta(x_1z_1+x_2z_2)-1$$

$$=\beta^2x_1^2z_1^2+\beta^2x_2^2z_2^2+2\beta^2(x_1z_1x_2z_2)+2\beta x_1z_1+2\beta x_2z_2$$

$$\therefore \phi(x)=(\beta x_1^2, \beta x_2^2, \sqrt{2}\beta x_1x_2, \sqrt{2\beta}x_1, \sqrt{2\beta}x_2)$$

$$\phi(z)=(\beta z_1^2, \beta z_2^2, \sqrt{2}\beta z_1z_2, \sqrt{2\beta}z_1, \sqrt{2\beta}z_2)$$

$$\phi(x)\,\phi z = \beta^2x_1z_1+\beta^2x_2z_2+2\beta^2x_1x_2z_1z_2+2\beta x_1z_1+2\beta x_2z_2 = k_\beta(x,z)$$

$\therefore k_\beta(x\cdot z)=(1+\beta x\cdot z)^2-1$ is a kernel

b) $\phi(x)=(x_1, x_2, \|x_1\|_2) \qquad \phi(z)=(z_1, z_2, \|z\|_2)$

$$\phi(x)\,\phi(z) = x_1z_1+x_2z_2+\|x_1\|_2\|z\|_2$$

$\therefore k(x,z)=(x^Tz+\|x_1\|_2\|x_2\|_2)$ is a kernel

$\therefore k(x\cdot z)=\left(\frac{x^Tz}{\|x_1\|_2\|x_2\|_2}+1\right)$ is a kernel (scaling)

$\therefore k(x\cdot z)=\left(1+\frac{x^Tz}{\|x_1\|_2\|x_2\|_2}\right)^3$ is a kernel (product)

HW4#2 Random Forest

Data processing and split the data

In [27]:

```
import csv
import numpy
import scipy
from copy import deepcopy
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import random
import math
from operator import itemgetter
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D



data_list=[]
y=[]
data=csv.reader(open('/Users/wendy/Documents/2017 Fall/CS 534/HW4/allhyper.data'))
for row in data:
    data_list.append(row)
features=deepcopy(data_list[0])
data_list.pop(0)

####convert string to int
for i in range(len(data_list)):
    if data_list[i][29]=='negative.':
        data_list[i][29]=0
    else:
        data_list[i][29]=1
############
for i in range(len(data_list)):
    if data_list[i][1]=='F':
        data_list[i][1]=1
    elif data_list[i][1]=='M':
        data_list[i][1]=2
    else:
        data_list[i][1]=0
index_list=(2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,18,20,22,24,26)
for i in index_list:
    for j in range(len(data_list)):

        if data_list[j][i]=='f':
            data_list[j][i]=0
        else:
```

```
            else:
                data_list[j][i]=1
############
class_mean={}
def calculate_mean(n):
    sum=0
    index=0
    for i in range(len(data_list)):
        if data_list[i][n]!='NA':
            sum+=float(data_list[i][n])
            index+=1
    return sum/index
class_mean[0]=calculate_mean(0)
class_mean[17]=calculate_mean(17)
class_mean[19]=calculate_mean(19)
class_mean[21]=calculate_mean(21)
class_mean[23]=calculate_mean(23)
class_mean[25]=calculate_mean(25)
print class_mean
feature_list=(0,17,19,21,23,25)
for i in feature_list:
    for j in range(len(data_list)):
        if data_list[j][i]=='NA':
            data_list[j][i]=class_mean[i]
###############
whole_data=numpy.array(data_list)
trainx=whole_data[0:int(0.7*len(whole_data)),0:26]
trainy=whole_data[0:int(0.7*len(whole_data)),29]
testx=whole_data[int(0.7*len(whole_data)):len(whole_data),0:26]
testy=whole_data[int(0.7*len(whole_data)):len(whole_data),29]
print len(trainx)
```

```
{0: 51.8442300821722, 17: 4.672150238473764, 19: 2.0249661399548584, 2
1: 109.07240061162081, 23: 0.9979121054734302, 25: 110.78798403193613}
1959
```

Function of random forest

In [33]:

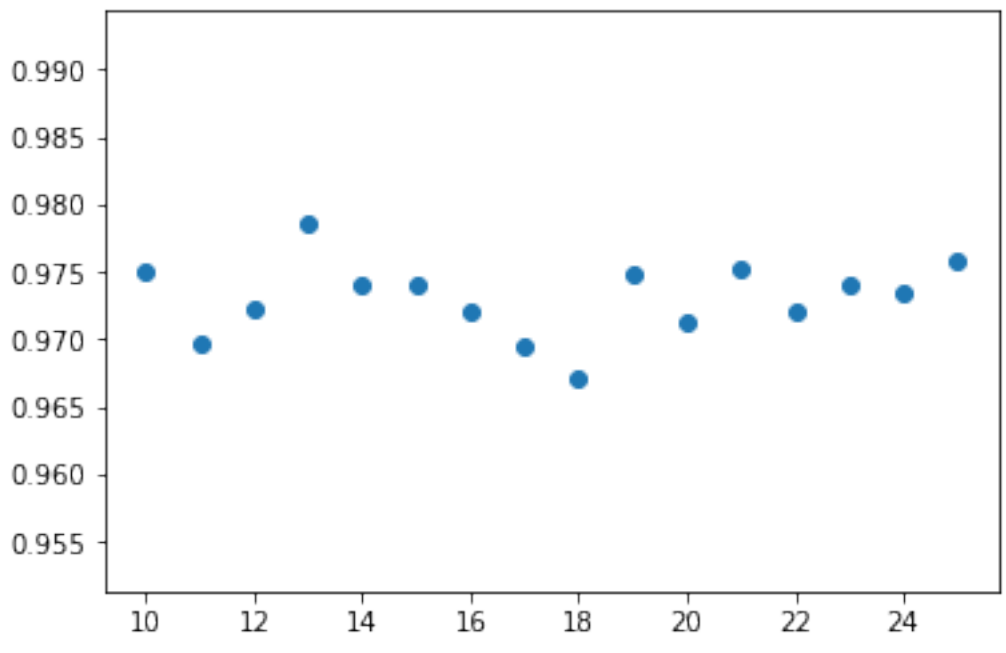
```
def RF(n,d,l,criterion):
    class_vote_0=[0]*1000000
    class_vote_1=[0]*1000000
    final_prediction=[]

    for i in range(n):

        clf=DecisionTreeClassifier(criterion=criterion,max_depth=d,max_leaf_nodes=l)
        #####random bootstrap sample######
        s=[]
        v=[]
        index=range(0,1959)
        for x in range(int(0.632*1958)):
            s.append(random.randint(0,1958))

        for n in range(len(index)):
            #print train[n]
            if index[n] not in s:
                v.append(index[n])

        bootstrapx=trainx[s,:]
        bootstrapy=trainy[s]
        valx=trainx[v,:]
        valy=trainy[v]
        #########random choose features############
        f=[]
        for x in range(int(math.sqrt(len(trainx[0])))):
            f.append(random.randint(0,25))
        ########################
        clf.fit(bootstrapx[:,f],bootstrapy)
        valyHat=clf.predict(valx[:,f])
        ####vote########
        for a in range(len(valyHat)):
            if valyHat[a]=='0':
                class_vote_0[a]+=1
            else:
                class_vote_1[a]+=1
    ###calculate accuracy########
    for a in range(len(valyHat)):
        if class_vote_0[a]>class_vote_1[a]:
            final_prediction.append(0)
        else:
            final_prediction.append(1)

    err=0
    for i in range(len(final_prediction)):
        if final_prediction[i]!=int(valy[i]):
            err+=1
    #####return OOB accuracy and prediction#######
    return 1-float(err)/float(len(valx)),valyHat
```

find the optimal number of nest for gini

In [44]:

```
acc=-float("inf")
accuracy_list=[]
index_list=[]
index=0
for i in range(10,26):
    a,prediction=RF(i,15,25,'gini')
    print a,i
    index_list.append(i)
    accuracy_list.append(a)
    if a>=acc:
        acc=a
        index=i

print "the optimal number of nest for gini is:",index,"with accuracy of:",acc
plt.scatter(index_list,accuracy_list)
plt.show()
```

```
0.975119617225 10
0.969639468691 11
0.972195589645 12
0.978620019436 13
0.974088291747 14
0.973963355834 15
0.97216890595 16
0.969465648855 17
0.967086156825 18
0.974830590513 19
0.971346704871 20
0.975238095238 21
0.972115384615 22
0.974038461538 23
0.973384030418 24
0.975751697381 25
the optimal number of nest for gini is: 13 with accuracy of: 0.9786200
19436
```



find the optimal number of depth and leaves with gini and optimal number of nests

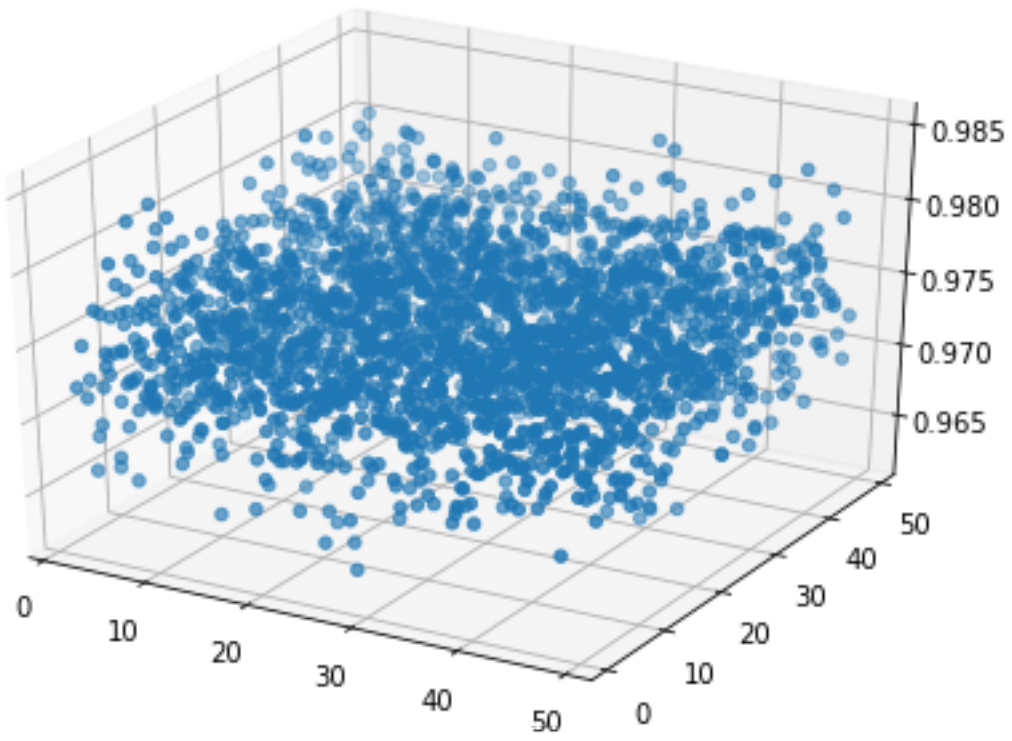
In [45]:

```
err_list=[]
depth_list=[]
l_list=[]
final_list=[]

for d in range(1,50):
    for l in range(2,50):
        err,prediction=RF(13,d,l,'gini')
        err_list.append(err)
        depth_list.append(d)
        l_list.append(l)
        final_list.append((err,d,l))
```

In [46]:

```
#from operator import itemgetter
sorted_err=sorted(final_list,key=itemgetter(0),reverse=True)
print "optimal depth is :",sorted_err[0][1],"optimal number of leaves is :",sorted_e

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

```
optimal depth is : 25 optimal number of leaves is : 10 with accuracy :
0.984761904762
```

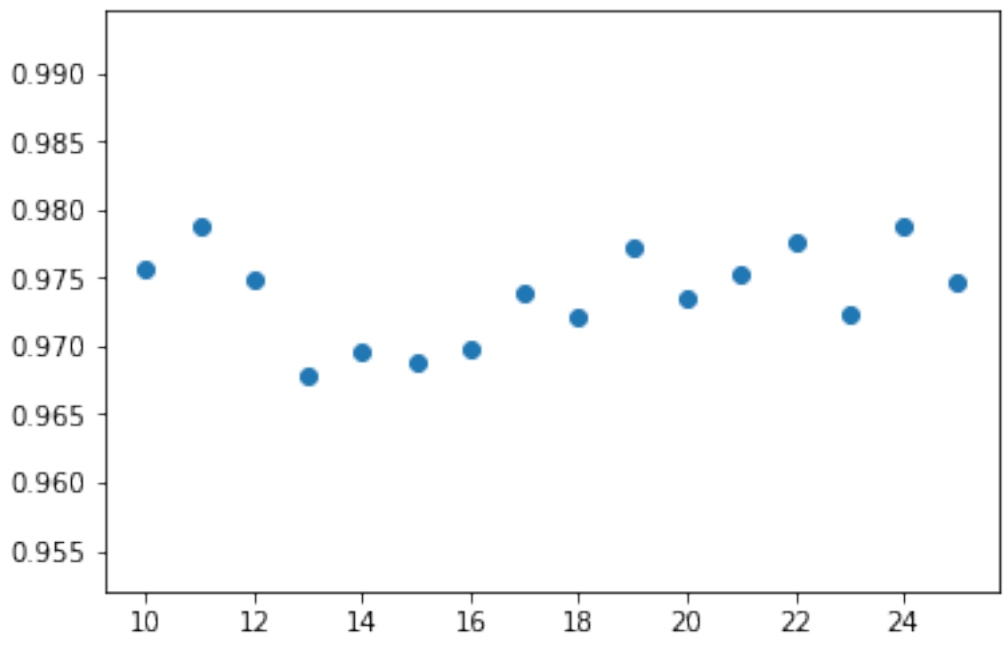


find the optimal number of nest for entropy

In [47]:

```
acc=-float("inf")
accuracy_list=[]
index_list=[]
index=0
for i in range(10,26):
    a,prediction=RF(i,15,25,'entropy')
    print a,i
    index_list.append(i)
    accuracy_list.append(a)
    if a>=acc:
        acc=a
        index=i

print "the optimal number of nest for entropy is:",index,"with accuracy of:",acc
plt.scatter(index_list,accuracy_list)
plt.show()
```

```
0.975657254138 10
0.978825794033 11
0.97480620155 12
0.967741935484 13
0.969668246445 14
0.96875 15
0.969696969697 16
0.973811833172 17
0.972088546679 18
0.977251184834 19
0.973581213307 20
0.975190839695 21
0.977648202138 22
0.972275334608 23
0.978764478764 24
0.97476635514 25
the optimal number of nest for entropy is: 11 with accuracy of: 0.9788
25794033
```



find the optimal number of depth and leaves with entropy and optimal number of nests

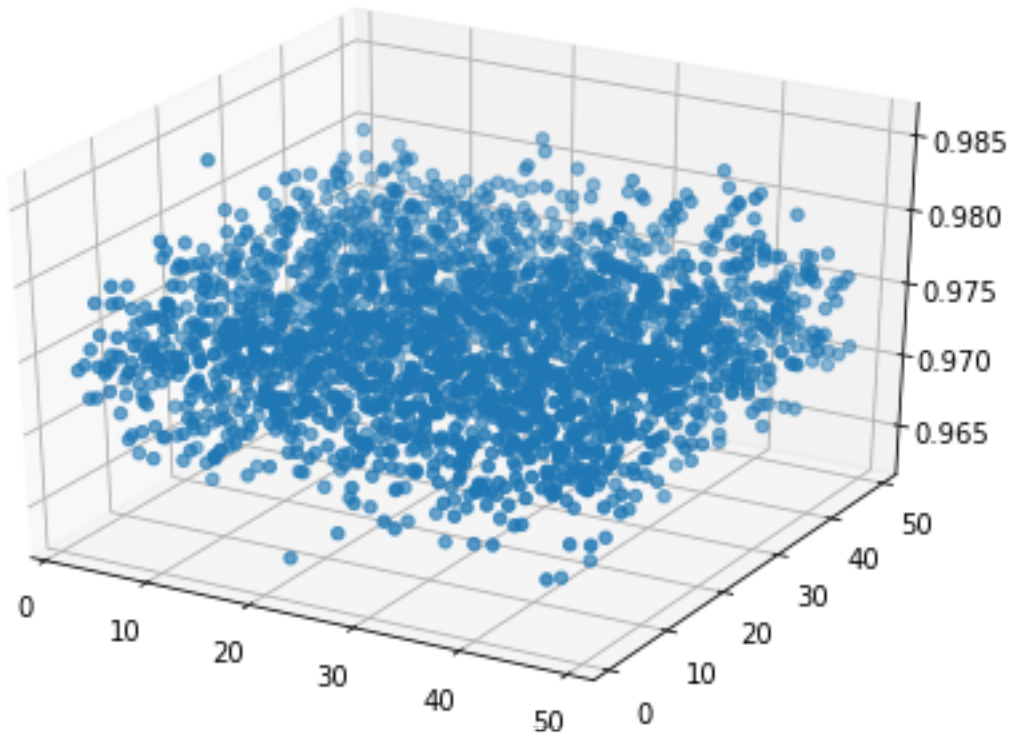
In [48]:

```
err_list=[]
depth_list=[]
l_list=[]
final_list=[]

for d in range(1,50):
    for l in range(2,50):
        err,prediction=RF(11,d,l,'entropy')
        err_list.append(err)
        depth_list.append(d)
        l_list.append(l)
        final_list.append((err,d,l))
```

In [49]:

```
sorted_err=sorted(final_list,key=itemgetter(0),reverse=True)
print "optimal depth is :",sorted_err[0][1],"optimal number of leaves is :",sorted_e
fig=plt.figure()
ax=Axes3D(fig)
ax.scatter(l_list,depth_list,err_list)
plt.show()
```

```
optimal depth is : 24 optimal number of leaves is : 45 with accuracy :
0.985493230174
```



Use algorithm:entropy, #of nests:11,optimal depth: 24,optimal #of leaves: 45 to identify the most importand features

In [51]:

```
clf=DecisionTreeClassifier('entropy',max_depth=24,max_leaf_nodes=45)
def feature_importance():
    importance_list=[]
    for i in range(len(trainx[0])):
        class_vote_0=[0]*1000000
        class_vote_1=[0]*1000000
        final_prediction=[]

        for iteration in range(11):
            #####random bootstrap sample######
            s=[]
            v=[]
            index=range(0,1959)
            for x in range(int(0.632*1958)):
                s.append(random.randint(0,1958))

            for n in range(len(index)):
                #print train[n]
                if index[n] not in s:
                    v.append(index[n])

            bootstrapx=numpy.delete(trainx[s,:],i,1)
            bootstrapy=trainy[s]
            valx=numpy.delete(trainx[v,:],i,1)
            valy=trainy[v]

            clf.fit(bootstrapx,bootstrapy)
            valyHat=clf.predict(valx)

            for a in range(len(valyHat)):
                if valyHat[a]=='0':
                    class_vote_0[a]+=1
                else:
                    class_vote_1[a]+=1
        for a in range(len(valyHat)):
            if class_vote_0[a]>class_vote_1[a]:
                final_prediction.append(0)
            else:
                final_prediction.append(1)
        #print final_prediction
        err=0
        #print len(final_prediction),len(valy)
        for p in range(len(final_prediction)):
            if final_prediction[p]!=int(valy[p]):
                err+=1

        importance_list.append((i,1-float(err)/float(len(valx))))
    return importance_list
```

In [52]:

```
true_accuracy=0.985493230174
importance=[]
feature_importance_list=feature_importance()
for i in feature_importance_list:
    importance.append((i,abs(i[1]-true_accuracy)))

sorted_importance=sorted(importance,key=itemgetter(1),reverse=True)
print sorted_importance
```

```
[((0, 0.9667616334283001), 0.018731596745699908), ((24, 0.969873663751
2148), 0.015619566422785 28), ((18, 0.9702495201535508), 0.015243710020
44922), ((10, 0.9703065134099617), 0.015186716764038377), ((14, 0.9706
744868035191), 0.014818743370480947), ((7, 0.9707271010387157), 0.0147
661291352843), ((9, 0.9715909090909091), 0.013902321083090974), ((17,
0.9719806763285024), 0.013512553845497655), ((5, 0.9732057416267943),
0.012287488547205738), ((1, 0.9732824427480916), 0.01221078742590842),
((6, 0.9740384615384615), 0.011454768635538515), ((3, 0.97428571428571
43), 0.011207515888285724), ((13, 0.974609375), 0.010883855174000034),
((19, 0.9760536398467433), 0.009439590327256 72), ((21, 0.9760765550239
234), 0.009416675150076603), ((2, 0.9767225325884544), 0.0087706975855
45606), ((11, 0.9769230769230769), 0.0085701532509 2314), ((4, 0.976945
2449567724), 0.008547985217227 66), ((23, 0.9770992366412213), 0.008393
993532778699), ((16, 0.9771210676835081), 0.008372162490491886), ((22,
0.9779058597502401), 0.007587370423759898), ((12, 0.9781368821292775),
0.007356348044722516), ((8, 0.9781990521327014), 0.007294178041298616)
, ((15, 0.9786407766990292), 0.006852453474970 8765), ((25, 0.980769230
7692307), 0.004723999404769308), ((20, 0.9816602316602316), 0.00383299
8513768393)]
```

the most important fove features are: 1st,24th,18th,10th and 14th, which will have the largest infulence on the accuarcy of prediction

In [ ]:

HW4#3 Percepton

In [1]:

```
import csv
import numpy
import scipy
from copy import deepcopy
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import random
import math
```

Import Data and split train and test

In [2]:

```
data_list=[]
data=pd.read_csv('/Users/wendy/Documents/2017 Fall/CS 534/HW4/spamAssassin.data',hea
for i in range(int(0.7*len(data))):
    data_list.append(data[0][i])
test_data=[]
for i in range(int(0.7*len(data)),len(data)):
    test_data.append(data[0][i])
```

Calculate word frequency

In [3]:

```
train=[]
for i in range(len(data_list)):
    train.append(data_list[i].split(' '))
test=[]
for i in range(len(test_data)):
    test.append(test_data[i].split(' '))
class_vote={}
for j in range(len(train)):
    for i in range(1,len(train[j])):
        if train[j][i] in class_vote:
            class_vote[train[j][i]]+=1
        else:
            class_vote[train[j][i]]=1
class_vote_key=list(class_vote.keys())
for i in class_vote_key:
    if class_vote[i]<30:
        del class_vote[i]
```

Data transformation

In [5]:

```
train_list=[]
for sample in train:
    sample_list=[]
    if sample[0]=='0':
        sample_list.append(1)
    else:
        sample_list.append(-1)
    for i in class_vote:
        if i in sample:
            sample_list.append(1)
        else:
            sample_list.append(0)
    train_list.append(sample_list)
test_list=[]
for sample in test:
    sample_list=[]
    if sample[0]=='0':
        sample_list.append(1)
    else:
        sample_list.append(-1)
    for i in class_vote:
        if i in sample:
            sample_list.append(1)
        else:
            sample_list.append(0)
    test_list.append(sample_list)
```

In [6]:

```
train_array=numpy.array(train_list)
test_array=numpy.array(test_list)
trainy=train_array[:,0]
trainx=train_array[:,1:3380]
testy=test_array[:,0]
testx=test_array[:,1:3380]
```

perceptron algorithm

In [50]:

```
def precrpton(echo,learning):
    w=numpy.zeros(shape=(1,3379))
    err_list=[]
    for i in range(echo):
        w_list=[]
        for sample in range(len(trainx)):
            a=w*trainx[sample]
            s=0
            for x in a[0]:
                s+=x
            if s<0:
                prediction=-1
            else:
                prediction=1

            if prediction!=trainy[sample]:

                w=w+learning*trainy[sample]*trainx[sample]

    train_err=0
    test_err=0
    for sample in range(len(testx)):
        a=w*testx[sample]
        s=0
        for x in a[0]:
            s+=x
        if s<0:
            prediction=-1
        else:
            prediction=1
        if prediction!=testy[sample]:
            test_err+=1
    for sample in range(len(trainx)):
        a=w*trainx[sample]
        s=0
        for x in a[0]:
            s+=x
        if s<0:
            prediction=-1
        else:
            prediction=1
        if prediction!=trainy[sample]:
            train_err+=1
    return train_err,test_err,w
```

perceptron algorithm with average coefficients

In [51]:

```
def average_precrpton(echo,learning):
    w=numpy.zeros(shape=(1,3379))
    err_list=[]
    for i in range(echo):
        w_list=[]
        for sample in range(len(trainx)):
            a=w*trainx[sample]
            s=0
            for x in a[0]:
                s+=x
            if s<0:
                prediction=-1
            else:
                prediction=1

            if prediction!=trainy[sample]:

                w=w+learning*trainy[sample]*trainx[sample]
            w_list.append(w)
            w=sum(w_list)/len(trainx)

    train_err=0
    test_err=0
    for sample in range(len(testx)):
        a=w*testx[sample]
        s=0
        for x in a[0]:
             s+=x
        if s<0:
             prediction=-1
        else:
             prediction=1
        if prediction!=testy[sample]:
            test_err+=1
    for sample in range(len(trainx)):
        a=w*trainx[sample]
        s=0
        for x in a[0]:
             s+=x
        if s<0:
             prediction=-1
        else:
             prediction=1
        if prediction!=trainy[sample]:
            train_err+=1
    return train_err,test_err,w
```

find the optimal paramater

In [52]:

```
train_err_list1=[]
train_err_list2=[]
test_err_list1=[]
test_err_list2=[]
for i in range(20):
    trainerr_1,testerr_1,w1=precrpton(i,0.05)
    trainerr_2,testerr_2,w2=average_precrpton(i,0.05)
    train_err_list1.append(trainerr_1)
    train_err_list2.append(trainerr_2)
    test_err_list1.append(testerr_1)
    test_err_list2.append(testerr_2)
```

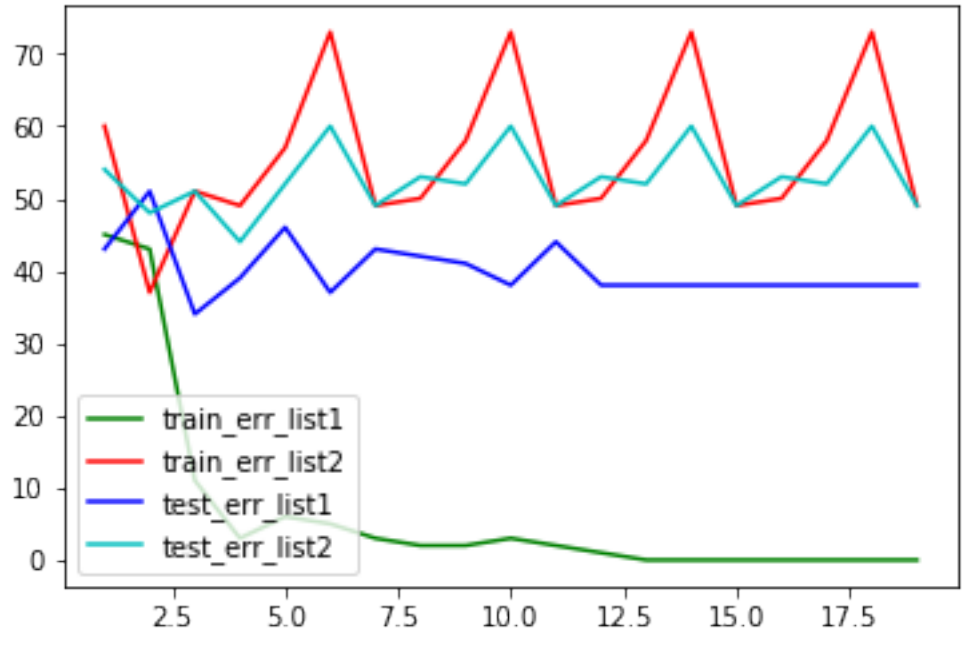
In [58]:

```
index=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
print train_err_list1,train_err_list2,test_err_list1,test_err_list2
```

```
[1331, 45, 43, 11, 3, 6, 5, 3, 2, 2, 3, 2, 1, 0, 0, 0, 0, 0, 0, 0] [13
31, 60, 37, 51, 49, 57, 73, 49, 50, 58, 73, 49, 50, 58, 73, 49, 50, 58
, 73, 49] [546, 43, 51, 34, 39, 46, 37, 43, 42, 41, 38, 44, 38, 38, 38
, 38, 38, 38, 38, 38] [546, 54, 48, 51, 44, 52, 60, 49, 53, 52, 60, 49
, 53, 52, 60, 49, 53, 52, 60, 49]
```

In [62]:

```
from matplotlib import pyplot as plt
ax=plt.gca()
ax.plot(index,train_err_list1[1:20],"g",label="train_err_list1")
ax.plot(index,train_err_list2[1:20],"r",label="train_err_list2")
ax.plot(index,test_err_list1[1:20],"b",label="test_err_list1")
ax.plot(index,test_err_list2[1:20],"c",label="test_err_list2")
plt.legend()
plt.show()
```



Based on the graph, we can see that the first algrithm which updates the coefficients with each misclassified sample has better peroormance. With enough amount of training data, the err_rate will goes to 0. The best max number of epoch is four with the lowest err_rate on test data.

In [98]:

```
train_err,test_err,coef=precrpton(4,0.05)

coef=numpy.array(coef)
import heapq
positive_words=heapq.nlargest(5, xrange(len(coef[0])), coef.take)
print "the 5 most negative words are:"
for i in positive_words:
    print class_vote.items()[i]

negative_words=heapq.nsmallest(5, xrange(len(coef[0])), coef.take)
print "the 5 most negative words are:"
for i in negative_words:
    print class_vote.items()[i]
```

```
the 5 most negative words are:
('wrote', 1188)
('which', 2153)
('copyright', 375)
('date', 966)
('ll', 917)
the 5 most negative words are:
('sight', 158)
('remov', 1510)
('click', 1846)
('deathtospamdeathtospamdeathtospam', 116)
('market', 1141)
```

In [ ]:

```

```

HW4 SVM

Loading data and data transformation

In [1]:

```
import csv
import numpy
import scipy
from copy import deepcopy
import pandas as pd
import random
import math
from operator import itemgetter
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

from sklearn import svm
from sklearn.metrics import f1_score
from sklearn.metrics import fbeta_score
```

```
data_list=[]
y=[]
data=csv.reader(open('/Users/wendy/Documents/2017 Fall/CS 534/HW4/allhyper.data'))
for row in data:
    data_list.append(row)
features=deepcopy(data_list[0])
data_list.pop(0)

####convert string to int
for i in range(len(data_list)):
    if data_list[i][29]=='negative.':
        data_list[i][29]=0
    else:
        data_list[i][29]=1
############
for i in range(len(data_list)):
    if data_list[i][1]=='F':
        data_list[i][1]=1
    elif data_list[i][1]=='M':
        data_list[i][1]=2
    else:
        data_list[i][1]=0
index_list=(2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,18,20,22,24,26)
for i in index_list:
    for j in range(len(data_list)):

        if data_list[j][i]=='f':
```

```
            data_list[j][i]=0
        else:
            data_list[j][i]=1
############
class_mean={}
def calculate_mean(n):
    sum=0
    index=0
    for i in range(len(data_list)):
        if data_list[i][n]!='NA':
            sum+=float(data_list[i][n])
            index+=1
    return sum/index
class_mean[0]=calculate_mean(0)
class_mean[17]=calculate_mean(17)
class_mean[19]=calculate_mean(19)
class_mean[21]=calculate_mean(21)
class_mean[23]=calculate_mean(23)
class_mean[25]=calculate_mean(25)
print class_mean
feature_list=(0,17,19,21,23,25)
for i in feature_list:
    for j in range(len(data_list)):
        if data_list[j][i]=='NA':
            data_list[j][i]=class_mean[i]
################
whole_data=numpy.array(data_list)
trainx=whole_data[0:int(0.7*len(whole_data)),0:26]
trainy=whole_data[0:int(0.7*len(whole_data)),29]
testx=whole_data[int(0.7*len(whole_data)):len(whole_data),0:26]
testy=whole_data[int(0.7*len(whole_data)):len(whole_data),29]
```

```
{0: 51.8442300821722, 17: 4.672150238473764, 19: 2.0249661399548584, 2
1: 109.07240061162081, 23: 0.9979121054734302, 25: 110.78798403193613}
```

SVM with linear kernel

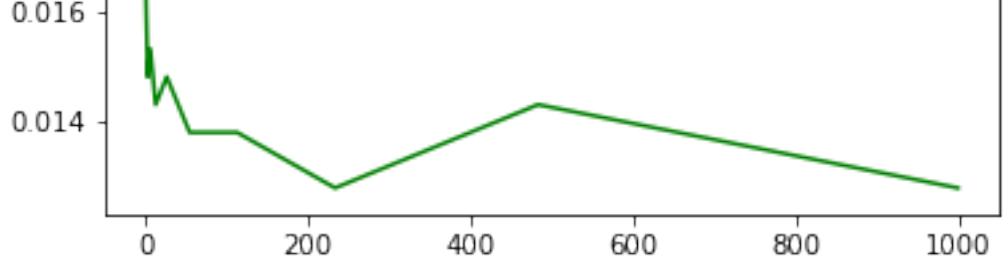
In [15]:

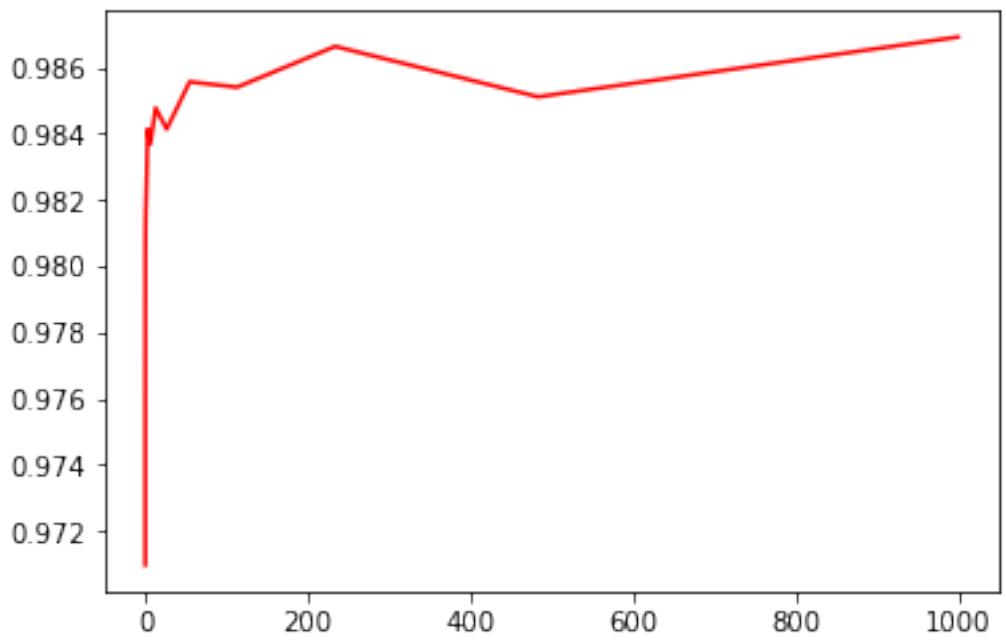
```
f1_list=[]
f2_list=[]
i_list=[]
all_list=[]
trainyHat_list=[]
misclassificate_rate=[]
lamlist=list(numpy.logspace(-3,3,20))
for i in lamlist:
    clf = svm.SVC(C=i,kernel='linear')
    clf.fit(trainx,trainy)
    trainyHat=clf.predict(trainx)
    trainyHat_list.append(trainyHat)
    err=0
    for j in range(len(trainyHat)):

        if trainyHat[j]!=trainy[j]:
            err+=1
    misclassificate_rate.append(float(err)/float(len(trainyHat)))
    f1=f1_score(trainy, trainyHat, average='weighted')
    f2=fbeta_score(trainy, trainyHat, average='weighted',beta=0.5)
    f1_list.append(f1)
    f2_list.append(f2)
    i_list.append(i)
    all_list.append((i,f1,f2,float(err)/float(len(trainyHat))))
```

find out the best parameter C for linear kernel

In [16]:

```
ax=plt.gca()
ax.plot(i_list,misclassificate_rate,"g",label="misclassification")
plt.show()
a=plt.gca()
a.plot(i_list,f1_list,"r",label="f1")
plt.show()

f1_sorted=sorted(all_list,key=itemgetter(1),reverse=True)
misclassification_sorted=sorted(all_list,key=itemgetter(3),reverse=False)
print f1_sorted[0:5]
print misclassification_sorted[0:5]
```






```
[(1000.0, 0.98691782903405134, 0.98677545609502948, 0.0127616130678917
82), (233.57214690901213, 0.98663590318803018, 0.98644521896530546, 0.
012761613067891782), (54.555947811685144, 0.98556677544307247, 0.98534
229698309617, 0.0137825421133231 24), (112.88378916846884, 0.9854042547
7706362, 0.98517106397629906, 0.0137825421133231 24), (483.293023857175
23, 0.98511373115938072, 0.98488527441882923, 0.014293006636038795)]
[(233.57214690901213, 0.98663590318803018, 0.98644521896530546, 0.0127
616 13067891782), (1000.0, 0.98691782903405134, 0.98677545609502948, 0.
012761613067891782), (54.555947811685144, 0.98556677544307247, 0.98534
229698309617, 0.0137825421133231 24), (112.88378916846884, 0.9854042547
7706362, 0.98517106397629906, 0.0137825421133231 24), (12.7427498570313
22, 0.98477652908660995, 0.9845209371795679, 0.014293006636038795)]
```

the best c for linear kernel is 1000, with highest f1 score at 0.98691782903405134, with err_rate at 0.012761613067891782

find out the best parameter C,and degree for poly kernel

In [2]:

```
f1_list=[]
f2_list=[]
i_list=[]
all_list=[]
trainyHat_list=[]
```

```
trainyHat_list=[]
misclassificate_rate=[]
d_list=[]
lamlist=list(numpy.logspace(-3,3,20))
for i in lamlist:
    for d in range(3):
        clf = svm.SVC(C=i,degree=d,kernel='poly')
        clf.fit(trainx,trainy)
        trainyHat=clf.predict(trainx)
        trainyHat_list.append(trainyHat)
        err=0
        for j in range(len(trainyHat)):

            if trainyHat[j]!=trainy[j]:
                err+=1
        print err
        misclassificate_rate.append(float(err)/float(len(trainyHat)))
        f1=f1_score(trainy, trainyHat, average='weighted')
        f2=fbeta_score(trainy, trainyHat, average='weighted',beta=0.5)
        f1_list.append(f1)
        f2_list.append(f2)
        i_list.append(i)
        d_list.append(d)
        all_list.append((i,d,f1,f2,float(err)/float(len(trainyHat))))
```

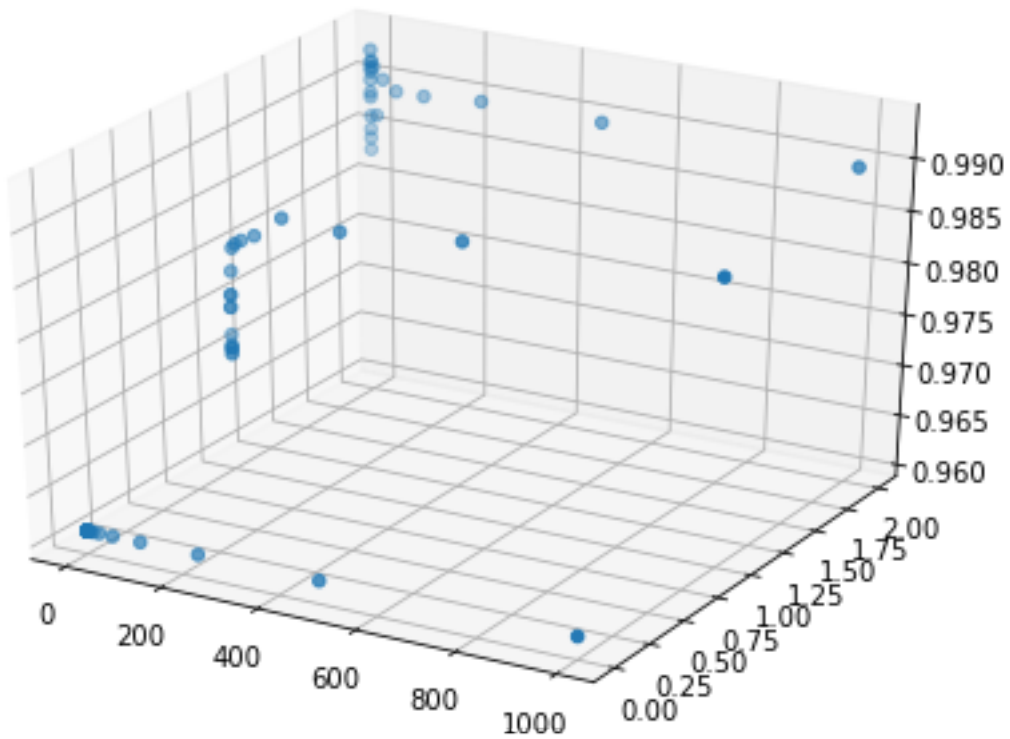
```
51
44
```

```
/Users/wendy/anaconda/lib/python2.7/site-packages/sklearn/metrics/clas
sification.py:1113: UndefinedMetricWarning: F-score is ill-defined and
being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)


30
51
44
28
51
44
27
51
44
25
51
44
22
51
43
21
51
42
19
```

51
40
17
51
40
16
51
39
14
51
39
18
51
36
16
51
34
17
51
33
25
51
32
19
51
31
21
51
28
21
51
29
20
51
27
20
51
25
20

In [5]:

```
fig=plt.figure()
ax=Axes3D(fig)
ax.scatter(i_list,d_list,f1_list)
plt.show()
f1_sorted=sorted(all_list,key=itemgetter(2),reverse=True)
misclassification_sorted=sorted(all_list,key=itemgetter(4),reverse=False)
print f1_sorted[0:5]
print misclassification_sorted[0:5]
```




```
[(0.69519279617756058, 2, 0.9927113913002642, 0.99266229968103303, 0.0
0714650331801939 75), (2.9763514416313162, 2, 0.9916701614860 1639, 0.99
160805196997337, 0.008167432363450 74), (0.33598182862837811, 2, 0.9915
8374623261958, 0.9915163023872 7517, 0.008167432363450 74), (6.158482110
6602607, 2, 0.9911041237431 5504, 0.9910286757637 7773, 0.00867789688616
6412), (0.16237767391887209, 2, 0.9910103358366 3134, 0.990934963121038
46, 0.0086778968861 66412)]
[(0.69519279617756058, 2, 0.9927113913002642, 0.99266229968103303, 0.0
0714650331801939 75), (0.33598182862837811, 2, 0.9915837462326 1958, 0.9
9151630238727517, 0.008167432363450 74), (2.9763514416313162, 2, 0.9916
7016148601639, 0.9916080519699 7337, 0.008167432363450 74), (0.162377673
91887209, 2, 0.9910103358366 3134, 0.9909349631210 3846, 0.0086778968861
66412), (6.1584821106602607, 2, 0.9911041237431 5504, 0.9910286757637 77
73, 0.0086778968861 66412)]
```

the optimal c is 0.69519279617756058, the optimal degree is 2, with f1 score :0.9927113913002642, err_rate: 0.0071465033180193975

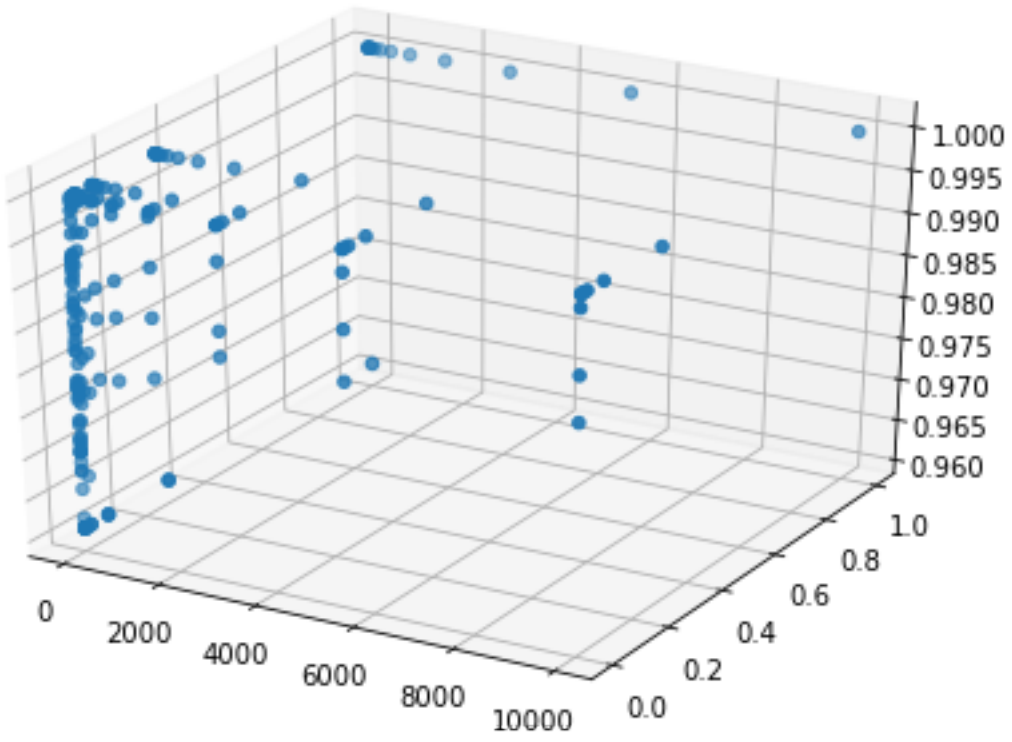
find out the best parameter C,and gamma for rbf kernel

In [ ]:

```
f1_list=[]
f2_list=[]
i_list=[]
all_list=[]
trainyHat_list=[]
misclassificate_rate=[]
gama_list=[]
lamlist=list(numpy.logspace(-1,4,20))
gamalist=list(numpy.logspace(-5,0,10))
for i in lamlist:
    for g in gamalist:
        clf = svm.SVC(C=i,gamma=g,kernel='rbf')
        clf.fit(trainx,trainy)
        trainyHat=clf.predict(trainx)
        trainyHat_list.append(trainyHat)
        err=0
        for j in range(len(trainyHat)):

            if trainyHat[j]!=trainy[j]:
                err+=1

        misclassificate_rate.append(float(err)/float(len(trainyHat)))
        f1=f1_score(trainy, trainyHat, average='weighted')
        f2=fbeta_score(trainy, trainyHat, average='weighted',beta=0.5)
        f1_list.append(f1)
        f2_list.append(f2)
        i_list.append(i)
        gama_list.append(g)
        all_list.append((i,g,f1,f2,float(err)/float(len(trainyHat))))
```

In [5]:

```
fig=plt.figure()
ax=Axes3D(fig)
ax.scatter(i_list,gama_list,f1_list)
plt.show()

f1_sorted=sorted(all_list,key=itemgetter(2),reverse=True)
misclassification_sorted=sorted(all_list,key=itemgetter(4),reverse=False)
print f1_sorted[0:5]
print misclassification_sorted[0:5]
```




```
[(1.1288378916846888, 0.07742636826811277, 1.0, 1.0, 0.0), (1.1288378
916846888, 0.27825594022071259, 1.0, 1.0, 0.0), (1.1288378916846888, 1
.0, 1.0, 1.0, 0.0), (2.0691380811147893, 0.07742636826811277, 1.0, 1.
0, 0.0), (2.0691380811147893, 0.27825594022071259, 1.0, 1.0, 0.0)]
[(1.1288378916846888, 0.07742636826811277, 1.0, 1.0, 0.0), (1.1288378
916846888, 0.27825594022071259, 1.0, 1.0, 0.0), (1.1288378916846888, 1
.0, 1.0, 1.0, 0.0), (2.0691380811147893, 0.07742636826811277, 1.0, 1.
0, 0.0), (2.0691380811147893, 0.27825594022071259, 1.0, 1.0, 0.0)]
```

the optimal c for rbf is 1.1288378916846888, the optimal gamma is 0.07742636826811277 with f1 score: 1 and err_rate: 0

In [12]:

```
def evaluation(trainx,trainy,testx,testy,clf):
    print clf
    clf.fit(trainx,trainy)
    trainyHat=clf.predict(trainx)
    linear_train_f1=f1_score(trainy, trainyHat, average='weighted')
    testyHat=clf.predict(testx)
    linear_test_f1=f1_score(testy, testyHat, average='weighted')
    return linear_train_f1,linear_test_f1
```

evaluation metrics for the training and test set for all three models

In [21]:

```
clf = svm.SVC(C=1000,kernel='linear')
linear_train_f1,linear_test_f1=evaluation(trainx,trainy,testx,testy,clf)
```

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

In [23]:

```
clf = svm.SVC(C=0.69519279617756058,degree=2,kernel='poly')
poly_train_f1,poly_test_f1=evaluation(trainx,trainy,testx,testy,clf)
```

```
SVC(C=0.695192796178, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=2, gamma='auto', kernel='poly',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [24]:

```
clf = svm.SVC(C=1.1288378916846888,gamma=0.077426368268112777,kernel='rbf')
rbf_train_f1,rbf_test_f1=evaluation(trainx,trainy,testx,testy,clf)
```

```
SVC(C=1.12883789168, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.0774263682681,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [25]:

```
print 'f1 score on training with linear kernel:',linear_train_f1
print 'f1 score on testing with linear kernel:',linear_test_f1
print 'f1 score on training with poly kernel:',poly_train_f1
print 'f1 score on testing with poly kernel:',poly_test_f1
print 'f1 score on training with rbf kernel:',rbf_train_f1
print 'f1 score on testing with rbf kernel:',rbf_test_f1
```

```
f1 score on training with linear kernel: 0.986917829034
f1 score on testing with linear kernel: 0.977142080437
f1 score on training with poly kernel: 0.9927113913
f1 score on testing with poly kernel: 0.978190802956
f1 score on training with rbf kernel: 1.0
f1 score on testing with rbf kernel: 0.953869329703
```

Table for f1 score:

	training	testing
linear	0.986917829034	0.977142080437
poly	0.9927113913	0.978190802956
rbf	1.0	0.953869329703

Generally, rbf kernel has a better berformance than poly kernel than linear kernel on both trsining and testing data. However although rbf kernel can have athe highest f1score of 1 on training data, it actually has a problem of overfitting, and has a wore performance on testing data on two other kernel

In [ ]: