In [1]:

```
%matplotlib inline
```

Identificación de la edad del usuario

Para identificar el género vamos a tener que usar el feature selection que definimos anteriormente. Para ello hemos creado un archivo .py en el que se realiza esta operación. Probaremos con todas las opciones realizadas y veremos con cual nos quedamos al final.

Puesto que tenemos datos con etiquetas de edad, usaremos modelos supervisados para la edad.

1.- Librerias

In [2]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import re
import time
import timeit

from sklearn import cross_validation
from sklearn.feature_selection import VarianceThreshold
from sklearn.decomposition import PCA
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.externals import joblib
import sklearn.preprocessing as pp
import random


import dateutil
#Hay que istalar esta librería que hace el parseo del user agent
#pip install pyyaml ua-parser user-agents

#Para pintar gráficos vistosos usamos seaborn:
import seaborn as sns

#y creamos la paleta:
sns.set_palette("deep", desat=.6)
sns.set_context(rc={"figure.figsize": (8, 4)})
```

2.- Descripcion de los datos

DESPUES DE ANONIMIZAR Y SELECCIONAR ÚNICAMENTE LAS VARIABLES QUE QUEREMOS

num_columna	**Nombre**	**Descripción**	**Variable**
1	ciudad	ciuda de origen del usuario	discreta
2	email_server	servidor de email del usuario	discreta
3	**edad**	edad del usuario (variable objetivo)	discreta
4	**genero**	genero del usuario (variable objetivo)	discreta
6	hora_visita	hora en que el usuario hace la visita	discreta
7	is_weekend	fin de semana	discreta
8	nombre_final	nombre del usuario	discreta
9	os	sistema operativo	discreta
10	pais	pais en el user agent	discreta
11	rango horario	momento del día en que se conecta el usuario	discreta
12	time_zone	zona horaria del usuario	discreta
13	ua_browser_family	familia del navegador en el user agent	discreta
14	ua_device	dispositivo que utiliza el usuario segun user agent	discreta
15	ua_device_family	familia del dispositivo en el user agent	discreta
16	ua_is_bot	si es un robot	discreta
17	ua_is_movile	si es un movil	discreta
19	ua_is_pc	si es un pc	discreta
20	ua_is_tablet	si es una tablet	discreta
21	ua_is_tounch_capable	si es táctil	discreta
22	ua_os_family	familia sistema operativo	discreta
23	weekday	dia de la semana	discreta
24	id_hotspots	id del local	discreta

-Faltaría saber si se ha conectado con facebook, google o email (debería hacerlo en la recolección de variables), así como rellenar los nulos con un valor ("vacio")

-También faltaría la categoría del local en que se ha conectado y hacer algo con las provincias.

3.- Descripción del DataFrame de resultados

id	Nombre	Descripción
1	feature_reduction	Tipo de reducción de características utilizado. Será: • Todas • Varianza • EstraTree Classifier • PCA
2	Model	Modelo supervisado utilizado. Será: • Decission_tree • Random_forest • Linear Regression • Logistic Regression • SVM
3	target	Variable objetivo del modelo. Sera: • edad • mayor_edad • rango_edad
4	resultado	score del modelo con datos de test
5	parameters	modelo utilizado con sus parámetros
6	exec_time	tiempo que tarda en predecir el modelo

En total son 4x5x3 pruebas diferentes = 60 experimentos. Se guardarán en un dataFrame para evaluar al final cual da mejores resultados.

In [265]:

```
#Creo un dataframe donde voy a guardar los resultados de los modelos para u
sarlo al final.
#Los campos que tendrá serán:
    # - ferture reduction,  Varianza, ExtraTree, PCA
    # - Model : tree, random_forest, regression ...
    # - target :  mayor_edad, edad, rango_edad
    # - Score
    # - Parameters: Parametros que pasamos a la función

Resultados = pd.DataFrame
Feature_Reduction = []
Model = []
Target =[]
Final_Score = []
Parameters = []
Exec_time = []
```

id	FR	Modelo	Target	Realizado
1	Varianza	Decisssion_tree	edad	Si
2	Varianza	Decisssion_tree	mayor_edad	Si
3	Varianza	Decisssion_tree	rango_edad	Si
4	Varianza	Random_forest	edad	Si
5	Varianza	Random_forest	mayor_edad	Si
6	Varianza	Random_forest	rango_edad	Si
7	Varianza	Linear Regression	edad	Si
8	Varianza	Linear Regression	mayor_edad	Si
9	Varianza	Linear Regression	rango_edad	Si
10	Varianza	Logistic Regression	edad	Si
11	Varianza	Logistic Regression	mayor_edad	Si
12	Varianza	Logistic Regression	rango_edad	Si
13	Varianza	SVM	edad	Si
14	Varianza	SVM	mayor_edad	Si
14	Varianza	SVM	rango_edad	Si
16	Extra_tree	Decisssion_tree	edad	Si
17	Extra_tree	Decisssion_tree	mayor_edad	Si
18	Extra_tree	Decisssion_tree	rango_edad	Si
19	Extra_tree	Random_forest	edad	Si
20	Extra_tree	Random_forest	mayor_edad	Si
21	Extra_tree	Random_forest	rango_edad	Si
22	Extra_tree	Linear Regression	edad	Si
23	Extra_tree	Linear Regression	mayor_edad	Si
24	Extra_tree	Linear Regression	rango_edad	Si
25	Extra_tree	Logistic Regression	edad	Si
26	Extra_tree	Logistic Regression	mayor_edad	Si
27	Extra_tree	Logistic Regression	rango_edad	Si
28	Extra_tree	SVM	edad	Si
29	Extra_tree	SVM	mayor_edad	Si
30	Extra_tree	SVM	rango_edad	Si
31	PCA	Decisssion_tree	edad	Si
32	PCA	Decisssion_tree	mayor_edad	Si

33	PCA	Decisssion_tree	rango_edad	Si
34	PCA	Random_forest	edad	Si
35	PCA	Random_forest	mayor_edad	Si
36	PCA	Random_forest	rango_edad	Si
37	PCA	Linear Regression	edad	Si
38	PCA	Linear Regression	mayor_edad	Si
39	PCA	Linear Regression	rango_edad	Si
40	PCA	Logistic Regression	edad	Si
41	PCA	Logistic Regression	mayor_edad	Si
42	PCA	Logistic Regression	rango_edad	Si
43	PCA	SVM	edad	Si
44	PCA	SVM	mayor_edad	Si
45	PCA	SVM	rango_edad	Si
46	Todas	Decisssion_tree	edad	Si
47	Todas	Decisssion_tree	mayor_edad	Si
48	Todas	Decisssion_tree	rango_edad	Si
49	Todas	Random_forest	edad	Si
50	Todas	Random_forest	mayor_edad	Si
51	Todas	Random_forest	rango_edad	Si
52	Todas	Linear Regression	edad	Si
53	Todas	Linear Regression	mayor_edad	Si
54	Todas	Linear Regression	rango_edad	Si
55	Todas	Logistic Regression	edad	Si
56	Todas	Logistic Regression	mayor_edad	Si
57	Todas	Logistic Regression	rango_edad	Si
58	Todas	SVM	edad	Si
59	Todas	SVM	mayor_edad	Si
60	Todas	SVM	rango_edad	Si

4.- Carga de los datos

Cargamos los datos que hemos limpiado anteriormente y guardado en un csv para cargarlos más fácilmente). Al final del ejercicio habría que integrarlo todo en un único proceso para su uso.

In [4]:

```
df = pd.read_csv('../csv/datos_explorados.csv')

#borro la columna unnamed
df.drop('Unnamed: 0', axis=1,inplace=True)

#y quito ciudad, ua_os_family  y ua_device_family, ua_is_pc

df.drop(['ua_os_family','ua_device','ciudad','ua_is_pc'], axis=1,inplace=Tr
ue)


print df.columns
```

```
Index([u'email_server', u'edad', u'genero', u'hora_visita', u'idioma', u'i
s_weekend', u'nombre_final', u'os', u'pais', u'rango_horario', u'timezone',
u'ua_browser_family', u'ua_device_family', u'ua_is_bot', u'ua_is_movile',
u'ua_is_tablet', u'ua_is_tounch_capable', u'weekday', u'id_hotspots'], dtyp
e='object')
```

5.-Feature Reduction

In [5]:

```
#Leemos los fichero que hemos creado con label_encode
def label_econde(df):
                df_features = pd.DataFrame()
                le_ciudad = joblib.load('models/le_ciudad.pkl')
                le_email_server = joblib.load('models/le_email_server.pkl')
                le_idioma = joblib.load('models/le_idioma.pkl')
                le_os = joblib.load('models/le_os.pkl')
                le_pais = joblib.load('models/le_pais.pkl')
                le_rango_horario = joblib.load('models/le_rango_horario.pk
l')
                le_time_zone = joblib.load('models/le_time_zone.pkl')
                le_browser_family = joblib.load('models/le_browser_family.p
kl')
                le_rango_horario = joblib.load('models/le_rango_horario.pk
l')
                le_device = joblib.load('models/le_device.pkl')
                le_device_family = joblib.load('models/le_device_family.pk
l')
                le_os_family = joblib.load('models/le_os_family.pkl')

                #df_features["ciudad"] = le_ciudad.transform(df.ciudad)
                df_features["email_server"] = le_email_server.transform(d
f.email_server)
                df_features["hora_visita"] = df.hora_visita
                df_features["idioma"] = le_idioma.transform(df.idioma)
                df_features["is_weekend"] = [1 if x else 0 for x in df.is_w
eekend]
                df_features["os"] = le_os.transform(df.os)
                df_features["pais"] = le_pais.transform(df.pais)
                df_features["rango_horario"] = le_rango_horario.transform(d
f.rango_horario)
                df_features["time_zone"] = le_time_zone.transform(df.timezo
ne)
                df_features["browser_family"] = le_browser_family.transfor
m(df.ua_browser_family)
                #df_features["device"] = le_device.transform(df.ua_device)
                df_features["device_family"] = le_device_family.transform(d
f.ua_device_family)
                df_features["is_movile"] = [1 if x else 0 for x in df.ua_i
s_movile]
                #df_features["is_pc"] = [1 if x else 0 for x in df.ua_is_p
c]
                df_features["is_tablet"] = [1 if x else 0 for x in df.ua_i
s_tablet]
                df_features["is_tounch_capable"] = [1 if x else 0 for x in
df.ua_is_tounch_capable]
                #df_features["os_family"] = le_os_family.transform(df.ua_o
s_family)
                df_features["weekday"] = df.weekday
                df_features["id_hotspots"] = df.id_hotspots
                return df_features
```

In [6]:

```
df_lb_features = label_econde(df)
```

In [7]:

```
#Separamos los que tienen edad de los que no
X_train = df_lb_features[pd.notnull(df.edad)]
X_predecir= df_lb_features[pd.isnull(df.edad)]
y_train =  df[pd.notnull(df.edad)].edad.values
print len(X_train)
print len(X_predecir)
print len(y_train)
```

```
10895
15183
10895
```

In [8]:

```
#VAmos a crear un set de prueba y test con todas las variables



X_train_todas, X_test_todas, y_train_todas, y_test_todas = (cross_validatio
n.train_test_split
                                                            (X_train
                                                             , y_train
                                                             , test_siz
e=0.4
                                                             , random_stat
e=0))
```

1.- Varianza

In [9]:

```
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
X_sel_new = sel.fit_transform(df_lb_features)

print df_lb_features.shape
print sel.get_support()
print X_sel_new.shape

df_features_varianza = df_lb_features[df_lb_features.columns[sel.get_suppor
t()]]
print df_features_varianza.columns
```

```
(26078, 15)
[ True  True  True  True  True  True  True  True  True  True False False
 False  True  True]
(26078, 12)
Index([u'email_server', u'hora_visita', u'idioma', u'is_weekend', u'os',
u'pais', u'rango_horario', u'time_zone', u'browser_family', u'device_famil
y', u'weekday', u'id_hotspots'], dtype='object')
```

In [10]:

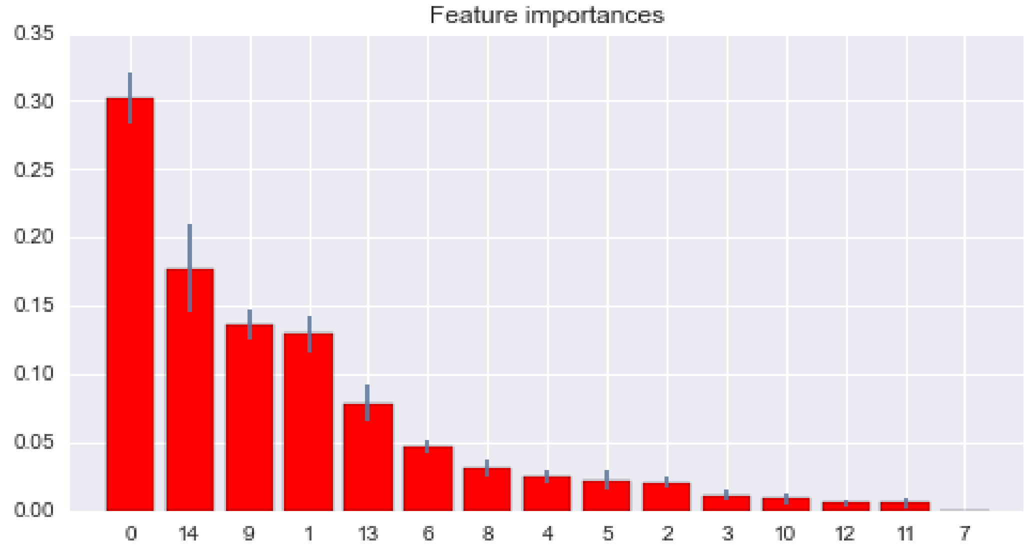
```
#usamos las columnas de la varianza

def convert_varianza(df,b_index):
    return df[df.columns[b_index]]


X_varianza = convert_varianza(X_train,sel.get_support())
X_predecir_varianza = convert_varianza(X_predecir,sel.get_support())

#Creamos dataframes de entrenamiento con la varianza

X_train_varianza, X_test_varianza, y_train_varianza, y_test_varianza = (cro
ss_validation
                                                                        .tr
ain_test_split

(X_varianza
                                                                         ,
y_train
                                                                         ,
test_size=0.4
                                                                         ,
random_state=0))
```

2.- Extra_tree

In [11]:

```
X = X_train
y = y_train
print X.shape

num_features = len(X_train.columns)
clf = ExtraTreesClassifier()
X_new = clf.fit(X, y).transform(X)

print "Features importances:" , clf.feature_importances_
new_num_features = X_new.shape[1]
print "Features a utilizar:" ,  new_num_features

importances = clf.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]


# Print the feature ranking
print("Feature ranking:")

i= 0
indice_ET = []
for f in range(num_features):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

    if i < new_num_features:
        indice_ET.append( indices[f])
    i+=1



# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(num_features), importances[indices],color="r", yerr=std[indices], align="center")
plt.xticks(range(num_features), indices)
plt.xlim([-1, num_features])
plt.show()
```

```
(10895, 15)
Features importances: [ 0.30191805  0.12910873  0.02038684  0.01143885  0.0
2454446  0.02201085
  0.04726038  0.          0.03109689  0.13627838  0.00838815  0.00559224
  0.00563301  0.07879103  0.17755212]
Features a utilizar: 5
Feature ranking:
1. feature 0 (0.301918)
2. feature 14 (0.177552)
3. feature 9 (0.136278)
4. feature 1 (0.129109)
5. feature 13 (0.078791)
6. feature 6 (0.047260)
7. feature 8 (0.031097)
8. feature 4 (0.024544)
9. feature 5 (0.022011)
10. feature 2 (0.020387)
11. feature 3 (0.011439)
12. feature 10 (0.008388)
13. feature 12 (0.005633)
14. feature 11 (0.005592)
15. feature 7 (0.000000)
```



In [12]:

```
#Vamos a hacer el subset:

X_ET = X[indice_ET]
X_predecir_ET = X_predecir[indice_ET]

X_train_ET, X_test_ET, y_train_ET, y_test_ET = cross_validation.train_tes
t_split(X_ET
                                                                     , y_tr
ain
                                                                     , tes
t_size=0.4
                                                                     , rand
om_state=0)
```

3.- PCA

In [13]:

```
cat_not_bin = [ 'email_server', 'hora_visita', 'idioma', 'os', 'pais', 'ran
go_horario',
               'timezone', 'ua_browser_family', 'ua_device_family' , 'weekd
ay', 'id_hotspots']

cat_bin=['is_weekend', 'ua_is_movile', 'ua_is_tounch_capable', 'ua_is_table
t']

#Y las columnas objetivo
cat_targets = ['edad', 'genero']
#Covierto las columnas boolean en 0 y 1

columnas = np.concatenate((cat_bin,cat_not_bin,cat_targets), axis=1)

df_nuevo = df[columnas]
for col in df_nuevo[cat_bin].columns:
    df_nuevo.loc[:,(col)]  = [1 if x else 0 for x in df_nuevo[col]]
```

```
/Users/Ana/anaconda/lib/python2.7/site-packages/pandas/core/indexing.py:41
5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  self.obj[item] = s
```

In [14]:

```
# Función binariza que genera las columnas binarizadas
def binariza(dataf, cat_bin, cat_not_bin):
    df_binarized_genero = pd.DataFrame()
    df_binarized = dataf[cat_bin]
    #Y ahora añado las binarizadas

    for column in cat_not_bin:
        #Genero un arrat con las categorías que va a haber
        classes = df[column].unique().tolist()
        #Binarizo las columnas teniendo en cuenta las categorías

        column_bin = pp.label_binarize(dataf[column], classes)
        #y lo inserto en un dataframe dando nombre a las columnas
        df_bin = pd.DataFrame(column_bin,columns =
                                            ['is_'+ column + "_" + str(x).repl
ace(" ","_")
                                             for x in classes])

        #Como las variables binarizadas tienen un index distinto al de las
variables que ya
        #existían, al hacer el concat no se hace bien, por lo que ponemos e
l mismo índice
        #a las variables binarizadas que el que tenían las variables existe
ntes

        df_bin.index = df_binarized.index
        df_binarized = pd.concat((df_binarized,df_bin), axis=1)

    return df_binarized

df_binarizado = binariza(df_nuevo, cat_bin, cat_not_bin)
```

In [15]:

```
print df_binarizado.shape
```

```
(26078, 1273)
```

In [16]:

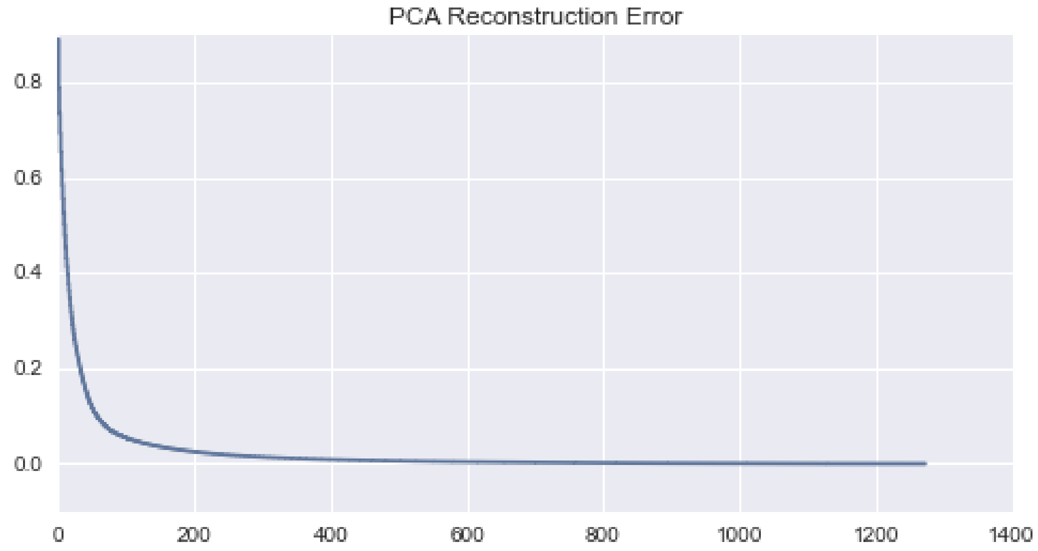
```
n_features = df_binarizado.columns.size
print "Total number of features for edad: %d" %n_features
```

```
Total number of features for edad: 1273
```

In [17]:

```
X_PCA = df_binarizado[pd.notnull(df.edad)]
X_predecir_PCA= df_binarizado[pd.isnull(df.edad)]


X_train_PCA, X_test_PCA, y_train_PCA, y_test_PCA = cross_validation.train_t
est_split(X_PCA
                                                                      , y_tr
ain
                                                                      , tes
t_size=0.4
                                                                      , rand
om_state=0)
print X_train_PCA.shape[1]

n_features = X_train_PCA.shape[1]
pca = PCA(n_components=n_features, whiten=False)
pca.fit(df_binarizado)
pca.explained_variance_ratio_[0:].cumsum()
plt.plot(1 - pca.explained_variance_ratio_.cumsum(), drawstyle = 'steps-pos
t')
plt.title('PCA Reconstruction Error');
```

```
1273
```



In [18]:

```
print df_binarizado.columns
```

```
Index([u'is_weekend', u'ua_is_movile', u'ua_is_tounch_capable', u'ua_is_tab
let', u'is_email_server_msn.com', u'is_email_server_hotmail.com', u'is_emai
l_server_naver.com', u'is_email_server_gmail.com', u'is_email_server_live.e
s', u'is_email_server_gmail.con', u'is_email_server_centrum.cz', u'is_emai
l_server_vacio', u'is_email_server_gmil.com', u'is_email_server_gg.con',
u'is_email_server_heineken.es', u'is_email_server_zadibe.es', u'is_email_se
rver_yahoo.com', u'is_email_server_wp.pl', u'is_email_server_yahoo.es', u'i
s_email_server_pepe.com', u'is_email_server_touristinfo.net', u'is_email_se
rver_libero.it', u'is_email_server_hotmail.es', u'is_email_server_tiscali.i
t', u'is_email_server_outlook.com', u'is_email_server_yahoo.it', u'is_emai
l_server_elpuig.org', u'is_email_server_ahora.es', u'is_email_server_alexal
caide.com', u'is_email_server_hotmail.it', u'is_email_server_me.com', u'i
s_email_server_ritasibarita.com', u'is_email_server_aferrando.com', u'is_em
ail_server_hotmail.col', u'is_email_server_gamil.com', u'is_email_server_ya
hoo.fr', u'is_email_server_mail.ru', u'is_email_server_infonegocio.com',
u'is_email_server_colesan.edu.co', u'is_email_server_hotmail.fr', u'is_emai
l_server_migue.com', u'is_email_server_net.hr', u'is_email_server_alumnos.u
chceu.es', u'is_email_server_Hotmail.com', u'is_email_server_hotmail.co.u
k', u'is_email_server_aol.com', u'is_email_server_dse.nl', u'is_email_serve
r_home.nl', u'is_email_server_gmail.es', u'is_email_server_iCloud.com', u'i
s_email_server_o2.pl', u'is_email_server_factoriasapiens.com', u'is_email_s
erver_live.com', u'is_email_server_mermelad.com', u'is_email_server_a.com',
u'is_email_server_live.be', u'is_email_server_t-online.de', u'is_email_serv
er_telefonica.net', u'is_email_server_virgilio.it', u'is_email_server_fff.c
o', u'is_email_server_zhaw.ch', u'is_email_server_uv.es', u'is_email_serve
r_gmx.com', u'is_email_server_live.it', u'is_email_server_alumni.uv.es',
u'is_email_server_btinternet.com', u'is_email_server_wanadoo.es', u'is_emai
l_server_postal.uv.es', u'is_email_server_terra.com', u'is_email_server_gm
x.ch', u'is_email_server_inmotello.com', u'is_email_server_mail.com', u'i
s_email_server_hotail.com', u'is_email_server_web.de', u'is_email_server_so
mnomed.com', u'is_email_server_live.fr', u'is_email_server_movistar.es',
u'is_email_server_ucm.es', u'is_email_server_kingston.ac.uk', u'is_email_se
rver_yahoo.co.uk', u'is_email_server_yahoo.com.br', u'is_email_server_vlccl
ubbing.com', u'is_email_server_sbcglobal.net', u'is_email_server_laqueado.c
om', u'is_email_server_togni.it', u'is_email_server_hoymail.com', u'is_emai
l_server_gmx.net', u'is_email_server_neuf.fr', u'is_email_server_outloock.c
om', u'is_email_server_baoproyectos.com', u'is_email_server_avory.es', u'i
s_email_server_gmx.de', u'is_email_server_victorgil.name', u'is_email_serve
r_fkfb.com', u'is_email_server_rocketmail.com', u'is_email_server_usc.edu',
u'is_email_server_ymail.com', u'is_email_server_live.co.uk', u'is_email_ser
ver_outlook.es', u'is_email_server_live.se', ...], dtype='object')
```

In [19]:

```
n_factors = sum(1-pca.explained_variance_ratio_[0:].cumsum() > 0.10)
print "Number of factors with 10% of reconstraction Error: ", n_factors

pca = PCA(n_components=n_factors)
pca.fit(df_binarizado)


print "Explained Variance Ratio"
print sum(pca.explained_variance_ratio_)

trainDS_pca = pca.transform(X_train_PCA)
X_predecir_PCA = pca.transform(X_predecir_PCA)
testDS_pca = pca.transform(X_test_PCA)
```

```
Number of factors with 10% of reconstraction Error:  57
Explained Variance Ratio
0.898230092731
```

In [20]:

```
X_train_PCA = trainDS_pca
X_test_PCA = testDS_pca
```

4.- Creacion de nuevas variables objetivo

In [21]:

```
# X_train_varianza, X_test_varianza, X_test_PCA, X_train_PCA, X_test_ET,
X_train_ET
```

Vamos a crear una variable objetivo que sea mayor de edad

In [22]:

```
y_train_mayor_edad_todas = [1 if y >= 18 else 0 for y in y_train_todas]
y_test_mayor_edad_todas = [1 if y >= 18 else 0 for y in y_test_todas]
y_train_mayor_edad = [1 if y >= 18 else 0 for y in y_train]
y_train_mayor_edad_V = [1 if y >= 18 else 0 for y in y_train_varianza]
y_test_mayor_edad_V = [1 if y >= 18 else 0 for y in y_test_varianza]
y_train_mayor_edad_ET = [1 if y >= 18 else 0 for y in y_train_ET]
y_test_mayor_edad_ET = [1 if y >= 18 else 0 for y in y_test_ET]
y_train_mayor_edad_PCA = [1 if y >= 18 else 0 for y in y_train_PCA]
y_test_mayor_edad_PCA = [1 if y >= 18 else 0 for y in y_test_PCA]
```

Y otra variable objetivo que sea rango de eadad

Vamos a crear los siguientes rangos de edad:

```
* < 18  --> 1
* 18-24 --> 2
* 25-34 --> 3
* 35-44 --> 4
* 45-54 --> 5
* 55-64 --> 6
* 65-74 --> 7
* > 74  --> 8
```

Para ello le damos un número a cada rango y creamos una variable objetivo con ese número

In [23]:

```
# Creamos un dataframe con el valor del rango por cada edad
edad = np.arange(1,100)

rango = [1 if x <18 else x  for x in edad]
rango = [2 if ( x >= 18 and x < 25) else x for x in rango]
rango = [3 if ( x >= 25 and x < 35) else x for x in rango]
rango = [4 if ( x >= 35 and x < 45) else x for x in rango]
rango = [5 if ( x >= 45 and x < 55) else x for x in rango]
rango = [6 if ( x >= 55 and x < 65) else x for x in rango]
rango = [7 if ( x >= 65 and x < 75) else x for x in rango]
rango = [8 if  x >= 75 else x for x in rango]

rango_edad = ["<18" if x ==1  else x  for x in rango]
rango_edad = ["18-24" if x==2 else x for x in rango_edad]
rango_edad = ["25-34" if x==3 else x for x in rango_edad]
rango_edad = ["35-44" if x==4 else x for x in rango_edad]
rango_edad = ["45-54" if x==5 else x for x in rango_edad]
rango_edad = ["55-64" if x==6  else x for x in rango_edad]
rango_edad = ["65-75" if x==7 else x for x in rango_edad]
rango_edad = [">75" if  x==8 else x for x in rango_edad]

df_rango_edad= pd.DataFrame()
df_rango_edad["edad"] = edad
df_rango_edad["rango"] = rango
df_rango_edad["rango_edad"] = rango_edad

df_rango = df_rango_edad[["rango","rango_edad"]]
df_rango = df_rango.drop_duplicates(("rango","rango_edad"),take_last=True)
df_rango = df_rango.reset_index(drop=True)


#Vamos a crear un campo con los rangos de edad:
def devuelve_rango(df_rango_edad, edad):
    #print edad
    rango = []
    for e in edad:
        if e > 0:
            rango.append(df_rango_edad[df_rango_edad.edad == e].rango.value
s[0])
        else:
            rango.append(0)
    return rango


y_rango = devuelve_rango(df_rango_edad,y_train)
```

In [24]:

```
y_train_rango_todas = devuelve_rango(df_rango_edad, y_train_todas)
y_test_rango_todas = devuelve_rango(df_rango_edad, y_test_todas)
y_train_rango = devuelve_rango(df_rango_edad,y_train)
y_train_rango_V = devuelve_rango(df_rango_edad, y_train_varianza)
y_test_rango_V = devuelve_rango(df_rango_edad, y_test_varianza)
y_train_rango_ET = devuelve_rango(df_rango_edad, y_train_ET)
y_test_rango_ET = devuelve_rango(df_rango_edad, y_test_ET)
y_train_rango_PCA = devuelve_rango(df_rango_edad, y_train_PCA)
y_test_rango_PCA = devuelve_rango(df_rango_edad,y_test_PCA)
```

6.-Decission Trees

In [25]:

```
from sklearn import tree

def Decission_tree (X_train, X_test, y_train,y_test, criterion):
    clt = tree.DecisionTreeClassifier(criterion=criterion)
    clt.fit(X_train, y_train)
    s = clt.score(X_test, y_test)
    d = clt.tree_.max_depth
    print "Score: " , s
    print "Depth: " , d
    print "Feature importances" , clt.feature_importances_
    return s , d , clt
```

In [26]:

```
#Get the best depth:
def getDecisionTreeMesures(initArg = 'gini', max_depth = 2,
                           train_labels = None, train_data = None,
                           test_label = None, test_data = None):
    model = tree.DecisionTreeClassifier(criterion=initArg, max_depth=max_de
pth)
    model.fit(train_data, train_labels)
    return [model.tree_.max_depth,
            model.score(train_data, train_labels), #E_in
            model.score(test_data, test_label)] #E_out
```

In [27]:

```
def pinta_grafico_tree(gini_measures
                       , ax1_ylim1, ax1_ylim2, ax2_ylim1, ax2_ylim2
                       , criterion="gini", xlabel="Max Depth of Tree"):
    fig, axes = plt.subplots(ncols=2, figsize=(13, 5) )
    ax1, ax2 = axes.ravel()

    #ax1.figure(figsize=(7,5))
    ax1.plot(gini_measures[:,0], gini_measures[:,1], label = 'Score in', c
= 'b')
    ax1.legend(loc=4)
    ax1.set_ylim(ax1_ylim1,ax1_ylim2)
    ax1.set_title("Decision Tree Scores ("+criterion+" criterion)")
    ax1.set_xlabel(xlabel)
    ax1.set_ylabel("Scores In");

    #ax2.figure(figsize=(7,5))
    ax2.plot(gini_measures[:,0], gini_measures[:,2], label = 'Score out',
c='g')
    ax2.legend(loc=4)
    ax2.set_ylim(ax2_ylim1,ax2_ylim2)
    ax2.set_title("Decision Tree Scores ("+criterion+ " criterion)")
    ax2.set_xlabel(xlabel)
    ax2.set_ylabel("Scores Out");
```

In [28]:

```
def pinta_grafico_tree_compara(gini_measures
                               , entropy_measures
                               , criterion="gini-entropy"
                               , xlabel="Max Depth of Tree"):
    #ax1 = plt.plot(ncols=1, figsize=(13, 5) )

    plt.plot(gini_measures[:,0], gini_measures[:,2], label = 'gini', c =
'b')
    plt.plot(entropy_measures[:,0], entropy_measures[:,2], label = 'entrop
y', c = 'r')
    plt.legend(loc=4)
    plt.title("Decision Tree Scores ("+criterion+" criterion)")
    plt.xlabel(xlabel)
    plt.ylabel("Scores In");
```

1.-Todas

- edad

In [29]:

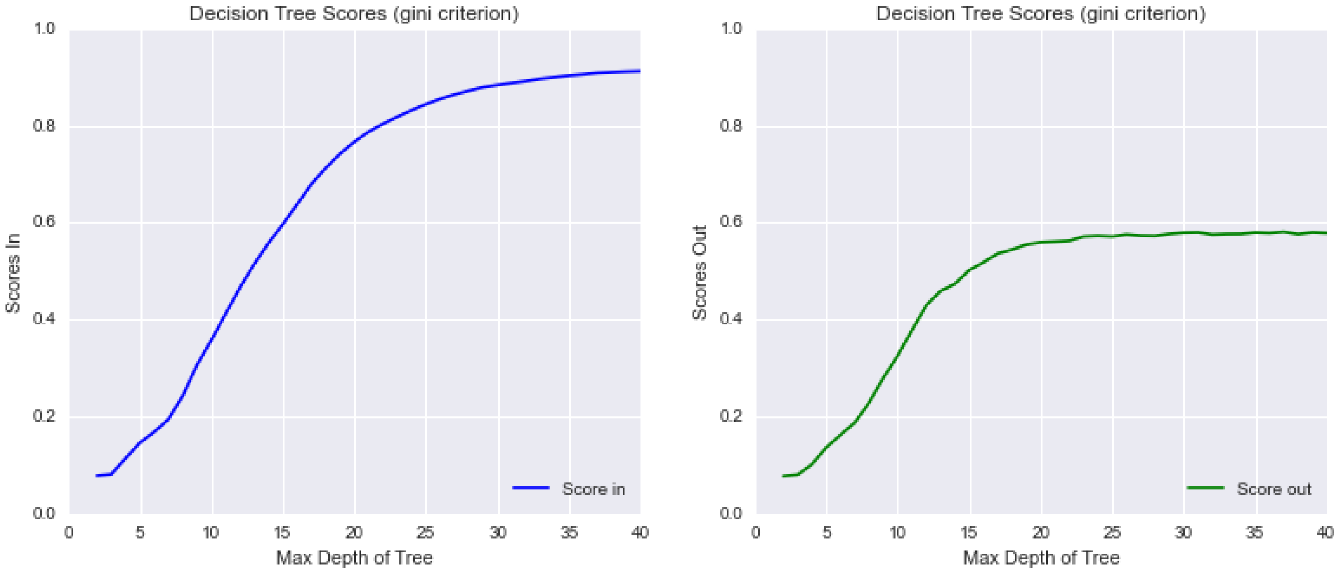
```
profundidad = 40
```

In [30]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                  , y_train_todas, X_train_t
odas
                                                  , y_test_todas, X_test_tod
as)
                          for max_depth in range(profundidad,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                 , y_train_todas, X_train_to
das
                                                  , y_test_todas, X_test_tod
as)
                          for max_depth in range(profundidad,1, -1)])
```

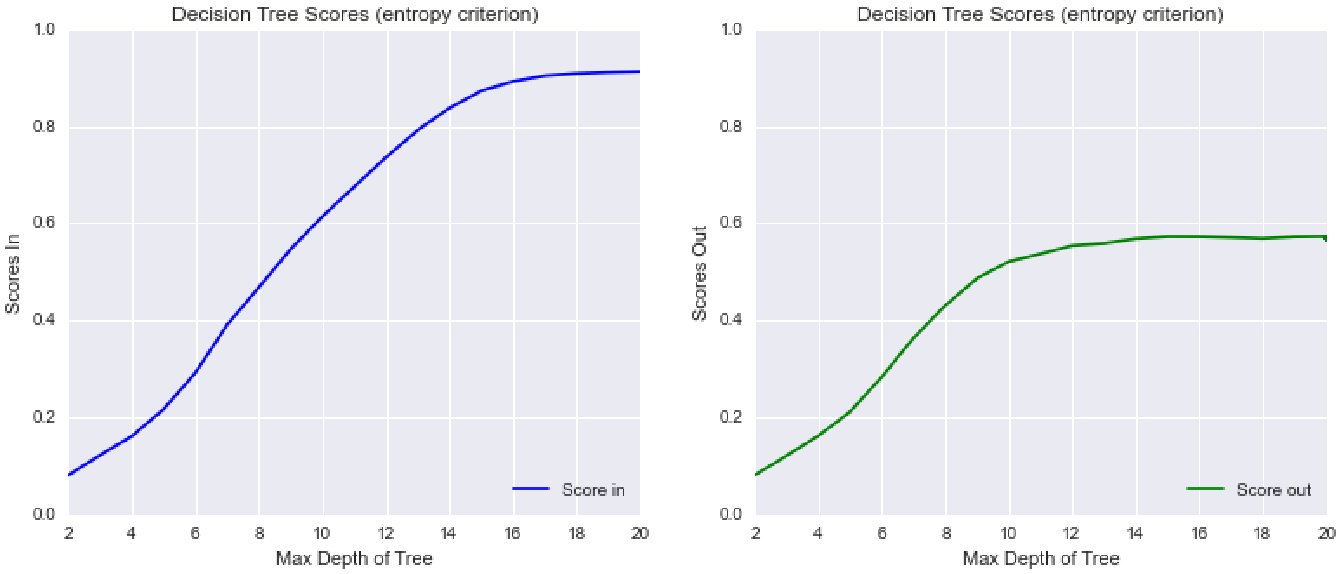
In [31]:

```
pinta_grafico_tree(gini_measures,0,1,0,1,'gini')
```



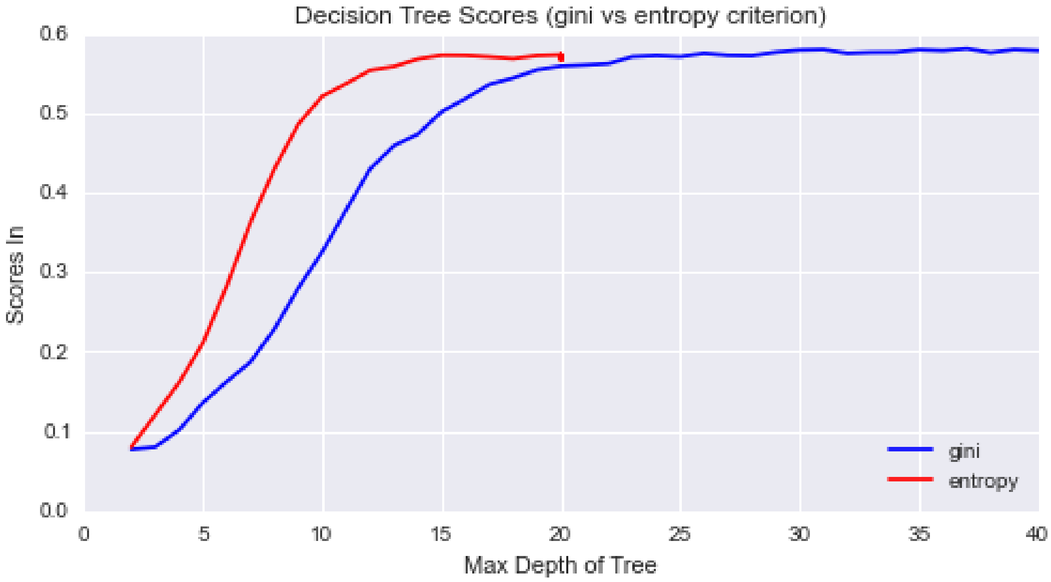
In [32]:

```
pinta_grafico_tree(entropy_measures,0,1,0,1,'entropy')
```



In [33]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entrop
y')
```



Mejor entropy porque corta el árbol antes y la profundidad 15

In [266]:

```
model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=15)
model.fit(X_train_todas, y_train_todas)
score = model.score(X_test_todas,y_test_todas)
print "Score:" , score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec time: " ,  total_time
```

```
Score: 0.572739788894
Exec time:  0.00132417678833
```

In [267]:

```
Feature_Reduction.append("Todas")
Model.append("Decission_tree")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

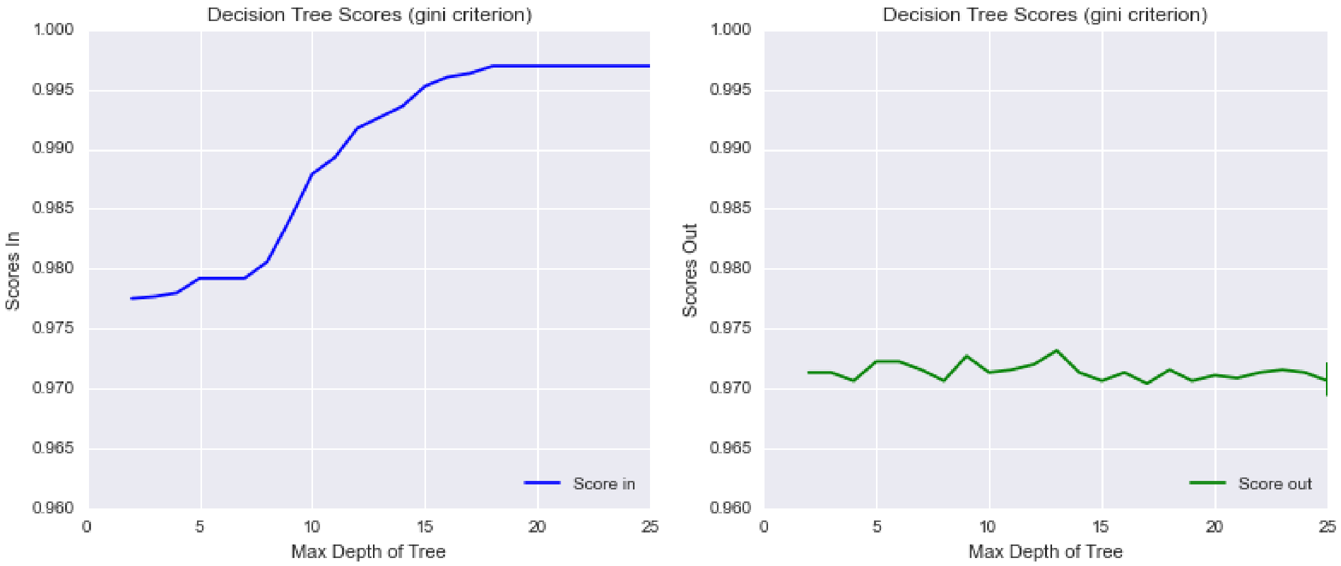
- mayor_edad

In [36]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                 , y_train_mayor_edad_toda
s, X_train_todas
                                                 , y_test_mayor_edad_todas,
X_test_todas)
                          for max_depth in range(profundidad,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                , y_train_mayor_edad_todas,
X_train_todas
                                                 , y_test_mayor_edad_todas,
X_test_todas)
                          for max_depth in range(profundidad,1, -1)])
```

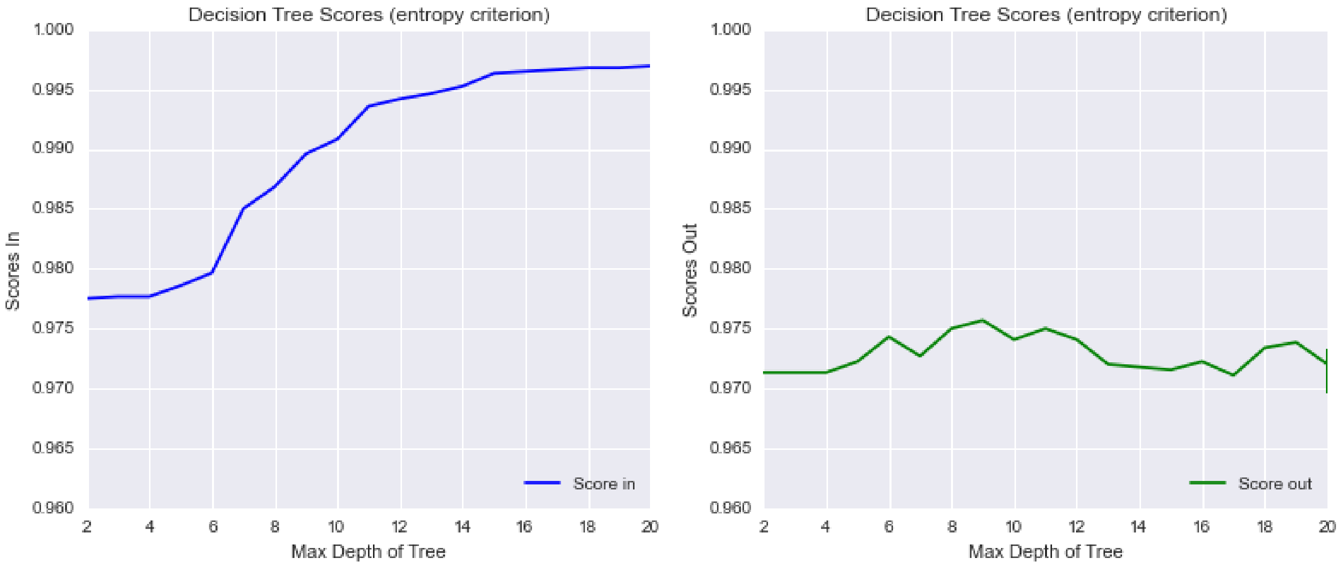
In [37]:

```
pinta_grafico_tree(gini_measures,0.96,1,0.96,1,'gini')
```



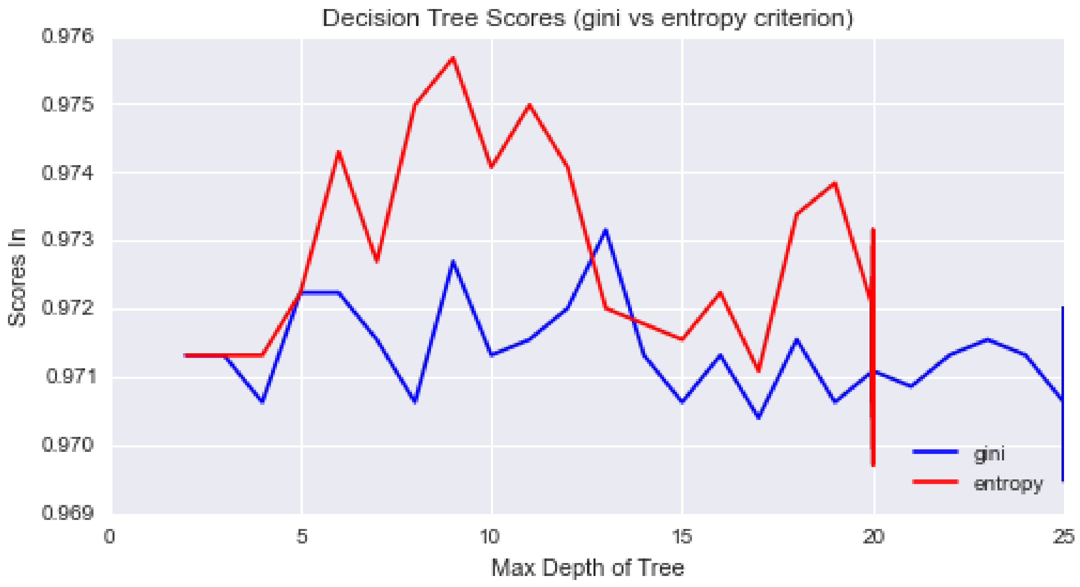
In [38]:

```
pinta_grafico_tree(entropy_measures,0.96,1,0.96,1,'entropy')
```



In [39]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entrop
y')
```



Mejor resultado entropy con de profundidad 9

In [268]:

```
model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=9)
model.fit(X_train_todas, y_train_mayor_edad_todas)
score = model.score(X_test_todas,y_test_mayor_edad_todas)
print "Score:", score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec time: ",total_time , "s"
```

```
Score: 0.974300137678
Exec time:  0.00151610374451 s
```

In [269]:

```
Feature_Reduction.append("Todas")
Model.append("Decission_tree")
Target.append("mayor_edad")
Final_Score.append(model.score(X_test_todas,y_test_mayor_edad_todas))
Parameters.append(model)
Exec_time.append(total_time)
```

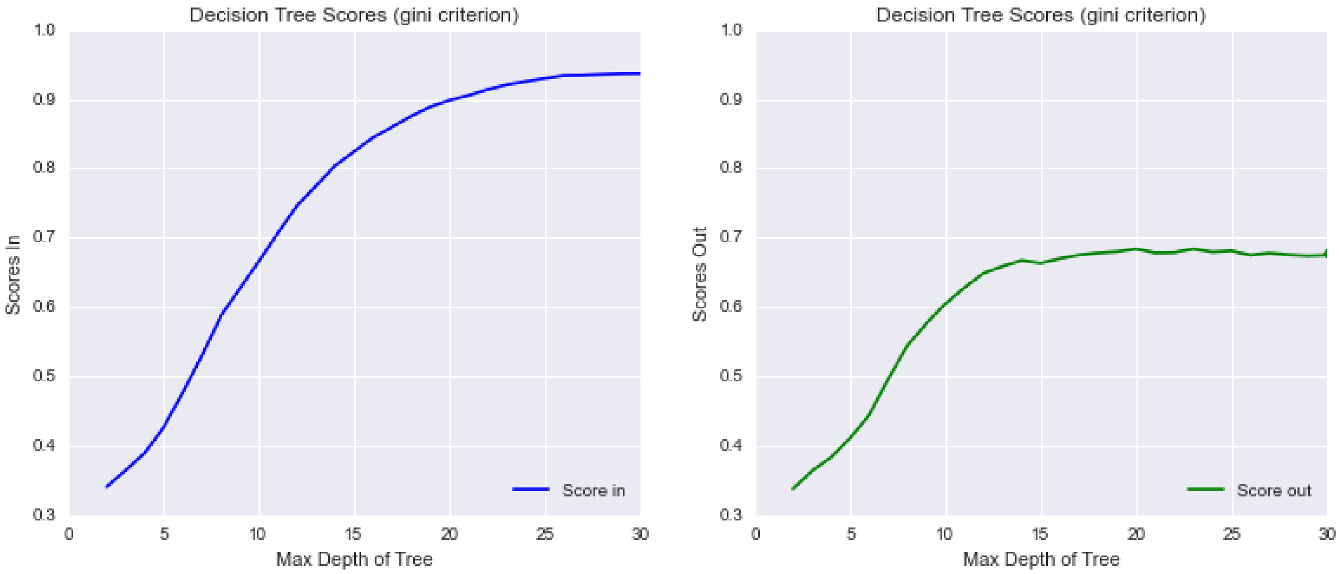
```
    - Rango_Edad
```

In [42]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                 , y_train_rango_todas, X_t
rain_todas
                                                 , y_test_rango_todas, X_te
st_todas)
                          for max_depth in range(profundidad,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                , y_train_rango_todas, X_tr
ain_todas
                                                 , y_test_rango_todas, X_te
st_todas)
                          for max_depth in range(profundidad,1, -1)])
```

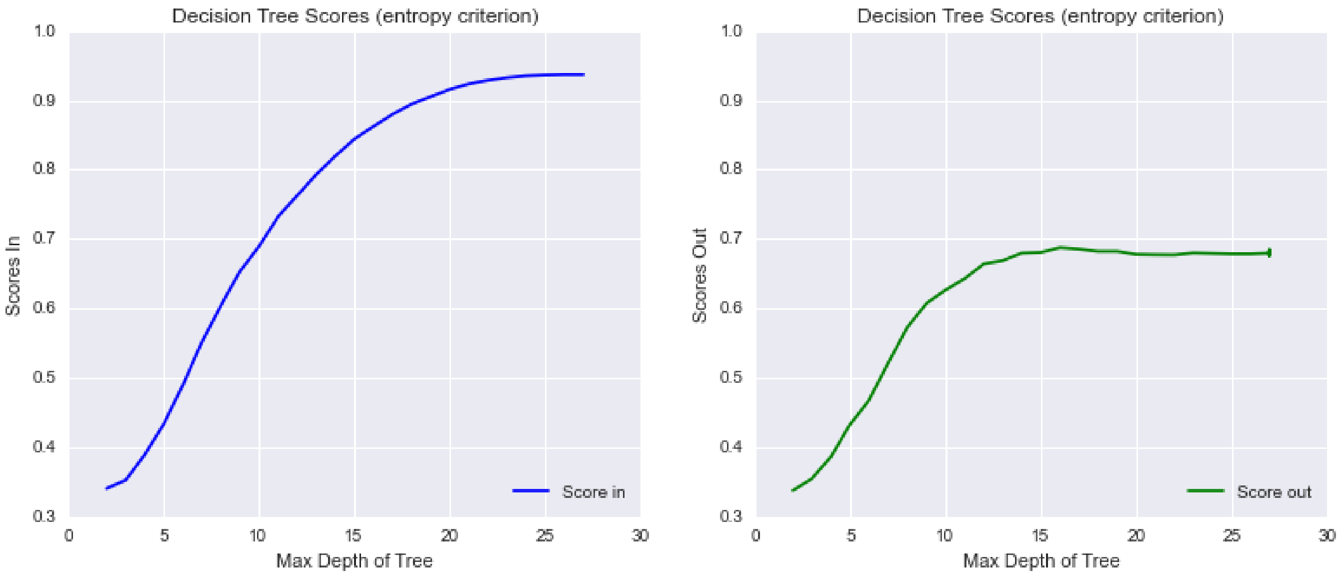
In [43]:

```
pinta_grafico_tree(gini_measures,0.3,1,0.3,1,'gini')
```



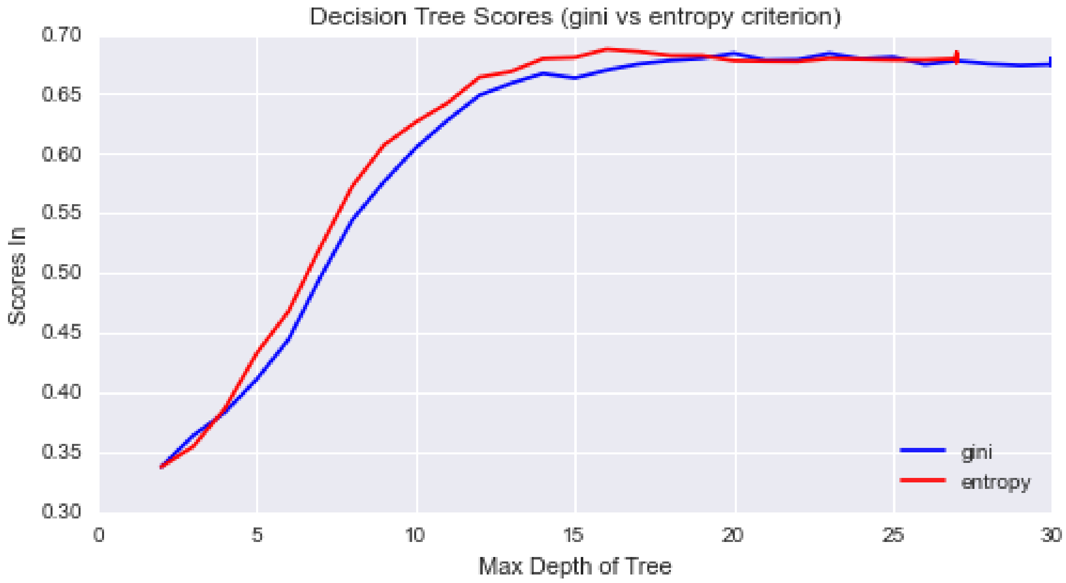
In [44]:

```
pinta_grafico_tree(entropy_measures,0.3,1,0.3,1,'entropy')
```



In [45]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entrop
y')
```



Algo mejor entropy con profundidad 16

In [270]:

```
model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=16)
model.fit(X_train_todas, y_train_rango_todas)
score = model.score(X_test_todas,y_test_rango_todas)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.684488297384
Exec_time:  0.00159788131714 s
```

In [271]:

```
Feature_Reduction.append("Todas")
Model.append("Decission_tree")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

2.-Varianza

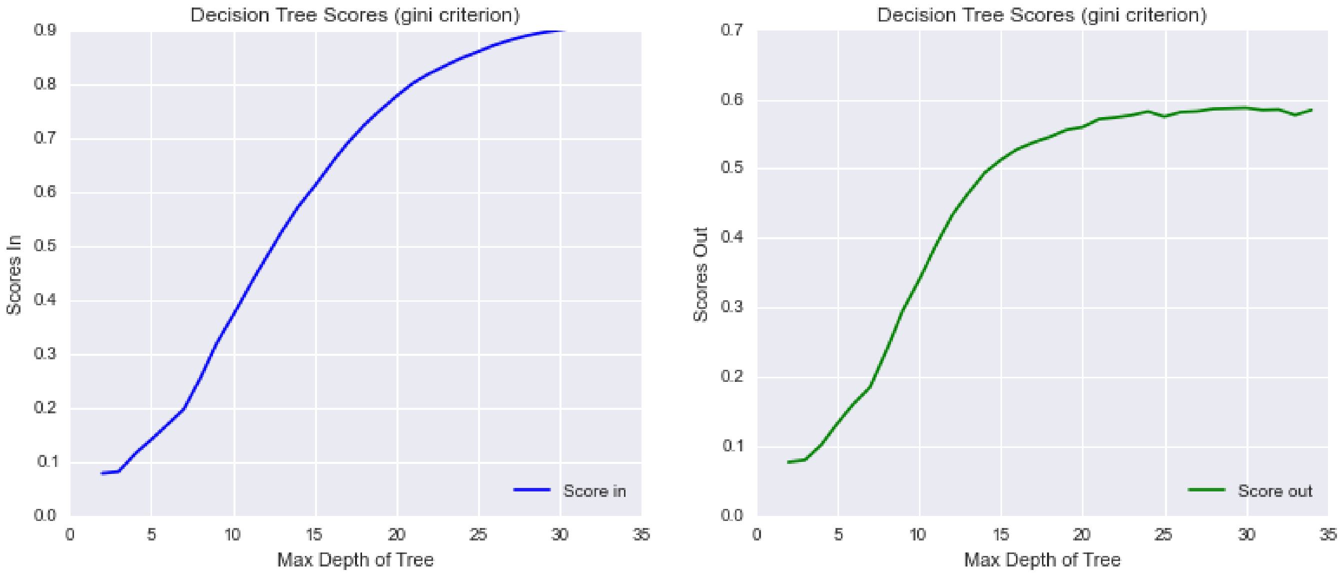
```
   - edad
```

In [48]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                 , y_train_varianza, X_trai
n_varianza
                                                 , y_test_varianza, X_tes
t_varianza)
                          for max_depth in range(34,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                , y_train_varianza, X_trai
n_varianza
                                                 , y_test_varianza, X_tes
t_varianza)
                          for max_depth in range(34,1, -1)])
```

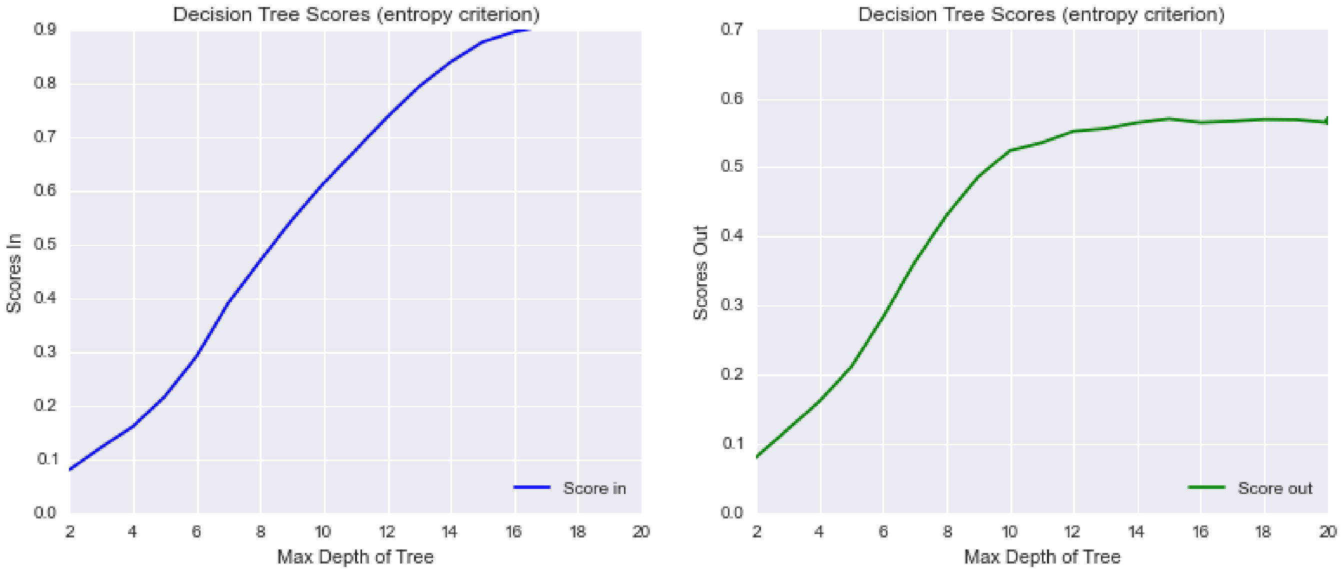
In [49]:

```
pinta_grafico_tree(gini_measures,0,0.9,0,0.7,'gini')
```



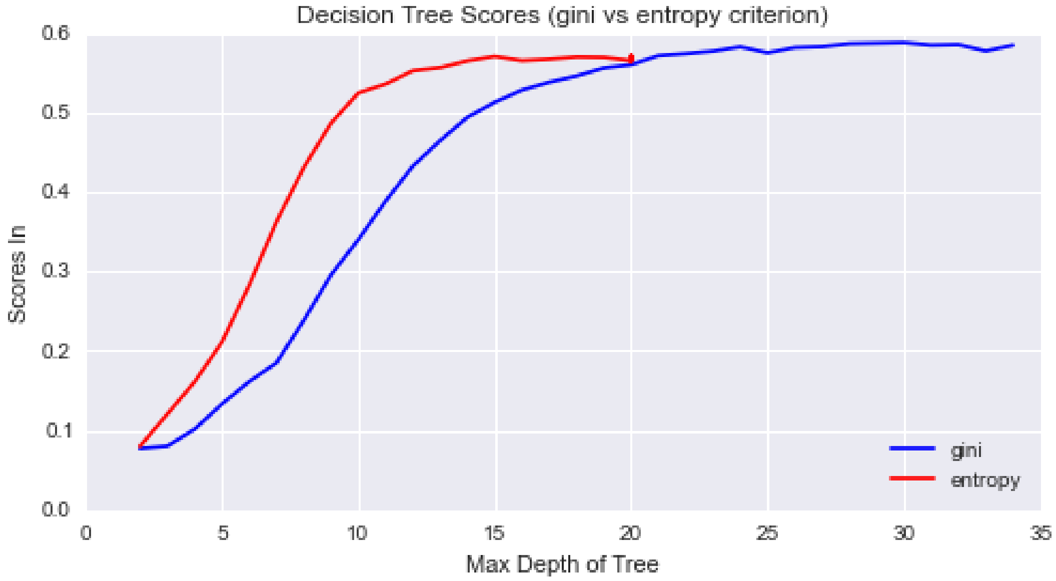
In [50]:

```
pinta_grafico_tree(entropy_measures,0,0.9,0,0.7,'entropy')
```



In [51]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entrop
y')
```



Mejor con entropía porque corta el árbol antes y profundidad 15

In [272]:

```
model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=15)
model.fit(X_train_varianza, y_train_varianza)
score = model.score(X_test_varianza,y_test_varianza)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.572051399725
Exec_time:  0.00211095809937 s
```

In [273]:

```
Feature_Reduction.append("Varianza")
Model.append("Decission_tree")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

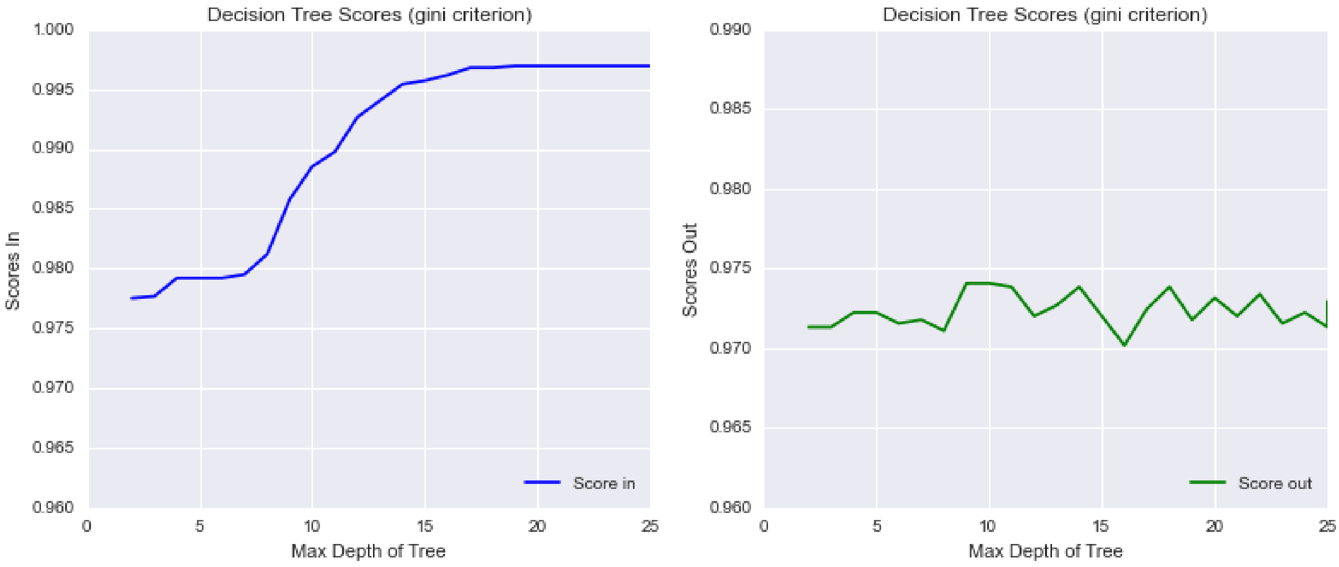
- mayor_edad

In [54]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                    , y_train_mayor_edad_V,
X_train_varianza
                                                    , y_test_mayor_edad_V, X_t
est_varianza)
                          for max_depth in range(34,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                   , y_train_mayor_edad_V, X_t
rain_varianza
                                                    , y_test_mayor_edad_V, X_t
est_varianza)
                          for max_depth in range(34,1, -1)])
```

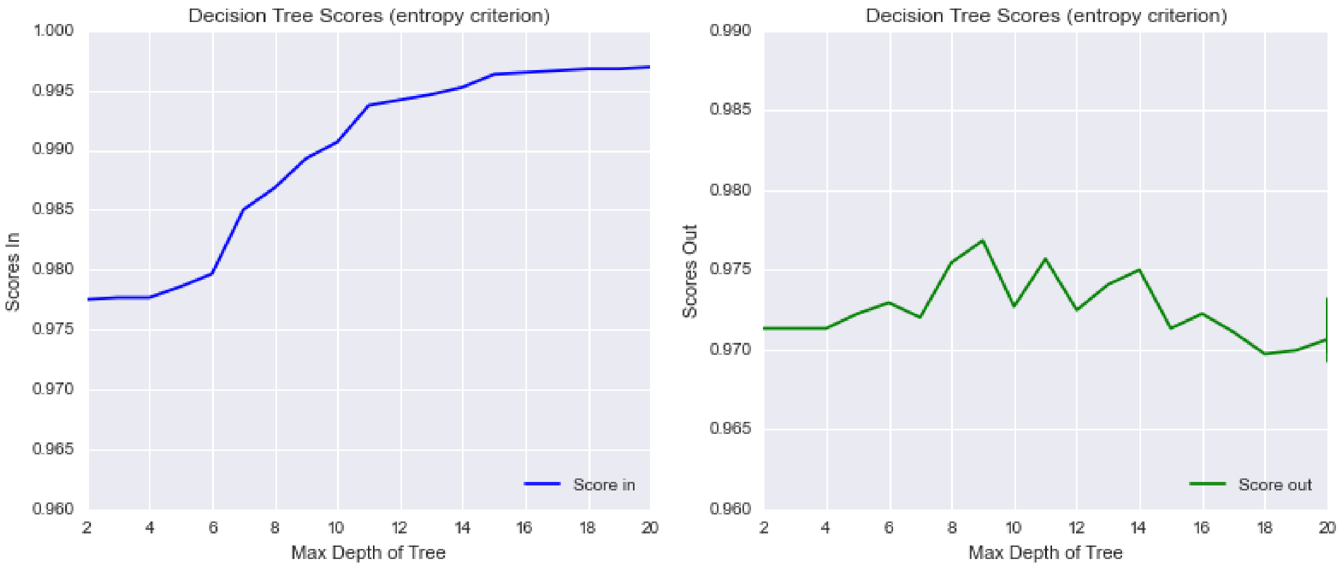
In [55]:

```
pinta_grafico_tree(gini_measures,0.96,1,0.96,0.99,'gini')
```



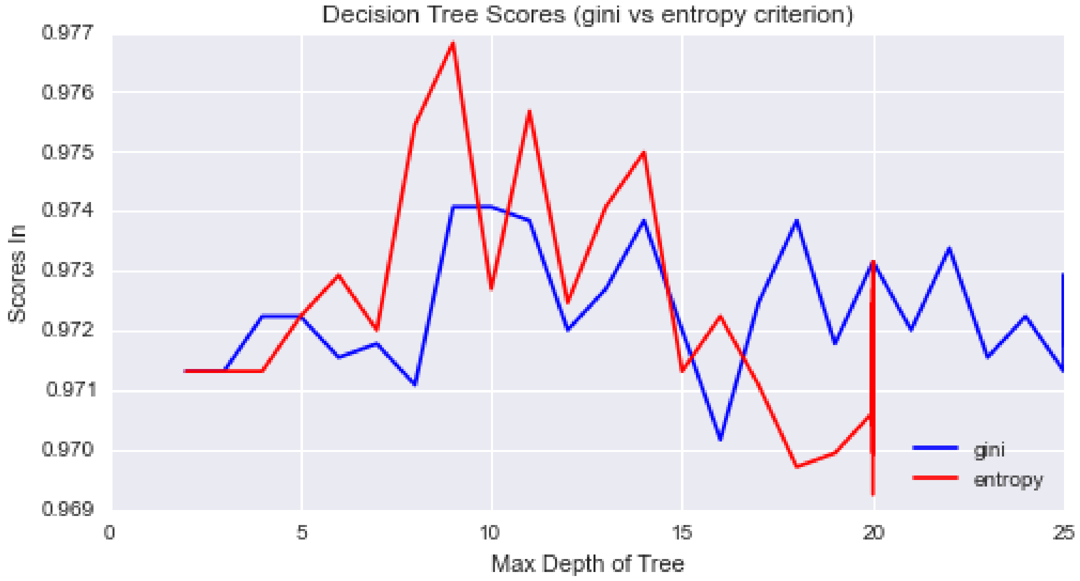
In [56]:

```
pinta_grafico_tree(entropy_measures,0.96,1,0.96,0.99,'entropy')
```



In [57]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entropy')
```



usamos entropy con profundudad 9

In [274]:

```
model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=9)
model.fit(X_train_varianza, y_train_mayor_edad_V)
score = model.score(X_test_varianza,y_test_mayor_edad_V)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.977053694355
Exec_time:  0.00143504142761 s
```

In [275]:

```
Feature_Reduction.append("Varianza")
Model.append("Decission_tree")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

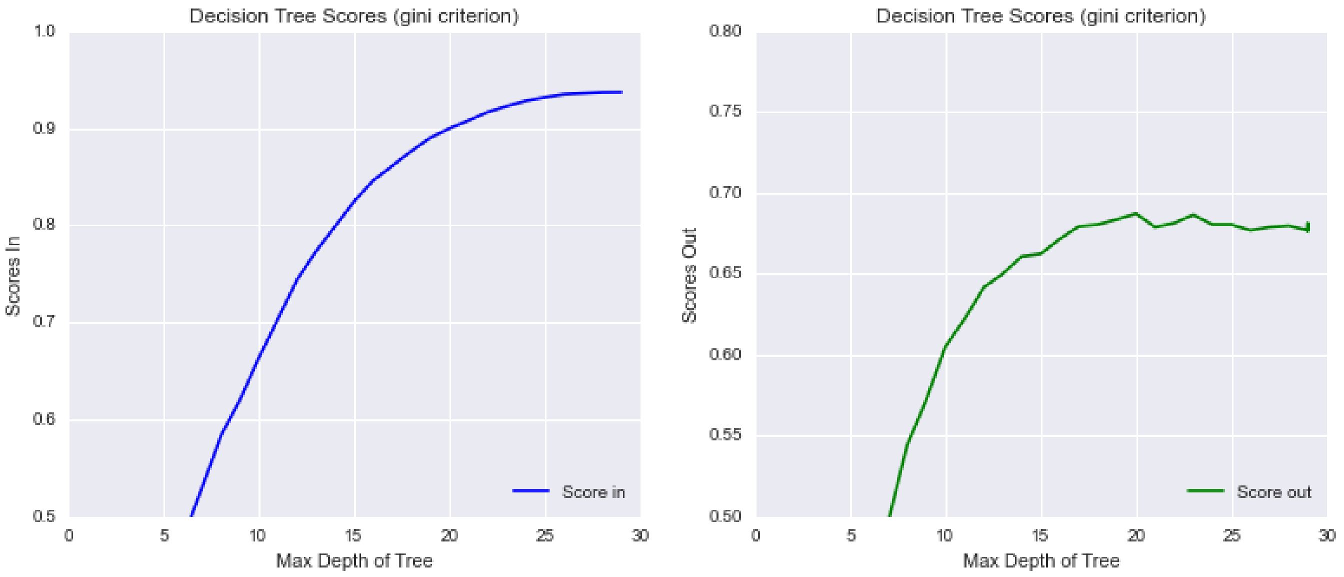
- rango_edad

In [60]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                 , y_train_rango_V, X_trai
n_varianza
                                                 , y_test_rango_V, X_test_v
arianza)
                          for max_depth in range(34,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                 , y_train_rango_V, X_trai
n_varianza
                                                 , y_test_rango_V, X_test_v
arianza)
                          for max_depth in range(34,1, -1)])
```

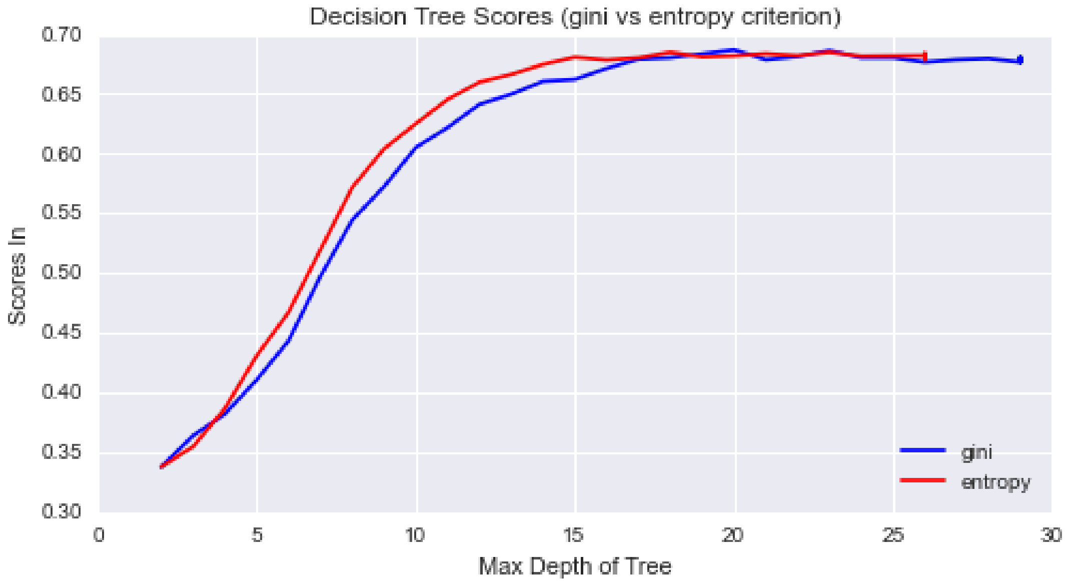
In [61]:

```
pinta_grafico_tree(gini_measures,0.5,1,0.5,0.8,'gini')
```



In [62]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entrop
y')
```



Usamos entropy con profundidad 15

In [276]:

```
model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=15)
model.fit(X_train_varianza, y_train_rango_V)
score = model.score(X_test_varianza,y_test_rango_V)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.673703533731
Exec_time:  0.00203585624695 s
```

In [277]:

```
Feature_Reduction.append("Varianza")
Model.append("Decission_tree")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

3.-Extra Tree

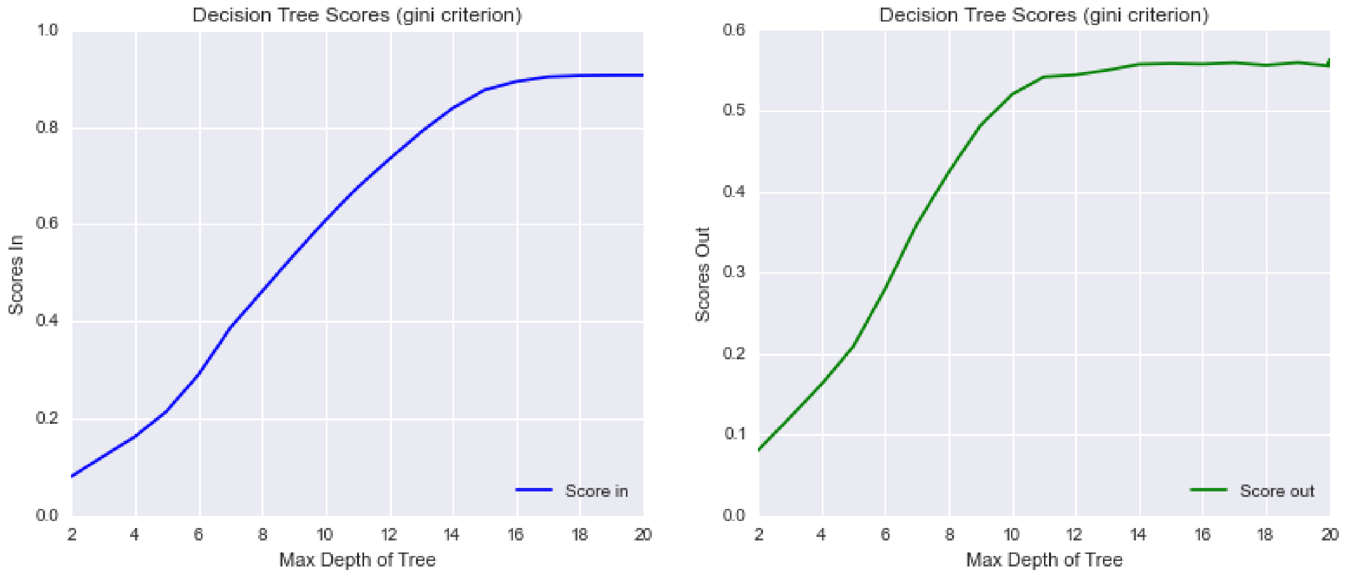
```
- edad
```

In [65]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                 , y_train_ET, X_train_ET
                                                 , y_test_ET, X_test_ET)
                          for max_depth in range(34,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                 , y_train_ET, X_train_ET
                                                 , y_test_ET, X_test_ET)
                          for max_depth in range(34,1, -1)])
```

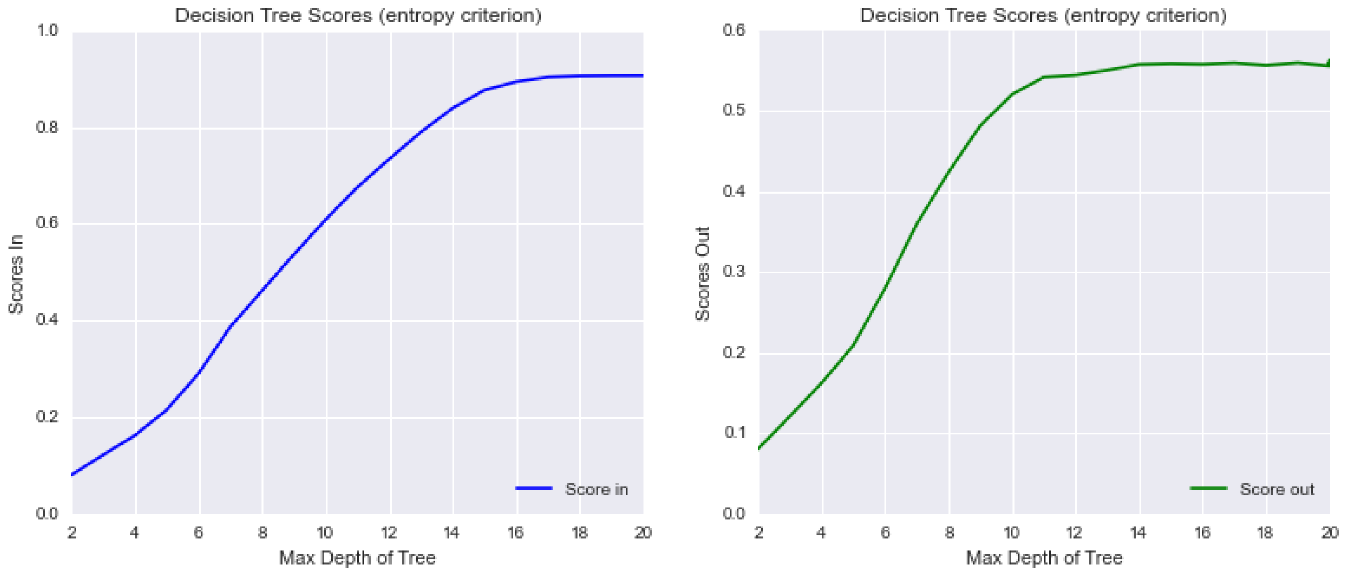
In [66]:

```
pinta_grafico_tree(entropy_measures,0,1,0,0.6,'gini')
```



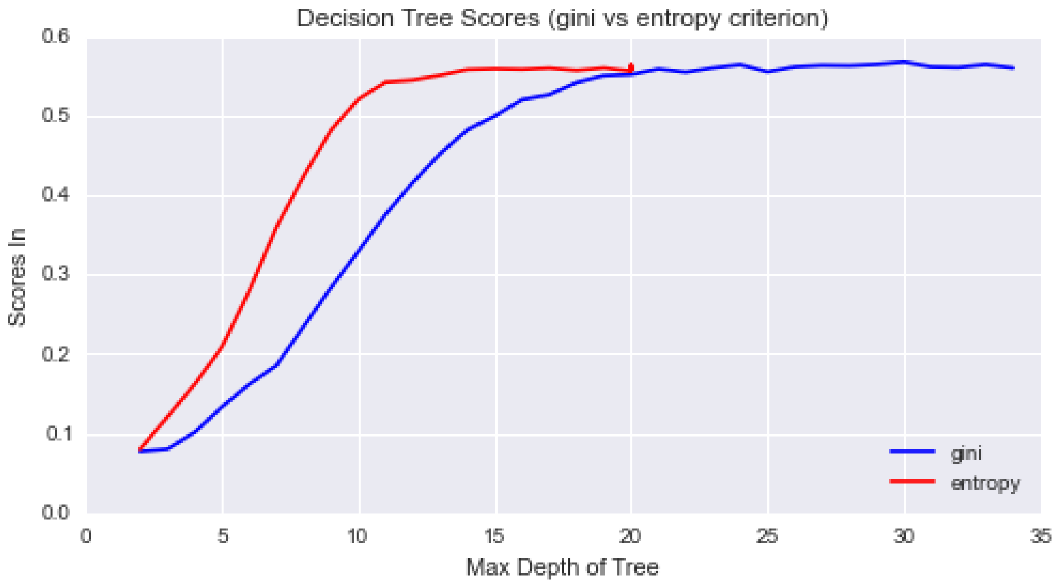
In [67]:

```
pinta_grafico_tree(entropy_measures,0,1,0,0.6,'entropy')
```



In [68]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entropy')
```



Como criterio entropy y cortamos en 15

In [278]:

```
model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=15)
model.fit(X_train_ET, y_train_ET)
score = model.score(X_test_ET,y_test_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.554382744378
Exec_time:  0.00128293037415 s
```

In [279]:

```
Feature_Reduction.append("Extra_tree")
Model.append("Decission_tree")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

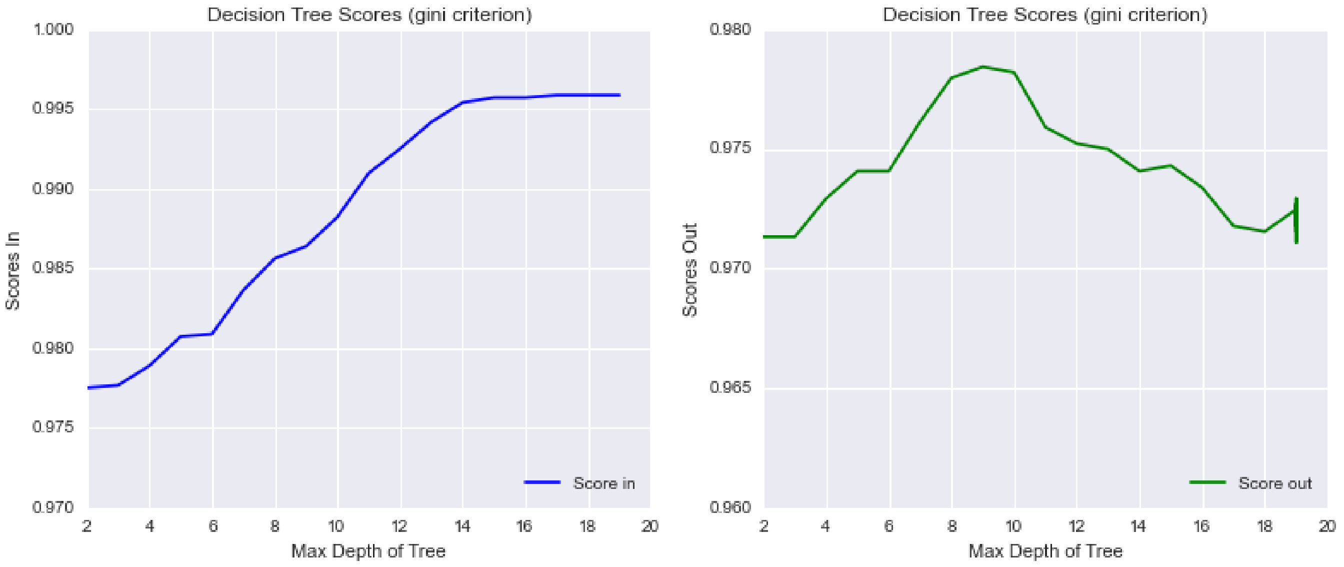
```
    - mayor_edad
```

In [71]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                  , y_train_mayor_edad_ET,
X_train_ET
                                                  , y_test_mayor_edad_ET,
X_test_ET)
                          for max_depth in range(34,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                  , y_train_mayor_edad_ET,
X_train_ET
                                                  , y_test_mayor_edad_ET,
X_test_ET)
                          for max_depth in range(34,1, -1)])
```

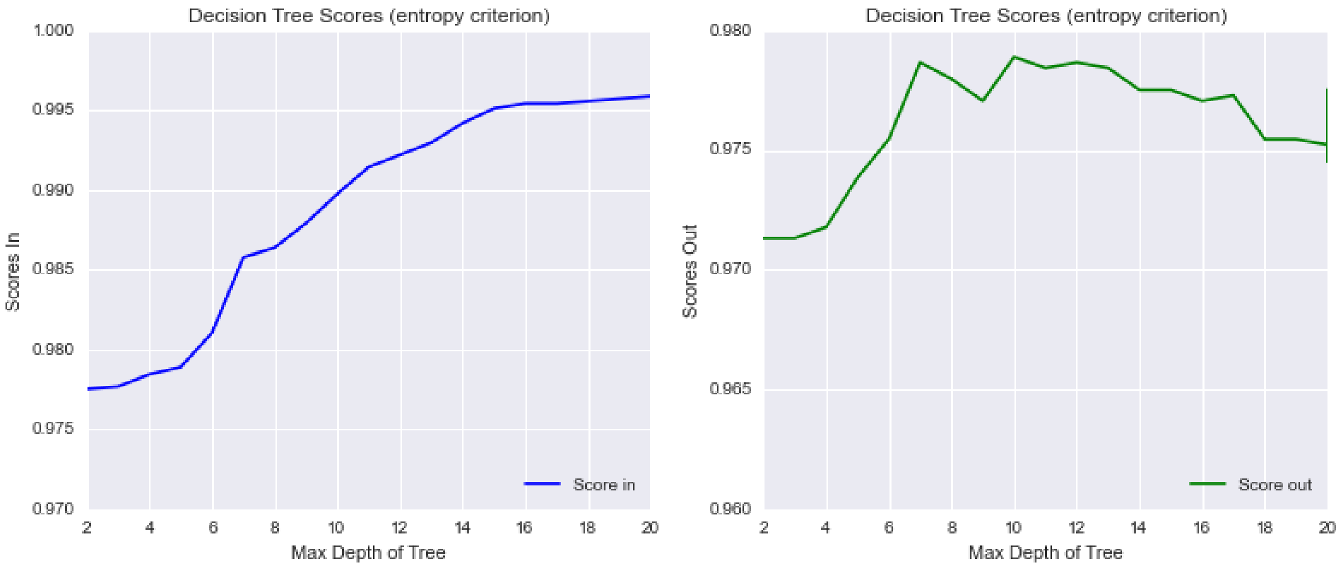
In [72]:

```
pinta_grafico_tree(gini_measures,0.97,1,0.96,0.98,'gini')
```



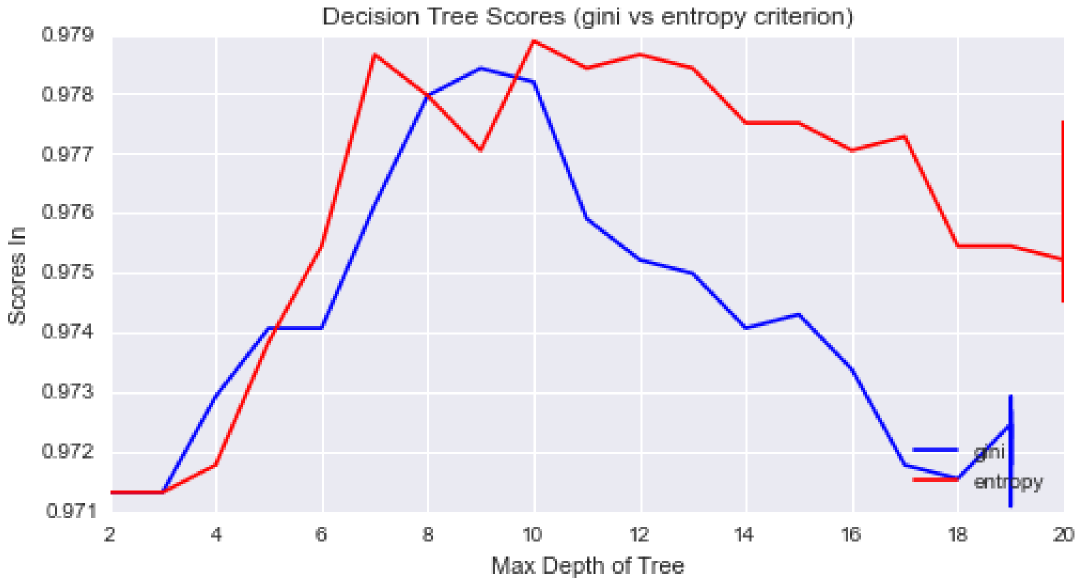
In [73]:

```
pinta_grafico_tree(entropy_measures,0.97,1,0.96,0.98,'entropy')
```



In [74]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entrop
y')
```



Como criterio entropy y cortamos en 10

In [280]:

```
model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=10)
model.fit(X_train_ET, y_train_mayor_edad_ET)
score = model.score(X_test_ET,y_test_mayor_edad_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.977971546581
Exec_time:  0.000900983810425 s
```

In [281]:

```
Feature_Reduction.append("Extra_tree")
Model.append("Decission_tree")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

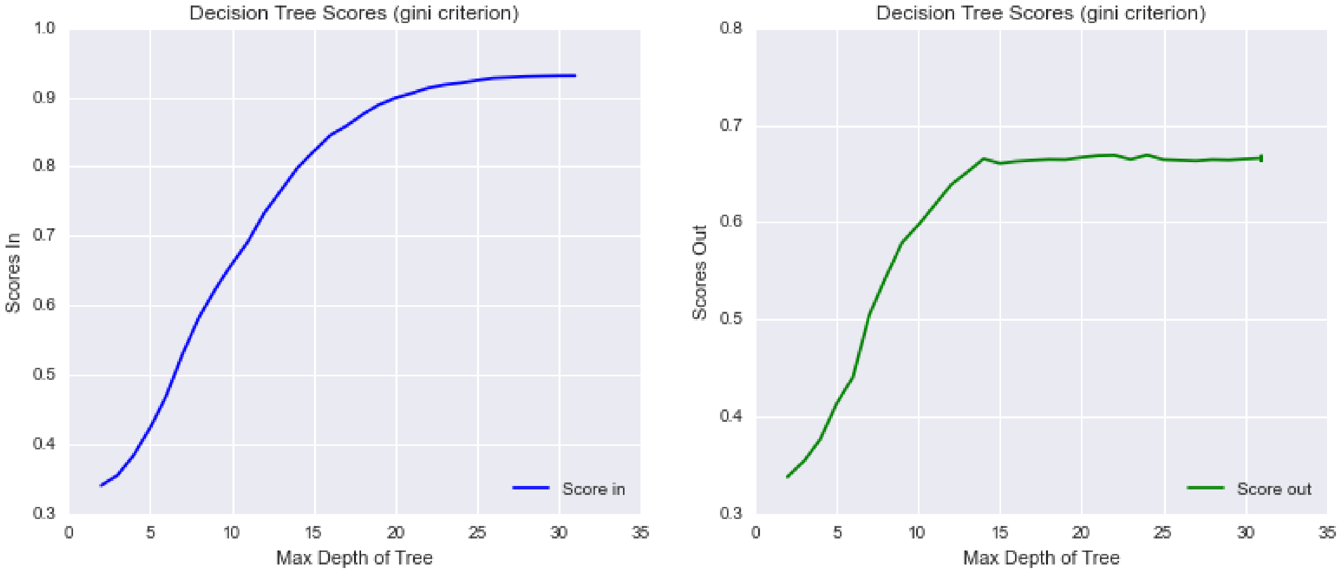
- rango_edad

In [77]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                 , y_train_rango_ET, X_trai
n_ET
                                                 , y_test_rango_ET, X_tes
t_ET)
                          for max_depth in range(34,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                 , y_train_rango_ET, X_trai
n_ET
                                                 , y_test_rango_ET, X_tes
t_ET)
                          for max_depth in range(34,1, -1)])
```

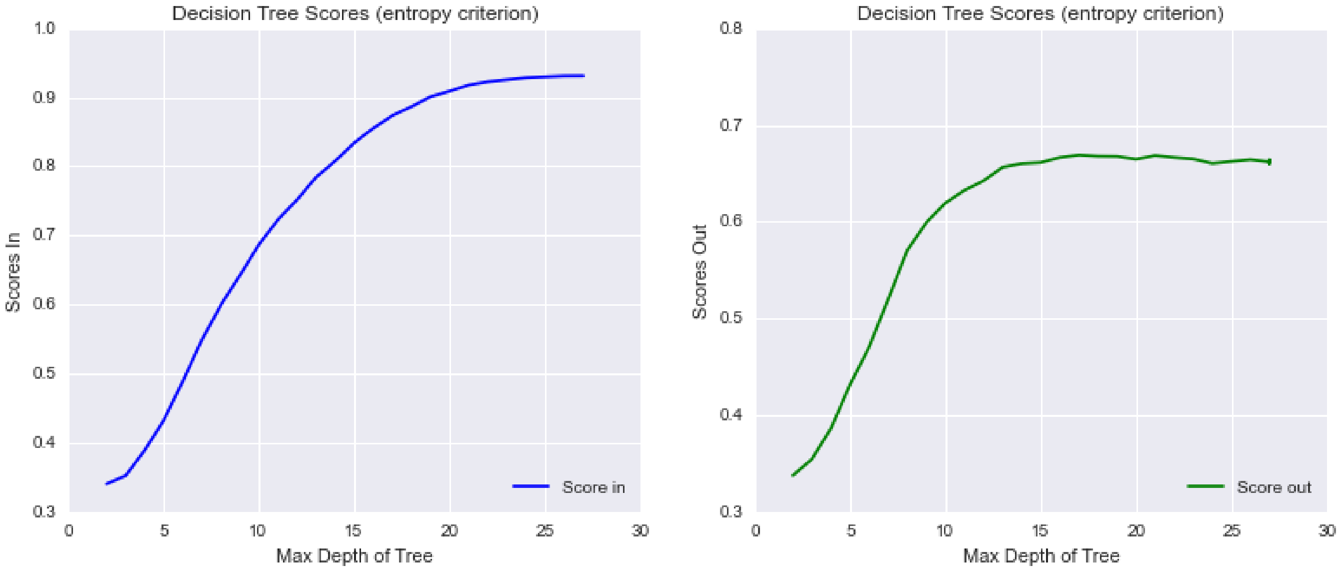
In [78]:

```
pinta_grafico_tree(gini_measures,0.3,1,0.3,0.8,'gini')
```



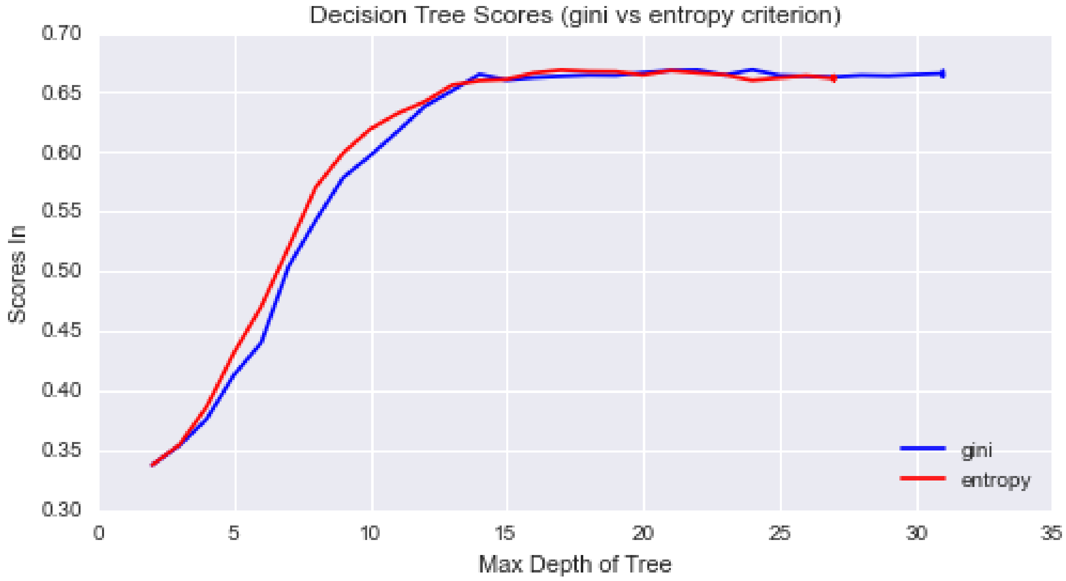
In [79]:

```
pinta_grafico_tree(entropy_measures,0.3,1,0.3,0.8,'entropy')
```



In [80]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entrop
y')
```



criterio gini y cortamos en 14

In [282]:

```
model = tree.DecisionTreeClassifier(criterion='gini', max_depth=14)
model.fit(X_train_ET, y_train_rango_ET)
score = model.score(X_test_ET,y_test_rango_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.66475447453
Exec_time:  0.00115895271301 s
```

In [283]:

```
Feature_Reduction.append("Extra_tree")
Model.append("Decission_tree")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

4.-PCA

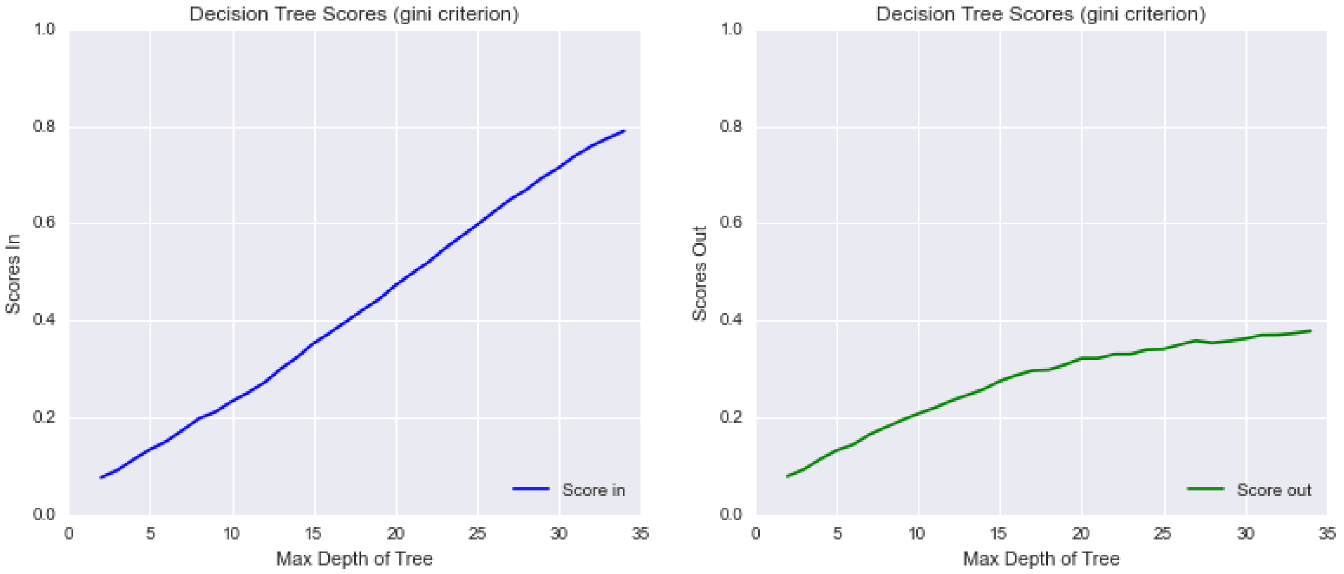
- edad

In [83]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                , y_train_PCA, X_train_PCA
                                                , y_test_PCA, X_test_PCA)
                          for max_depth in range(34,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                , y_train_PCA, X_train_PCA
                                                , y_test_PCA, X_test_PCA)
                          for max_depth in range(34,1, -1)])
```

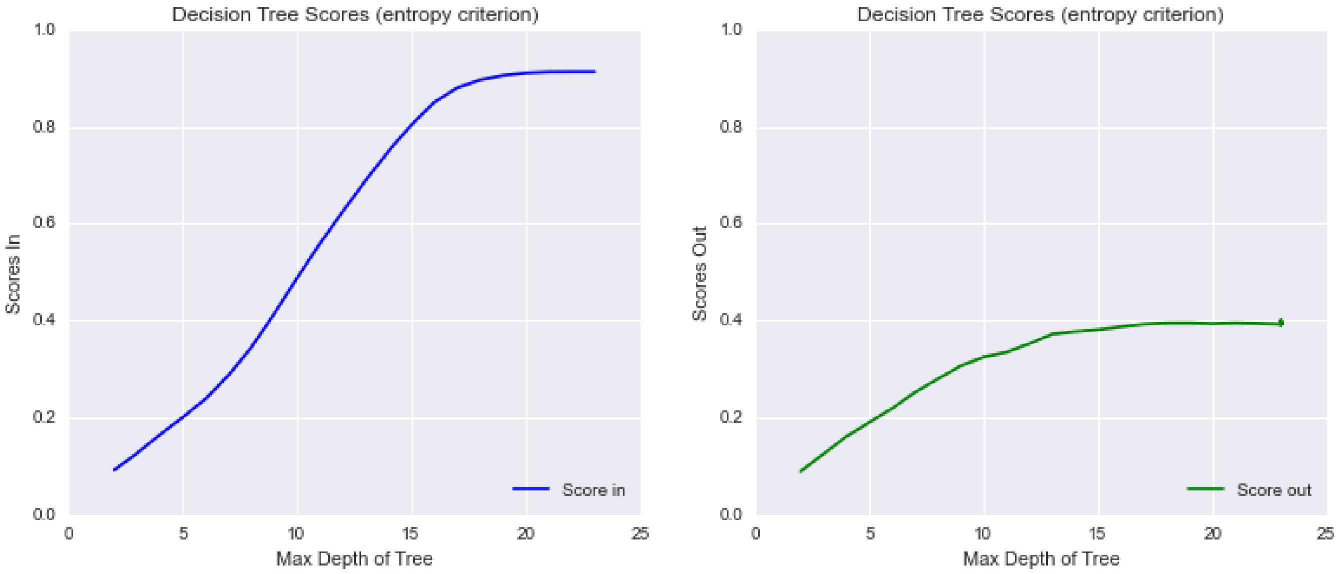
In [84]:

```
pinta_grafico_tree(gini_measures,0,1,0,1,'gini')
```



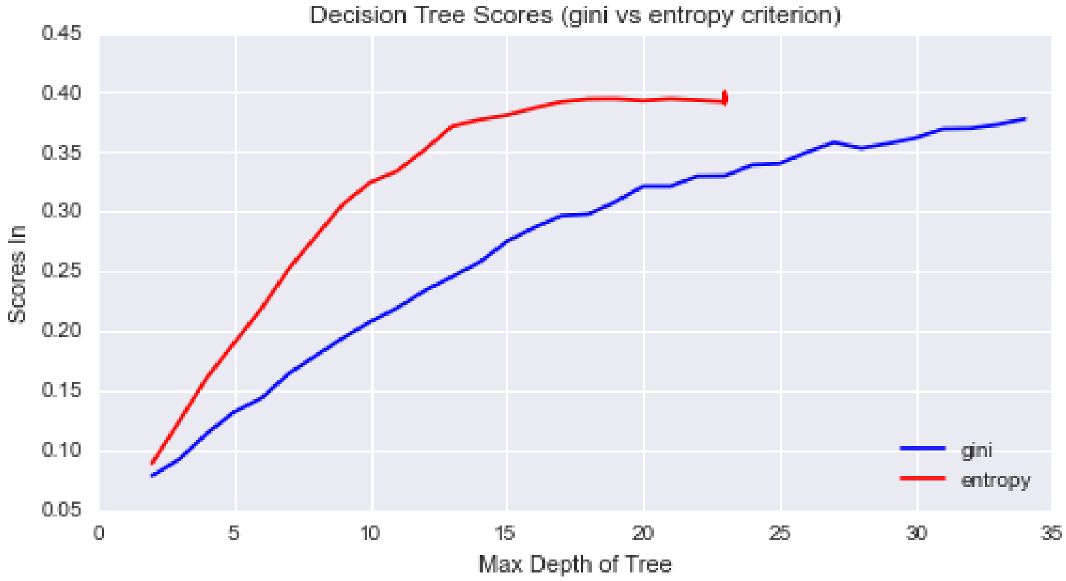
In [85]:

```
pinta_grafico_tree(entropy_measures,0,1,0,1,'entropy')
```



In [86]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entrop
y')
```



Usaremos como criterio entropy y la profundidad 18

In [284]:

```
model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=18)
model.fit(X_train_PCA, y_train_PCA)
score = model.score(X_test_PCA,y_test_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.39536484626
Exec_time:  0.00375390052795 s
```

In [285]:

```
Feature_Reduction.append("PCA")
Model.append("Decission_tree")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

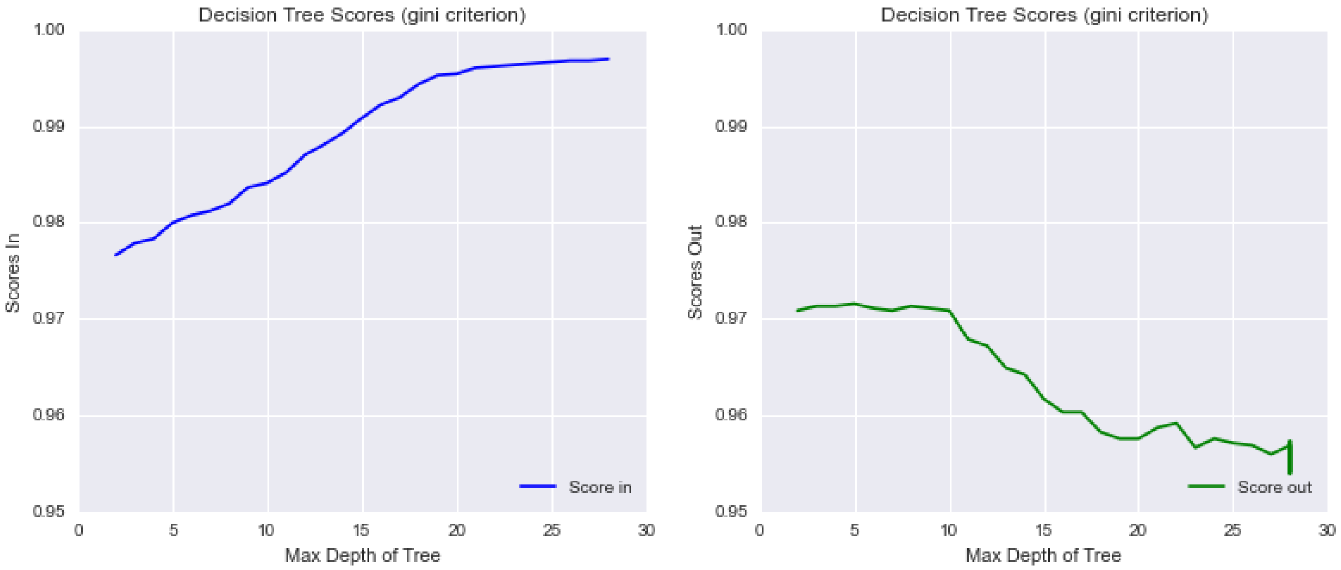
```
    - mayor_edad
```

In [89]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                 , y_train_mayor_edad_PCA,
X_train_PCA
                                                 , y_test_mayor_edad_PCA,
X_test_PCA)
                          for max_depth in range(34,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                 , y_train_mayor_edad_PCA,
X_train_PCA
                                                 , y_test_mayor_edad_PCA,
X_test_PCA)
                          for max_depth in range(34,1, -1)])
```

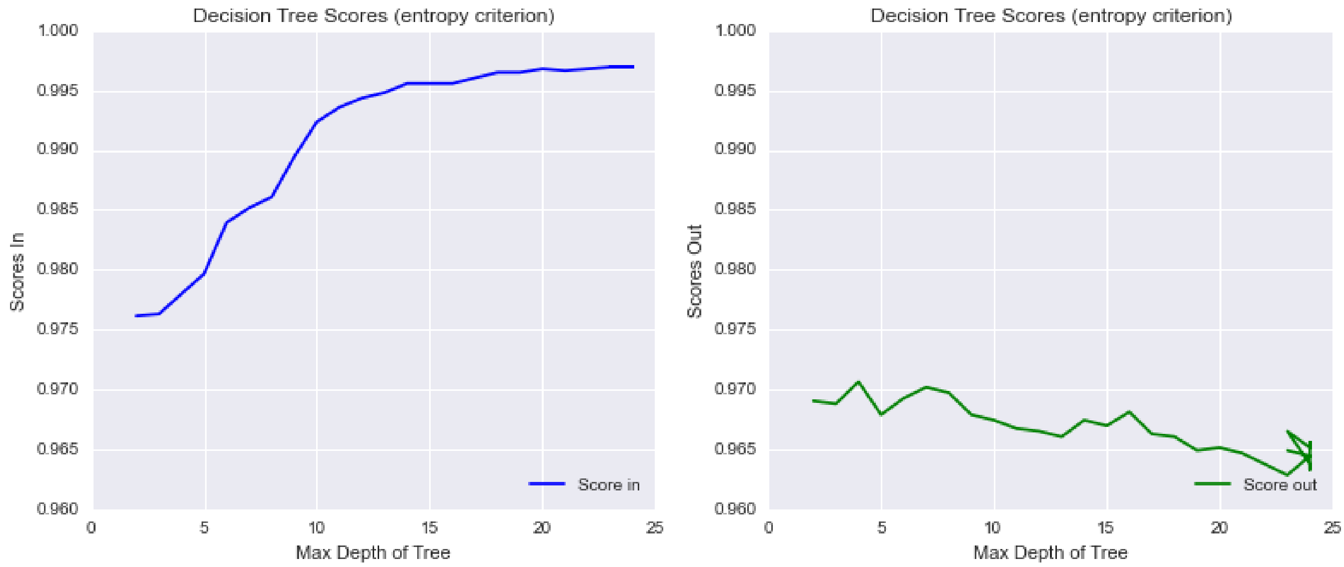
In [90]:

```
pinta_grafico_tree(gini_measures,0.95,1,0.95,1,'gini')
```



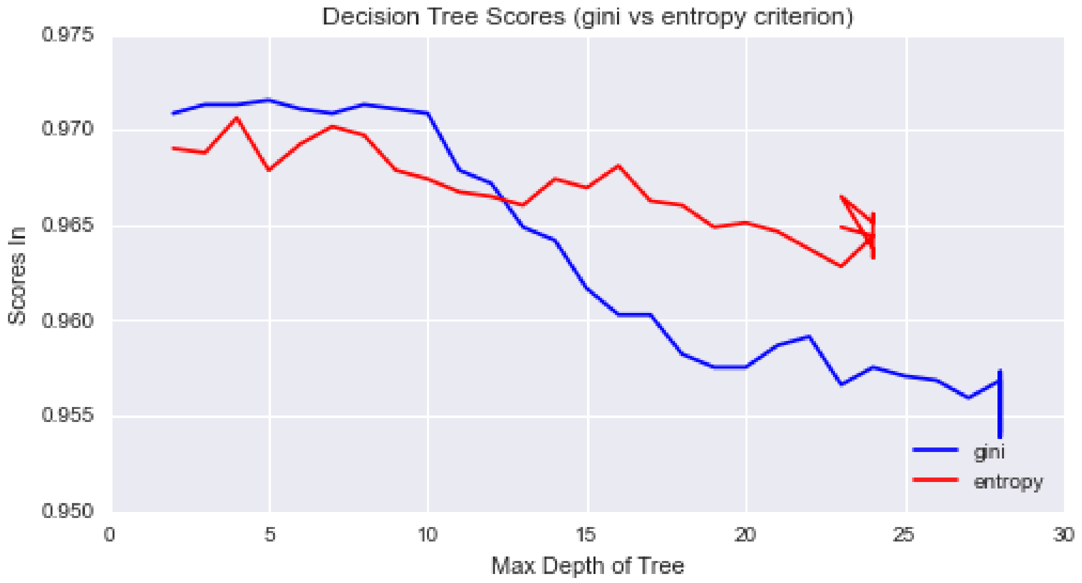
In [91]:

```
pinta_grafico_tree(entropy_measures,0.96,1,0.96,1,'entropy')
```



In [92]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entropy')
```



usaremos el criterio gini y la profundidad 5

In [286]:

```
model = tree.DecisionTreeClassifier(criterion='gini', max_depth=5)
model.fit(X_train_PCA, y_train_mayor_edad_PCA)
score = model.score(X_test_PCA,y_test_mayor_edad_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.971776044057
Exec_time:  0.00200700759888 s
```

In [287]:

```
Feature_Reduction.append("PCA")
Model.append("Decission_tree")
Target.append("mayor_edad")
Final_Score.append(model.score(X_test_PCA,y_test_mayor_edad_PCA))
Parameters.append(model)
Exec_time.append(total_time)
```

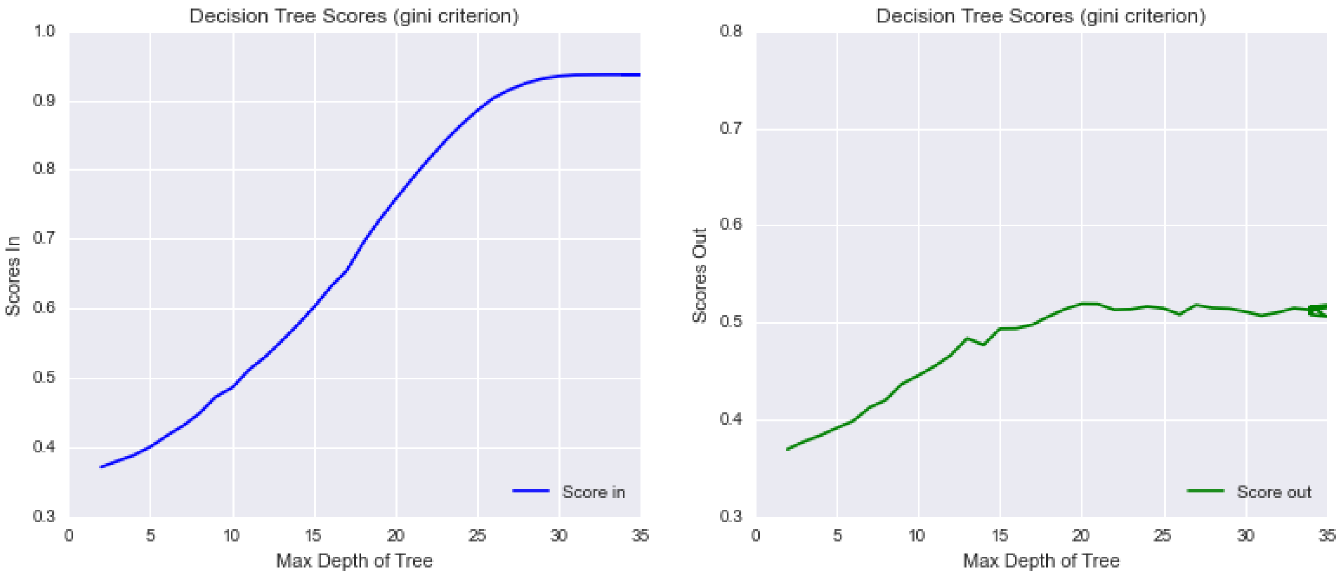
- rango_edad

In [95]:

```
#Con gini
gini_measures = np.array([getDecisionTreeMesures('gini', max_depth
                                                  , y_train_rango_PCA, X_tra
in_PCA
                                                  , y_test_rango_PCA, X_tes
t_PCA)
                          for max_depth in range(50,1, -1)])
#con entropy
entropy_measures = np.array([getDecisionTreeMesures('entropy', max_depth
                                                  , y_train_rango_PCA, X_tra
in_PCA
                                                  , y_test_rango_PCA, X_tes
t_PCA)
                          for max_depth in range(50,1, -1)])
```

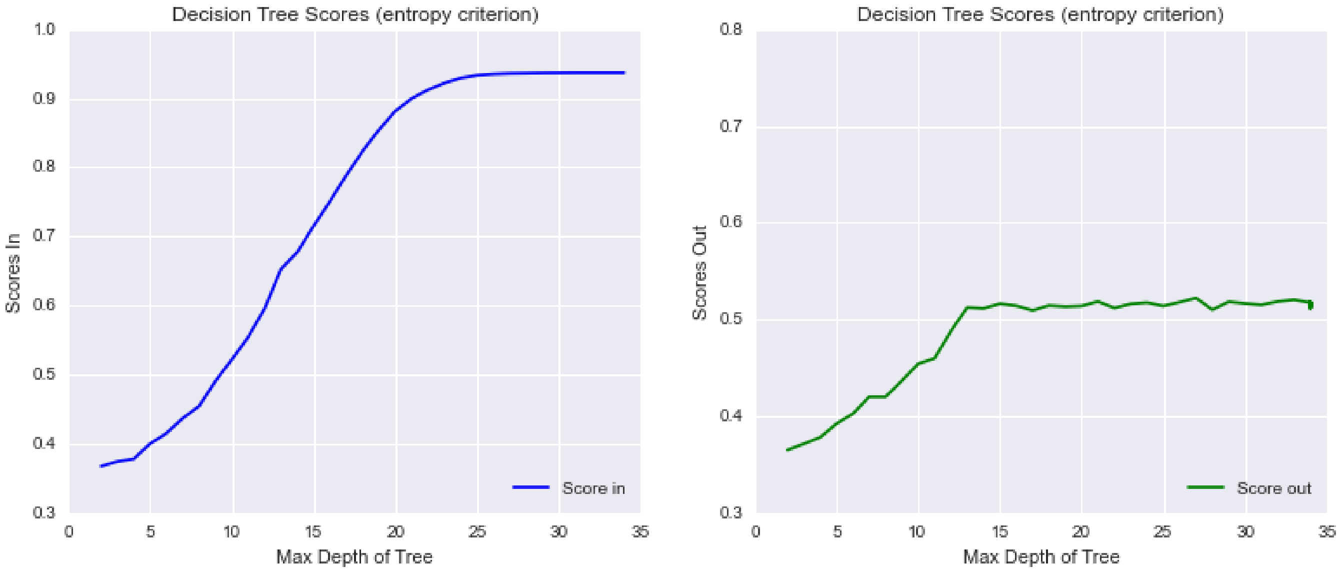
In [96]:

```
pinta_grafico_tree(gini_measures,0.3,1,0.3,0.8,'gini')
```



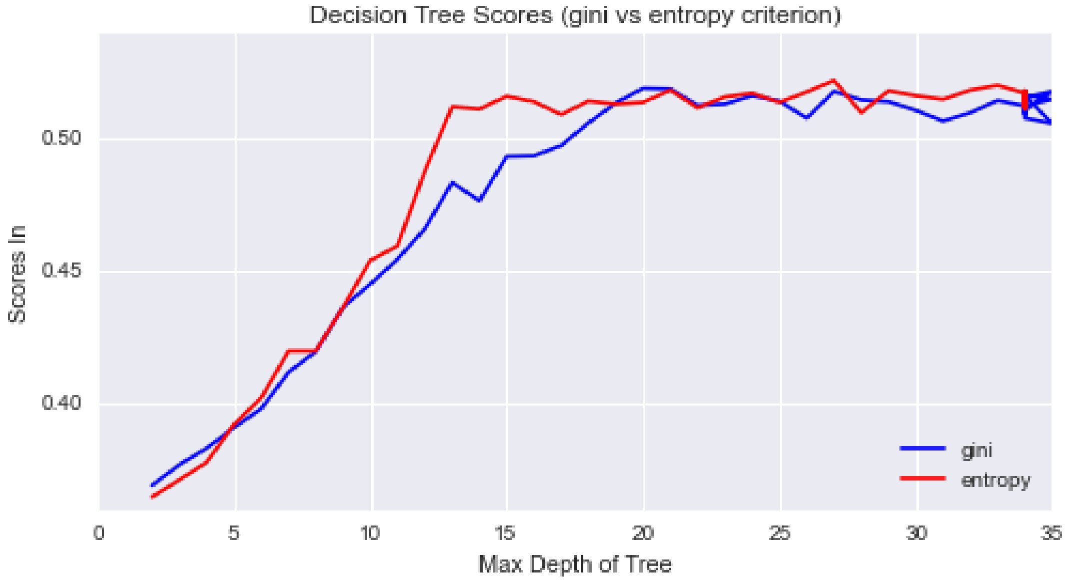
In [97]:

```
pinta_grafico_tree(entropy_measures,0.3,1,0.3,0.8,'entropy')
```



In [98]:

```
pinta_grafico_tree_compara(gini_measures, entropy_measures,'gini vs entrop
y')
```



Vamos a selecciona entropy y profundidad 15

In [288]:

```
model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=15)
model.fit(X_train_PCA, y_train_rango_PCA)
score = model.score(X_test_PCA,y_test_rango_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.518586507572
Exec_time:  0.00411796569824 s
```

In [289]:

```
Feature_Reduction.append("PCA")
Model.append("Decission_tree")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

7.-Random Forest

In [290]:

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

In [291]:

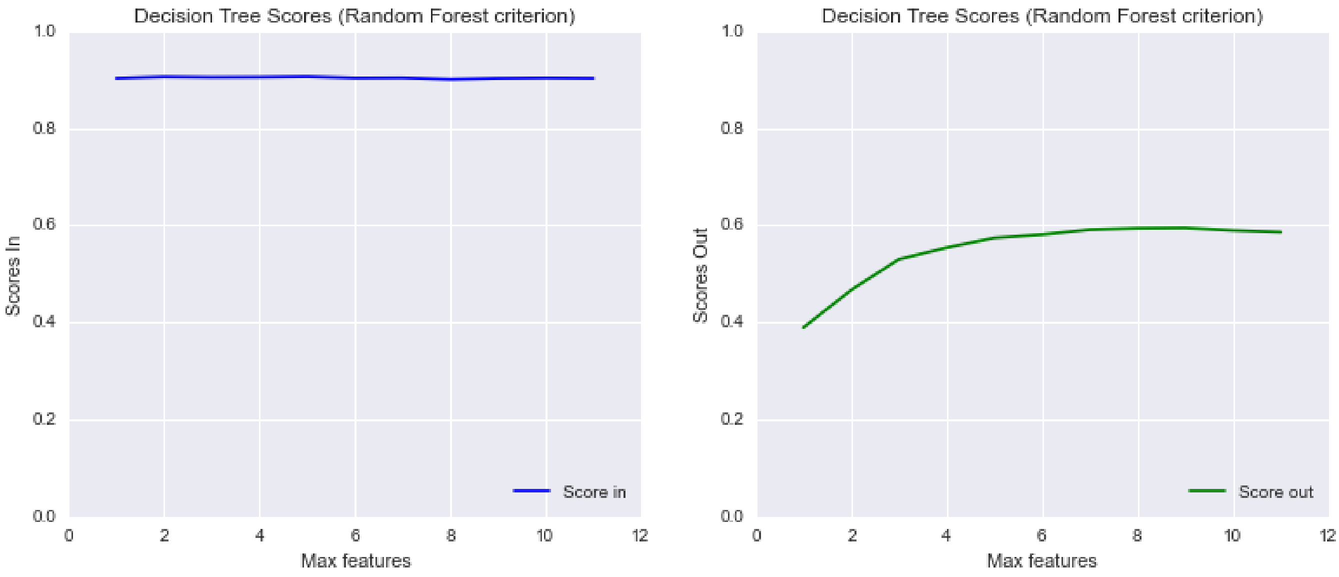
```
def getRFMesures(min_samples_leaf = 1, num_estimators = 10, max_features =
1,
                           train_labels = None, train_data = None,
                           test_label = None, test_data = None, variable =
'max_features'):
    model = RandomForestClassifier(n_estimators = num_estimators,
                                   max_features = max_features,
                                   min_samples_leaf= min_samples_leaf)
    model.fit(train_data, train_labels)

    if variable == 'max_features':
        usa = model.max_features
    elif variable == 'min_samples_leaf':
        usa = model.min_samples_leaf
    elif variable == 'num_estimators':
        usa = model.n_estimators
    else:
        usa = model.max_features
    return [usa,
            model.score(train_data, train_labels), #E_in
            model.score(test_data, test_label)] #E_out
```

In [296]:

```
RF_measures = np.array([getRFMesures( 1, 10 ,max_features,
                                     y_train_varianza, X_train_varianza,
                                     y_test_varianza, X_test_varianza,'num
estimators')
                     for max_features in range(1, 12, 1)])
```

In [297]:

```
pinta_grafico_tree(RF_measures,0,1,0,1,'Random Forest','Max features')
```



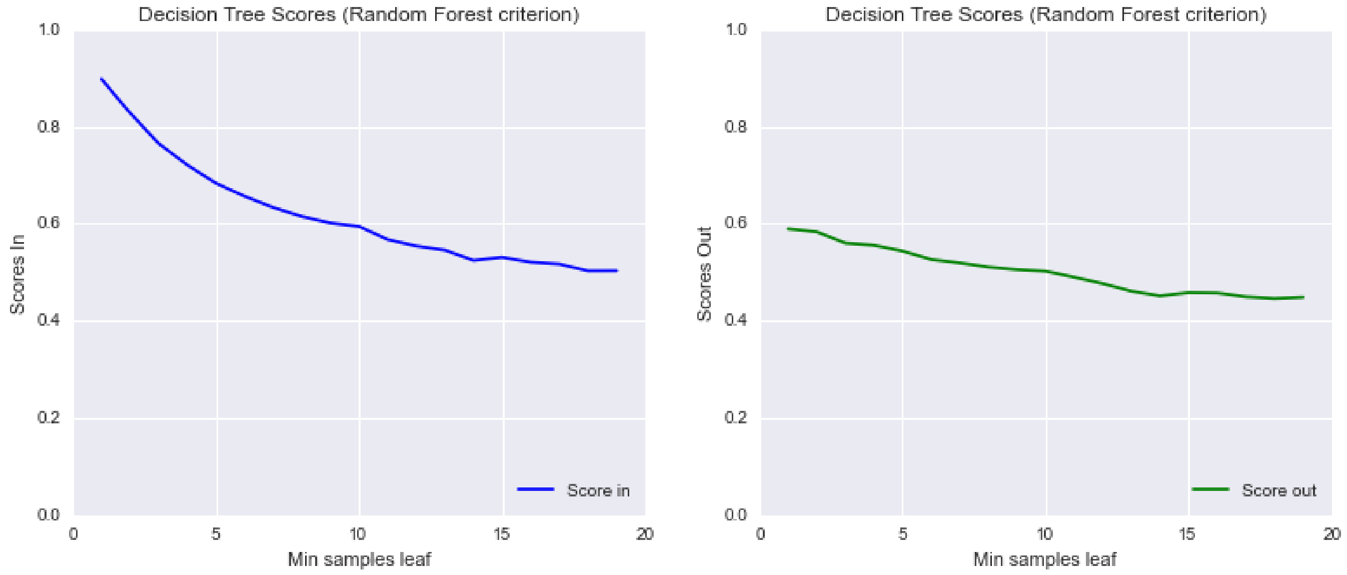
Tomamos 8 max features

In [105]:

```
#Número de características
RF_measures = np.array([getRFMesures( min_samples_leaf, 8, 8,
                                     y_train_varianza, X_train_varianza,
                                     y_test_varianza, X_test_varianza,'mi
n_samples_leaf')
                     for min_samples_leaf in range(1, 20, 1)])
```

In [106]:

```
pinta_grafico_tree(RF_measures,0,1,0,1,'Random Forest','Min samples leaf')
```



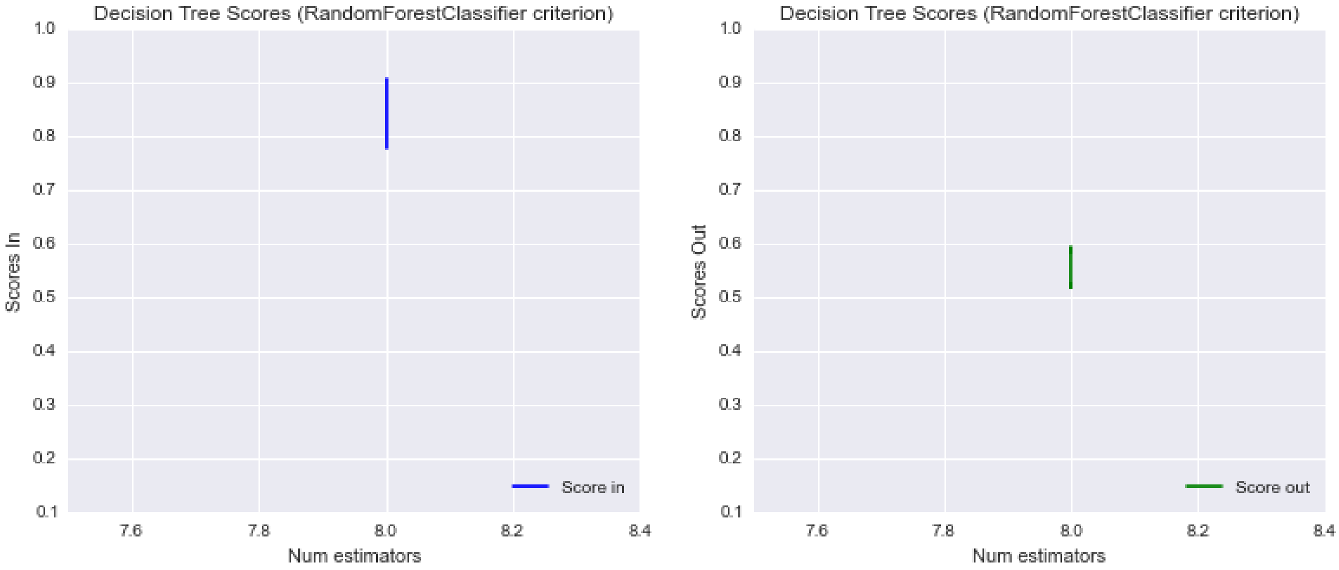
Decrece el resultado así que lo dejamos en 1

In [107]:

```
#Número de árboles
RF_measures = np.array([getRFMesures( 1, num_estimators,8,
                                     y_train_varianza, X_train_varianza,
                                     y_test_varianza, X_test_varianza,'num
estimators')
                     for num_estimators in range(1, 12, 1)])
```

In [108]:

```
pinta_grafico_tree(RF_measures,0.1,1,0.1,1,'RandomForestClassifier','Num es
timators')
```



El mismo que max features, así que dejamos el que hay por defecto

El número de árboles por defecto es 10, dejamos 10

```
* num_estimators = 8
* min_samples_leaf = 1
* max_features = 8
```

1.- Todas

```
- edad
```

In [299]:

```
model = RandomForestClassifier(n_estimators = 8,
                                   max_features = 8,
                                   min_samples_leaf= 1)
model.fit(X_train_todas, y_train_todas)
score = model.score(X_test_todas,y_test_todas)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.584442404773
Exec_time:  0.0042998790741 s
```

In [300]:

```
Feature_Reduction.append("Todas")
Model.append("Random_forest")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- mayor_edad

In [301]:

```
model = RandomForestClassifier(n_estimators = 8,
                                  max_features = 8,
                                  min_samples_leaf= 1)
model.fit(X_train_todas, y_train_mayor_edad_todas)
score = model.score(X_test_todas,y_test_mayor_edad_todas)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.980495640202
Exec_time:  0.00420784950256 s
```

In [302]:

```
Feature_Reduction.append("Todas")
Model.append("Random_forest")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- rango_edad

In [303]:

```
model = RandomForestClassifier(n_estimators = 8,
                                   max_features = 8,
                                   min_samples_leaf= 1)
model.fit(X_train_todas, y_train_rango_todas)
model.score(X_test_todas,y_test_rango_todas)
score = model.score(X_test_todas,y_test_rango_todas)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.680816888481
Exec_time:  0.004075050354 s
```

In [304]:

```
Feature_Reduction.append("Todas")
Model.append("Random_forest")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

2.- Varianza

```
- edad
```

In [305]:

```
model = RandomForestClassifier(n_estimators = 8,
                                   max_features = 8,
                                   min_samples_leaf= 1)
model.fit(X_train_varianza, y_train_varianza)
model.score(X_test_varianza,y_test_varianza)
score = model.score(X_test_varianza,y_test_varianza)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.592244148692
Exec_time:  0.00292015075684 s
```

In [306]:

```
Feature_Reduction.append("Varianza")
Model.append("Random_forest")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- mayor_edad

In [307]:

```
model = RandomForestClassifier(n_estimators = 8,
                                  max_features = 8,
                                  min_samples_leaf= 1)
model.fit(X_train_varianza, y_train_mayor_edad_V)
score = model.score(X_test_varianza,y_test_mayor_edad_V)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.980725103258
Exec_time:  0.00255012512207 s
```

In [308]:

```
Feature_Reduction.append("Varianza")
Model.append("Random_forest")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- rango_edad

In [309]:

```
model = RandomForestClassifier(n_estimators = 8,
                                   max_features = 8,
                                   min_samples_leaf= 1)
model.fit(X_train_varianza, y_train_rango_V)
score = model.score(X_test_varianza,y_test_rango_V)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.686324001836
Exec_time:  0.00260186195374 s
```

In [310]:

```
Feature_Reduction.append("Varianza")
Model.append("Random_forest")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

3.- Extra Tree

```
   - Edad
```

In [317]:

```
#Aqui tenemos solo 5 features, por lo que ponemos 5 en el máximo
model = RandomForestClassifier(n_estimators = 5,
                                max_features = 5,
                                min_samples_leaf= 1)
model.fit(X_train_ET, y_train_ET)
score = model.score(X_test_ET,y_test_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.570215695273
Exec_time:  0.00221610069275 s
```

In [312]:

```
Feature_Reduction.append("Extra_tree")
Model.append("Random_forest")
Target.append("edad")
Final_Score.append(model.score(X_test_ET,y_test_ET))
Parameters.append(model)
Exec_time.append(total_time)
```

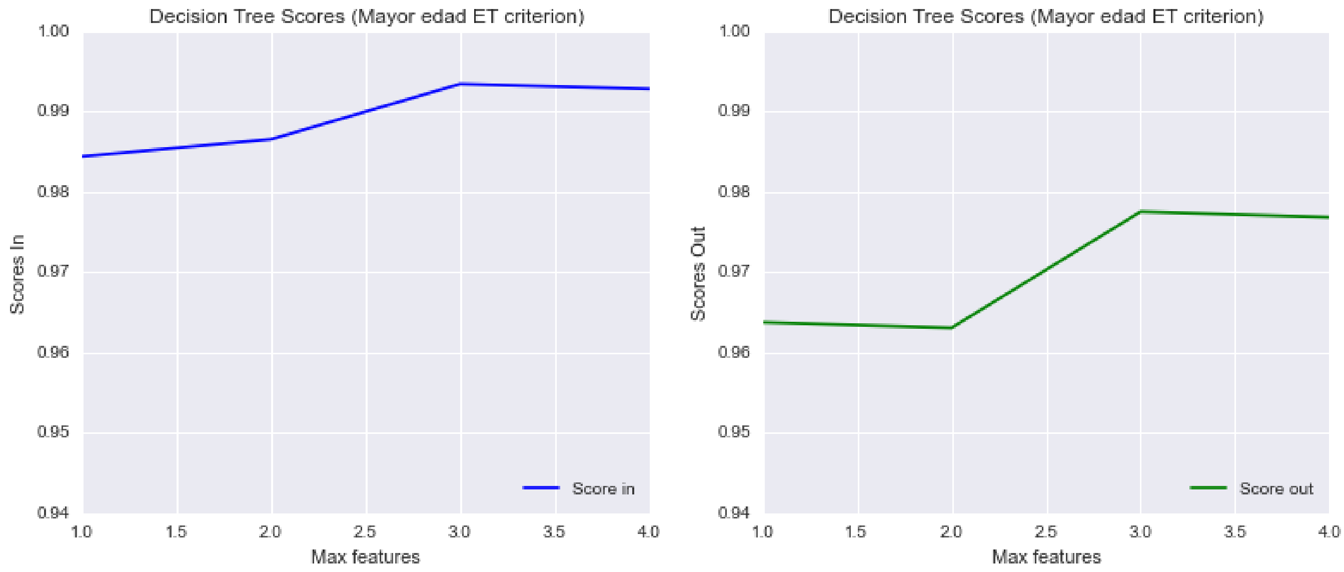
- Mayor_Edad

In [313]:

```
numero = X_test_ET.shape[1]
RF_measures = np.array([getRFMesures(1, max_features, max_features,
                                     y_train_mayor_edad_ET, X_train_ET,
                                     y_test_mayor_edad_ET, X_test_ET,'max_f
eatures')
                     for max_features in range(1, numero, 1)])
```

In [124]:

```
pinta_grafico_tree(RF_measures,0.94,1,0.94,1,'Mayor edad ET','Max feature
s')
```



In [320]:

```
#Ponemos 3

model = RandomForestClassifier(n_estimators = 3,
                               max_features = 3,
                               min_samples_leaf= 1)
model.fit(X_train_ET, y_train_mayor_edad_ET)
score = model.score(X_test_ET,y_test_mayor_edad_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.974759063791
Exec_time:  0.000830888748169 s
```

In [321]:

```
Feature_Reduction.append("Extra_tree")
Model.append("Random_forest")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- Rango_edad

In [127]:

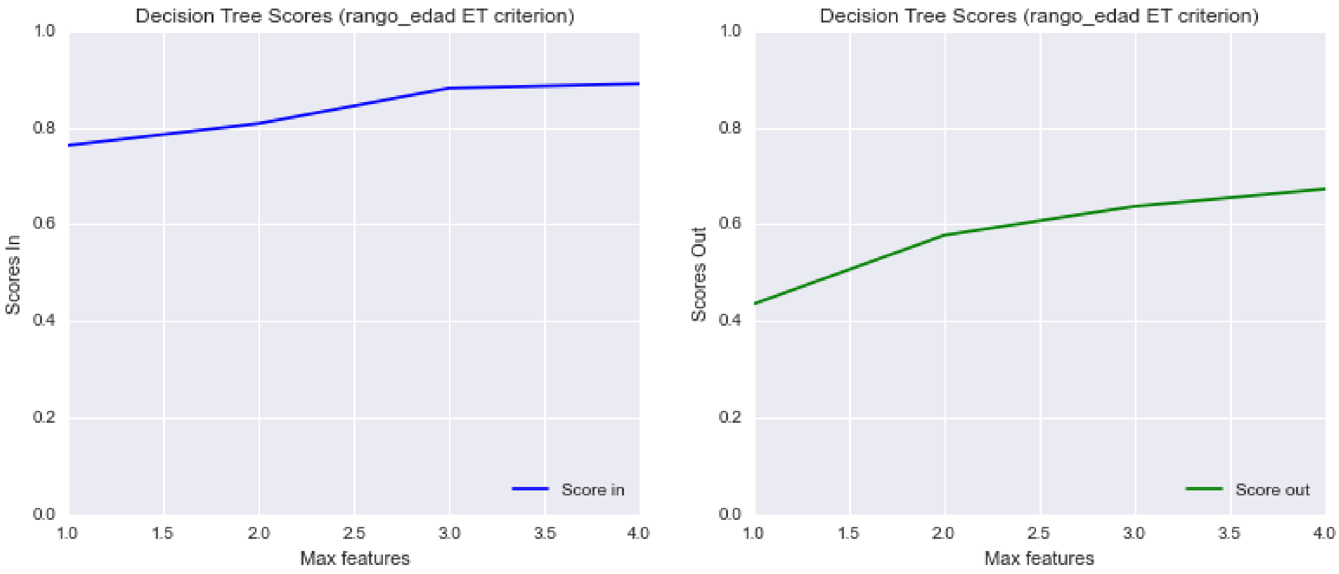
```
numero = X_test_ET.shape[1]
```

In [128]:

```
RF_measures = np.array([getRFMesures(1, max_features, max_features,
                                    y_train_rango_ET, X_train_ET,
                                    y_test_rango_ET, X_test_ET,'max_featur
es')
                     for max_features in range(1, numero, 1)])
```

```
In [129]:
```

```
pinta_grafico_tree(RF_measures,0,1,0,1,'rango_edad ET','Max features')
```



Metemos los 4 en el modelo

```
In [322]:
```

```
#Aqui tenemos solo 5 features, parece que 3 es suficiente
model = RandomForestClassifier(n_estimators = 4,
                                max_features = 4,
                                min_samples_leaf= 1)
model.fit(X_train_ET, y_train_rango_ET)
score = model.score(X_test_ET,y_test_rango_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.66085360257
Exec_time:  0.00161910057068 s
```

```
In [323]:
```

```
Feature_Reduction.append("Extra_tree")
Model.append("Random_forest")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

4.- PCA

```
- Edad
```

In [132]:

```
numero = trainDS_pca.shape[1]
print numero
```

```
57
```

In [133]:

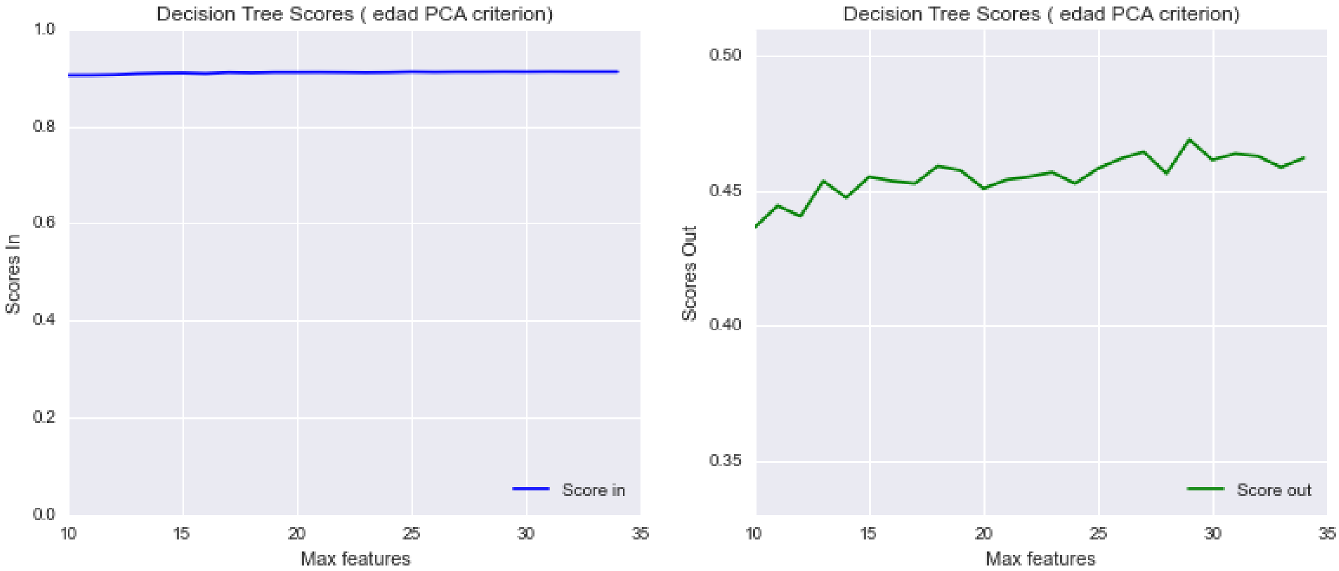
```
numero = 35
```

In [134]:

```
RF_measures = np.array([getRFMesures(1, max_features, max_features,
                                     y_train_PCA, trainDS_pca,
                                     y_test_PCA, testDS_pca,'max_features')
                    for max_features in range(10, numero, 1)])
```

In [135]:

```
pinta_grafico_tree(RF_measures,0,1,0.33,0.51,' edad PCA','Max features')
```



Cogemos 29

In [324]:

```
model = RandomForestClassifier(n_estimators = 29,
                               max_features =29,
                               min_samples_leaf= 1)
model.fit(X_train_PCA, y_train_PCA)
score = model.score(X_test_PCA,y_test_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.459385039009
Exec_time:  0.00884199142456 s
```

In [325]:

```
Feature_Reduction.append("PCA")
Model.append("Random_forest")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

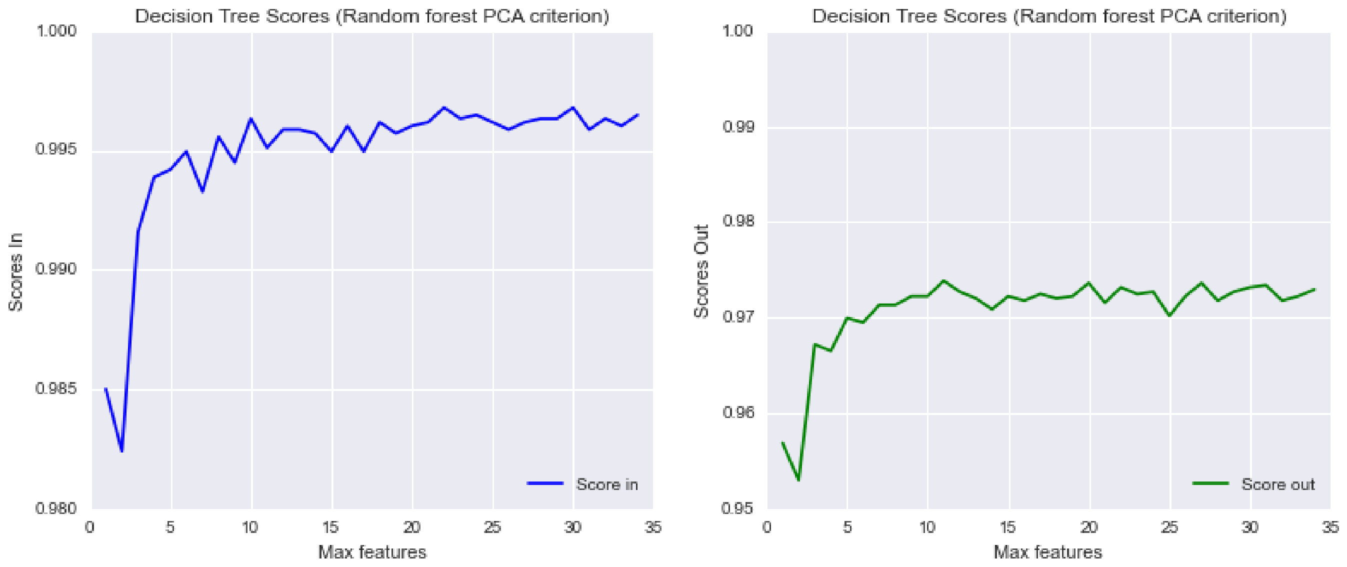
- mayor_edad

In [138]:

```
RF_measures = np.array([getRFMesures(1, max_features, max_features,
                                    y_train_mayor_edad_PCA, trainDS_pca,
                                    y_test_mayor_edad_PCA, testDS_pca,'ma
x_features')
                     for max_features in range(1, numero, 1)])
```

In [139]:

```
pinta_grafico_tree(RF_measures,0.98,1,0.95,1,'Random forest PCA','Max featu
res')
```



Cogemos 11

In [326]:

```
model = RandomForestClassifier(n_estimators = 11,
                               max_features =11,
                               min_samples_leaf= 1)
model.fit(X_train_PCA, y_train_mayor_edad_PCA)
score = model.score(X_test_PCA,y_test_mayor_edad_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.972693896283
Exec_time:  0.00681114196777 s
```

In [327]:

```
Feature_Reduction.append("PCA")
Model.append("Random_forest")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

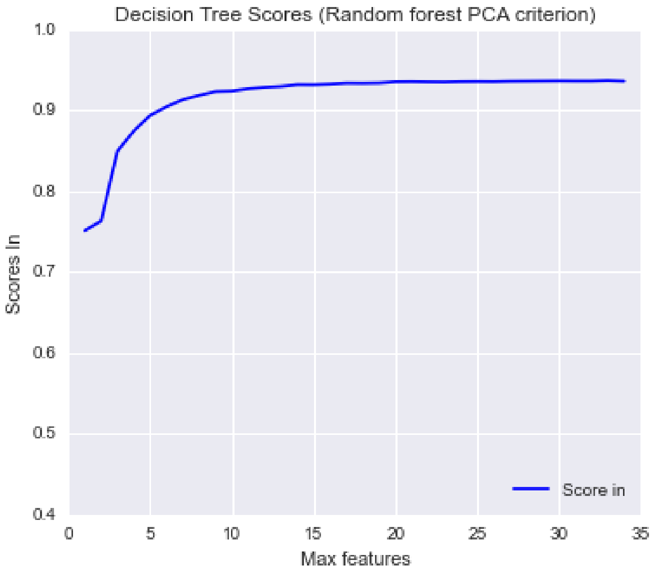
- Rango_edad

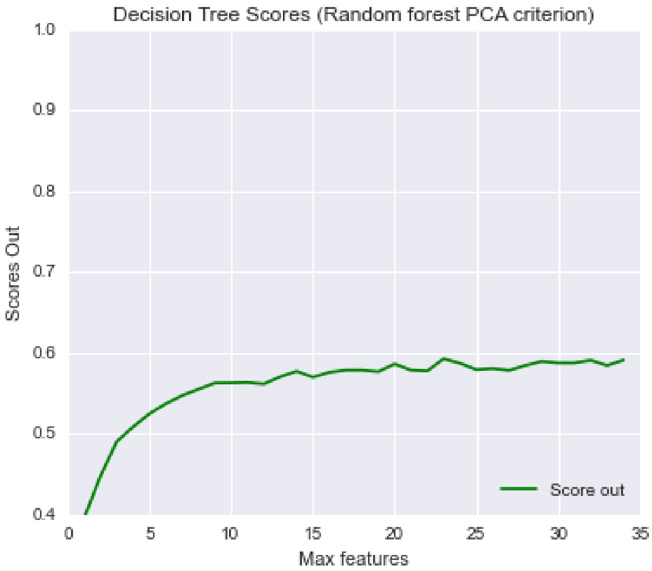
In [142]:

```
RF_measures = np.array([getRFMesures(1, max_features, max_features,
                                    y_train_rango_PCA, trainDS_pca,
                                    y_test_rango_PCA, testDS_pca,'max_feat
ures')
                     for max_features in range(1, numero, 1)])
```

In [143]:

```
pinta_grafico_tree(RF_measures,0.4,1,0.4,1,'Random forest PCA','Max feature
s')
```





In [328]:

```
#cogemos 23
model = RandomForestClassifier(n_estimators = 23,
                               max_features =23,
                               min_samples_leaf= 1)
model.fit(X_train_PCA, y_train_rango_PCA)
score = model.score(X_test_PCA,y_test_rango_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.transform(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.582147774208
Exec_time:  0.00610303878784 s
```

In [329]:

```
Feature_Reduction.append("PCA")
Model.append("Random_forest")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

8.- Regression

Desde el principio sabemos que este no es un problema de regresión, dado que todas las características son categóricas. En cualquier caso vamos a probar

In [146]:

```
from sklearn import linear_model
```

In [147]:

```
def getLinearRMesures(train_labels = None, train_data = None,
                      test_labels = None, test_data = None):
    model = linear_model.LinearRegression()
    model.fit(train_data, train_labels)
    return [model.intercept_,
            model.score(train_data, train_labels), #E_in
            model.score(test_data, test_labels)] #E_out
```

In [148]:

```
def getLogisticRMesures(C = 1e5, train_labels = None, train_data = None,
                 test_labels = None, test_data = None):
    model = linear_model.LogisticRegression(C = C)
    model.fit(train_data, train_labels)
    return [model.C,
            model.score(train_data, train_labels), #E_in
            model.score(test_data, test_labels)] #E_out
```

In [149]:

```
def pinta_Regression(LR_measures
                     , ax1_ylim1 = -1
                     , ax1_ylim2 = 1
                     , ax2_ylim1 = -1
                     , ax2_ylim2= 1
                     , tipo="Logistic"
                     , xlabel="Inverse of regularization strength (log)"
                    ):
    fig, axes = plt.subplots(ncols=2, figsize=(13, 5) )
    ax1, ax2 = axes.ravel()

    ax1.plot(LR_measures[:,0], LR_measures[:,1], label = 'Score in', c =
'b')
    ax1.legend(loc=4)
    ax1.set_title(tipo + 'Regresion Scores. Test data')
    ax1.set_xscale("log")
    ax1.set_xlabel(xlabel)
    ax1.set_ylabel("Train")
    ax1.set_ylim(ax1_ylim1,ax1_ylim2);

    #ax2.figure(figsize=(7,5))
    ax2.plot(LR_measures[:,0], LR_measures[:,2], label = 'Score out',
c='g')
    ax2.legend(loc=4)
    ax2.set_title(tipo + "Regresion Scores. Train data")
    ax2.set_xscale("log")
    ax2.set_xlabel(xlabel)
    ax2.set_ylabel('Test')
    ax2.set_ylim(ax2_ylim1,ax2_ylim2);
```

In [150]:

```
Cs = [1e5, 1e4, 1e3, 1e2, 1e1, 1e0, 1e-1,1e-2,1e-3,1e-4,1e-5]
```

1.- Todas

```
   - edad:
```

In [330]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_todas, y_train_todas)
score = model.score(X_test_todas, y_test_todas)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.053907981555
Exec_time:  0.00143313407898 s
```

In [331]:

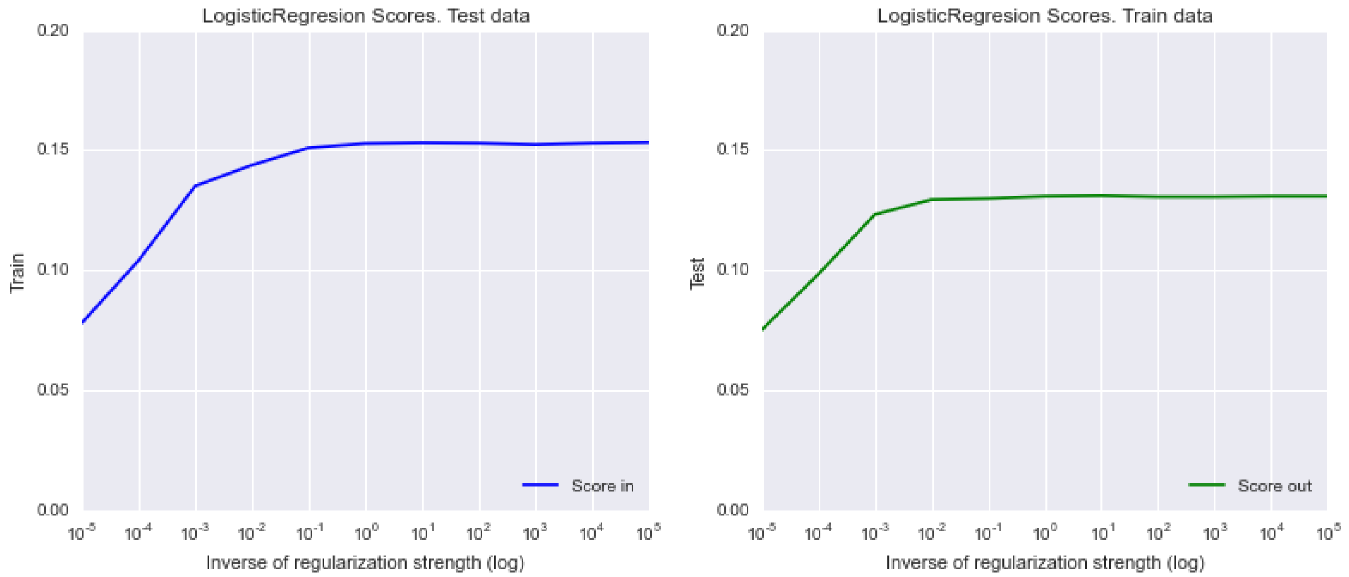
```
Feature_Reduction.append("Todas")
Model.append("Linear Regression")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [153]:

```
#Logistic Regression
LR_measures = np.array([getLogisticRMesures(C = c,
                                     train_labels=y_train_todas,
                                     train_data = X_train_todas,
                                     test_labels = y_test_todas,
                                     test_data = X_test_todas)
                        for c in Cs])
```

In [154]:

```
pinta_Regression(LR_measures,0,0.2,0,0.2)
```



Usamos como C = 1e-2

In [332]:

```
model = linear_model.LogisticRegression(C = 1e2)
model.fit(X_train_todas, y_train_todas)
score = model.score(X_test_todas,y_test_todas)

print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.131252868288
Exec_time:  0.0222470760345 s
```

In [333]:

```
Feature_Reduction.append("Todas")
Model.append("Logistic Regression")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- mayor_edad

In [334]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_todas, y_train_mayor_edad_todas)
score = model.score(X_test_todas, y_test_mayor_edad_todas)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.0103725752226
Exec_time:  0.00117206573486 s
```

In [335]:

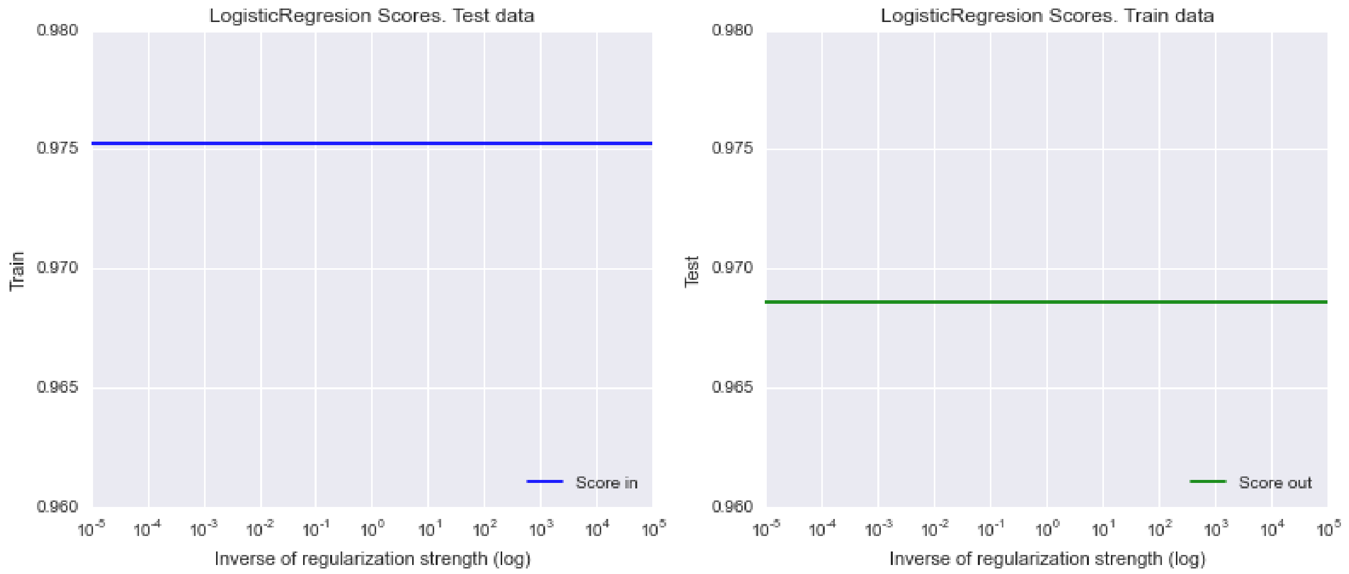
```
Feature_Reduction.append("Todas")
Model.append("Linear Regression")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [159]:

```
#Logistic Regression
LR_measures = np.array([getLogisticRMesures(C = c,
                                     train_labels=y_train_mayor_edad_todas,
                                     train_data = X_train_todas,
                                     test_labels = y_test_mayor_edad_todas,
                                     test_data = X_test_todas)
                        for c in Cs])
```

```
In [160]:
```

```
pinta_Regression(LR_measures,0.96,0.98,0.96,0.98)
```



Da igual el C

```
In [336]:
```

```
model = linear_model.LogisticRegression()
model.fit(X_train_todas, y_train_mayor_edad_todas)
score = model.score(X_test_todas,y_test_mayor_edad_todas)

print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.968563561267
Exec_time:  0.00119495391846 s
```

```
In [337]:
```

```
Feature_Reduction.append("Todas")
Model.append("Logistic Regression")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- rango_edad

In [338]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_todas, y_train_rango_todas)
score = model.score(X_test_todas, y_test_rango_todas)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.050242510922
Exec_time:  0.00133395195007 s
```

In [339]:

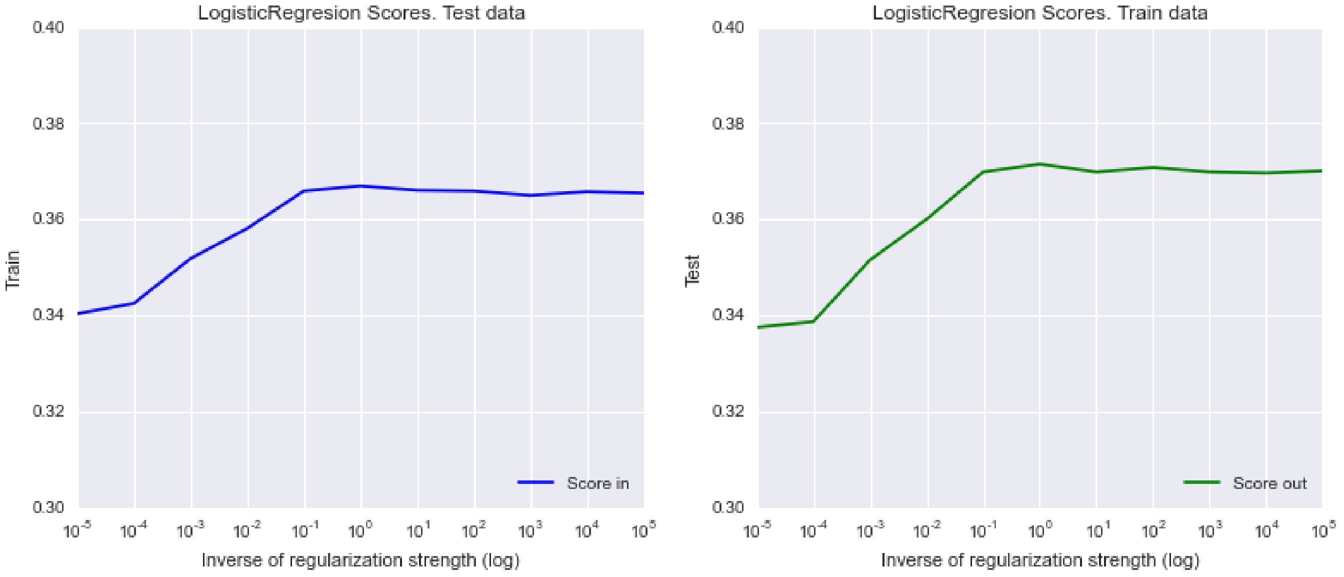
```
Feature_Reduction.append("Todas")
Model.append("Linear Regression")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [165]:

```
#Logistic Regression
LR_measures = np.array([getLogisticRMesures(C = c,
                                    train_labels=y_train_rango_todas,
                                    train_data = X_train_todas,
                                    test_labels = y_test_rango_todas,
                                    test_data = X_test_todas)
                        for c in Cs])
```

In [166]:

```
pinta_Regression(LR_measures,0.3,0.4,0.3,0.4)
```



Usamos C = 1e0

In [340]:

```
model = linear_model.LogisticRegression(C=1e0)
model.fit(X_train_todas, y_train_rango_todas)
score = model.score(X_test_todas,y_test_rango_todas)

print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.371500688389
Exec_time:  0.00454998016357 s
```

In [341]:

```
Feature_Reduction.append("Todas")
Model.append("Logistic Regression")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

2.- Varianza

- edad:

In [342]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_varianza, y_train_varianza)
score = model.score(X_test_varianza, y_test_varianza)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.0336905370999
Exec_time:  0.00152182579041 s
```

In [343]:

```
Feature_Reduction.append("Varianza")
Model.append("Linear Regression")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [171]:

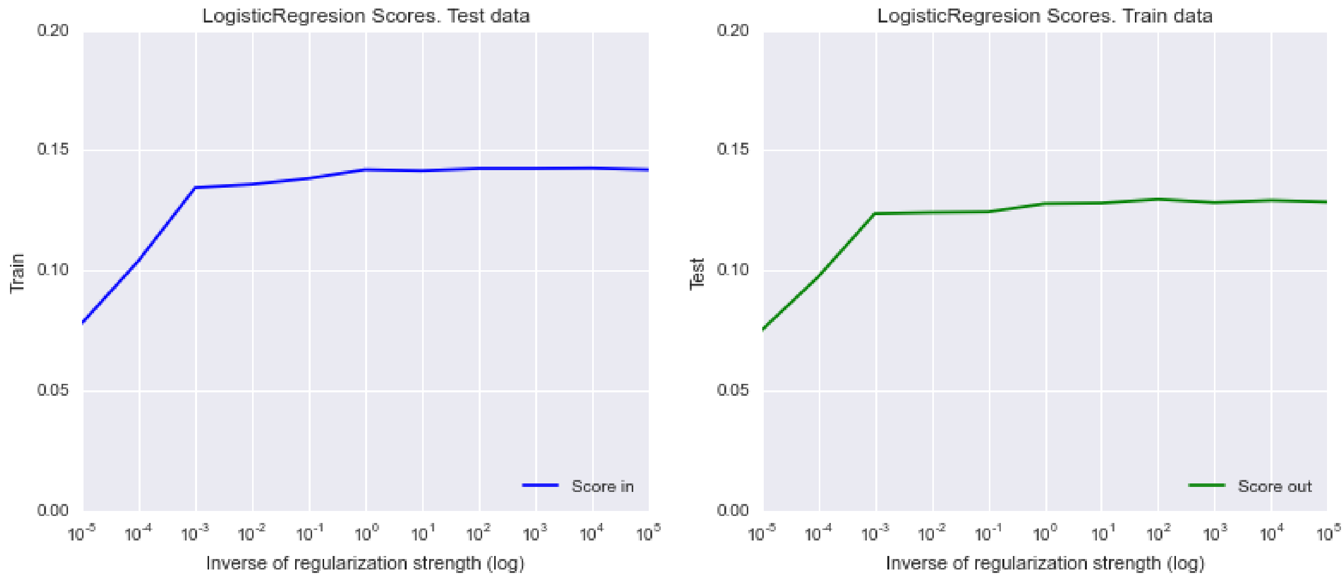
```
#Logistic Regression
```

In [172]:

```
LR_measures = np.array([getLogisticRMesures(C = c,
                                    train_labels=y_train_varianza,
                                    train_data = X_train_varianza,
                                    test_labels = y_test_varianza,
                                    test_data = X_test_varianza)
                        for c in Cs])
```

In [173]:

```
pinta_Regression(LR_measures,0,0.2,0,0.2)
```



Parece que con C = 1e0 es el mejor modelo (sin muchas diferencias)

In [344]:

```
model = linear_model.LogisticRegression(C = 1e0)
model.fit(X_train_varianza, y_train_varianza)
score = model.score(X_test_varianza,y_test_varianza)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.128269848554
Exec_time:  0.0142920017242 s
```

In [345]:

```
Feature_Reduction.append("Varianza")
Model.append("Logistic Regression")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- mayor_edad

In [346]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_varianza, y_train_mayor_edad_V)
score = model.score(X_test_varianza, y_test_mayor_edad_V)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.00807575583076
Exec_time:  0.00135207176208 s
```

In [347]:

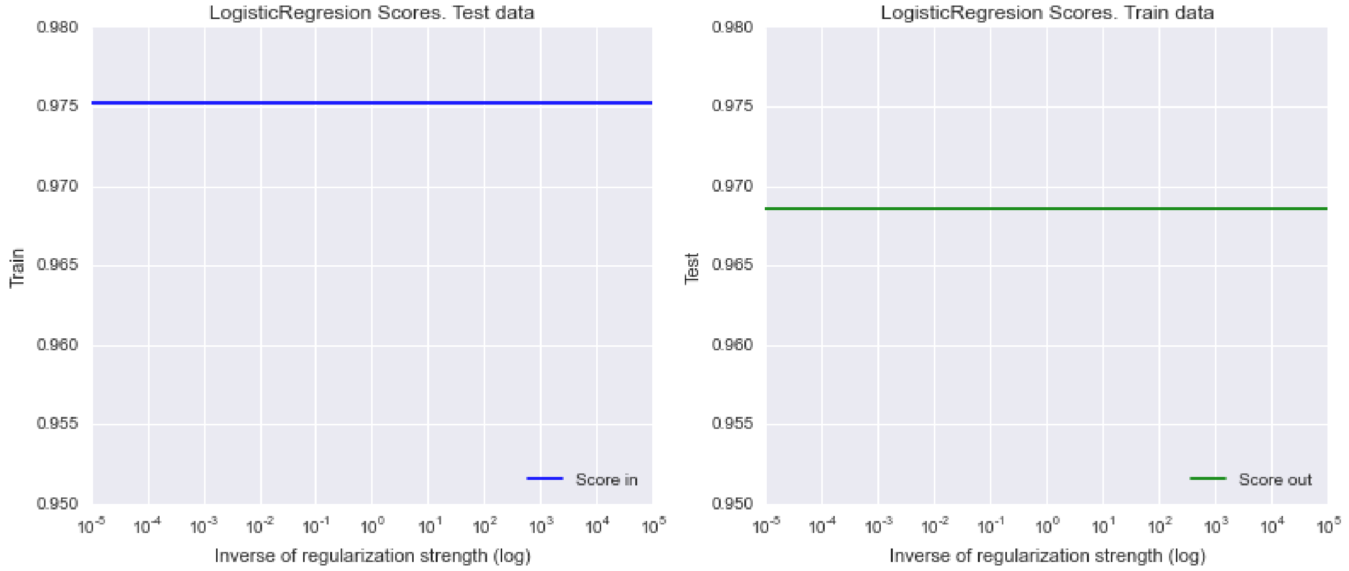
```
Feature_Reduction.append("Varianza")
Model.append("Linear Regression")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [178]:

```
#Logistic Regression
LR_measures = np.array([getLogisticRMesures(C = c,
                                    train_labels=y_train_mayor_edad_V,
                                    train_data = X_train_varianza,
                                    test_labels = y_test_mayor_edad_V,
                                    test_data = X_test_varianza)
                        for c in Cs])
```

In [179]:

```
pinta_Regression(LR_measures,0.95,0.98,0.95,0.98)
```



Da igual el C, dejamos el por defecto

In [348]:

```
model = linear_model.LogisticRegression()
model.fit(X_train_varianza, y_train_mayor_edad_V)
score = model.score(X_test_varianza,y_test_mayor_edad_V)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.968563561267
Exec_time:  0.00114607810974 s
```

In [349]:

```
Feature_Reduction.append("Varianza")
Model.append("Logistic Regression")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- rango_edad

In [350]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_varianza, y_train_rango_V)
score = model.score(X_test_varianza, y_test_rango_V)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.0282132554302
Exec_time:  0.0013530254364 s
```

In [351]:

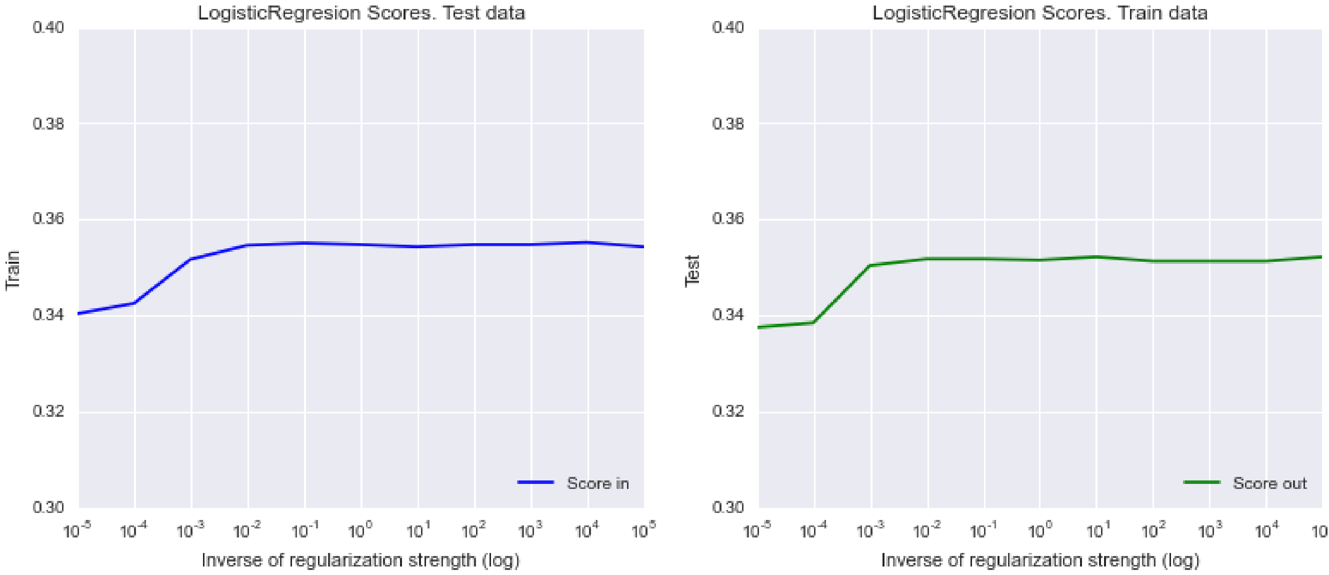
```
Feature_Reduction.append("Varianza")
Model.append("Linear Regression")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [184]:

```
#Logistic Regression
LR_measures = np.array([getLogisticRMesures(C = c,
                                    train_labels=y_train_rango_V,
                                    train_data = X_train_varianza,
                                    test_labels = y_test_rango_V,
                                    test_data = X_test_varianza)
                        for c in Cs])
```

In [185]:

```
pinta_Regression(LR_measures,0.3,0.4,0.3,0.4)
```



C= 1e-3

In [352]:

```
model = linear_model.LogisticRegression(C=1e-3)
model.fit(X_train_varianza, y_train_rango_V)
score = model.score(X_test_varianza,y_test_rango_V)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.350390087196
Exec_time:  0.00256013870239 s
```

In [353]:

```
Feature_Reduction.append("Varianza")
Model.append("Logistic Regression")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

2.- Extra Tree

- edad:

In [354]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_ET, y_train_ET)
score = model.score(X_test_ET, y_test_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.0175573822323
Exec_time:  0.000890016555786 s
```

In [355]:

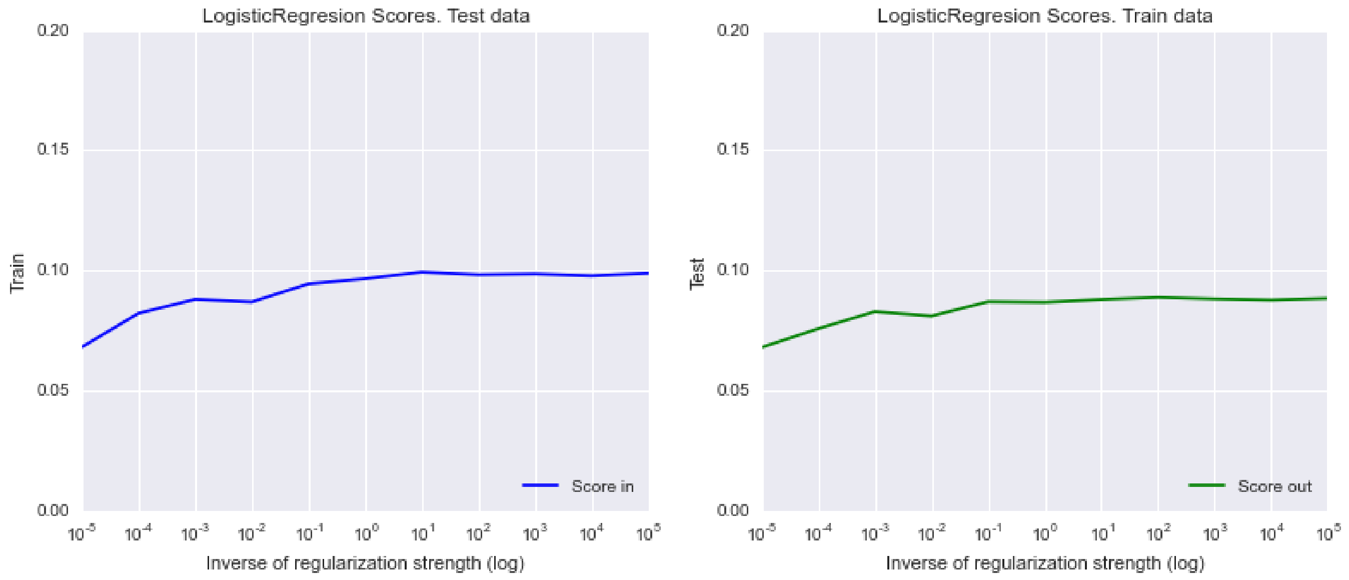
```
Feature_Reduction.append("Extra_tree")
Model.append("Linear Regression")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [190]:

```
#Logistic Regression
LR_measures = np.array([getLogisticRMesures(C = c,
                                    train_labels=y_train_ET,
                                    train_data = X_train_ET,
                                    test_labels = y_test_ET,
                                    test_data = X_test_ET)
                        for c in Cs])
```

```
In [191]:
```

```
pinta_Regression(LR_measures,0,0.2,0,0.2)
```



Cogemos C = 1e-1

```
In [356]:
```

```
model = linear_model.LogisticRegression(C=1e-1)
model.fit(X_train_ET, y_train_ET)
score = model.score(X_test_ET,y_test_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.0869664983938
Exec_time:  0.0113918781281 s
```

```
In [357]:
```

```
Feature_Reduction.append("Extra_tree")
Model.append("Logistic Regression")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- mayor_edad

In [358]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_ET, y_train_mayor_edad_ET)
score = model.score(X_test_ET, y_test_mayor_edad_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.000575541742781
Exec_time:  0.000990152359009 s
```

In [359]:

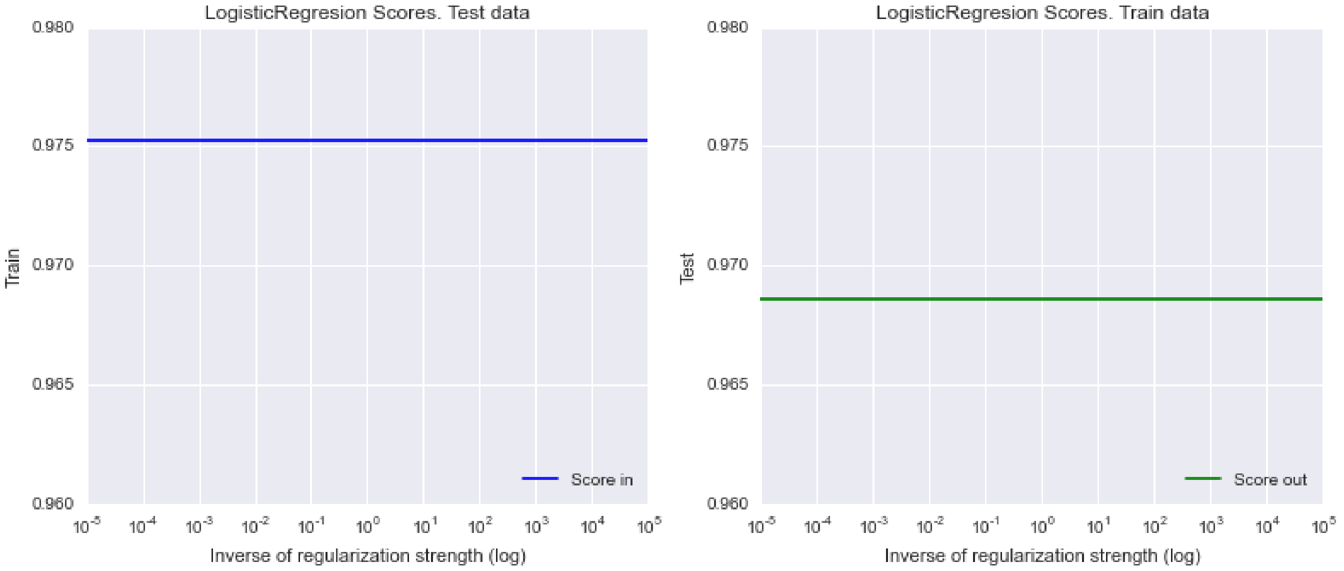
```
Feature_Reduction.append("Extra_tree")
Model.append("Linear Regression")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [196]:

```
#Logistic Regression
LR_measures = np.array([getLogisticRMesures(C = c,
                                     train_labels=y_train_mayor_edad_ET,
                                     train_data = X_train_ET,
                                     test_labels = y_test_mayor_edad_ET,
                                     test_data = X_test_ET)
                        for c in Cs])
```

```
In [197]:
```

```
pinta_Regression(LR_measures,0.96,0.98,0.96,0.98)
```



No hay diferencia en C

```
In [360]:
```

```
model = linear_model.LogisticRegression()
model.fit(X_train_ET, y_train_mayor_edad_ET)
score = model.score(X_test_ET,y_test_mayor_edad_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.968563561267
Exec_time:  0.000962018966675 s
```

```
In [361]:
```

```
Feature_Reduction.append("Extra_tree")
Model.append("Logistic Regression")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

```
    - rango_edad
```

In [362]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_ET, y_train_rango_ET)
score = model.score(X_test_ET, y_test_rango_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.0147163176909
Exec_time:  0.00104594230652 s
```

In [363]:

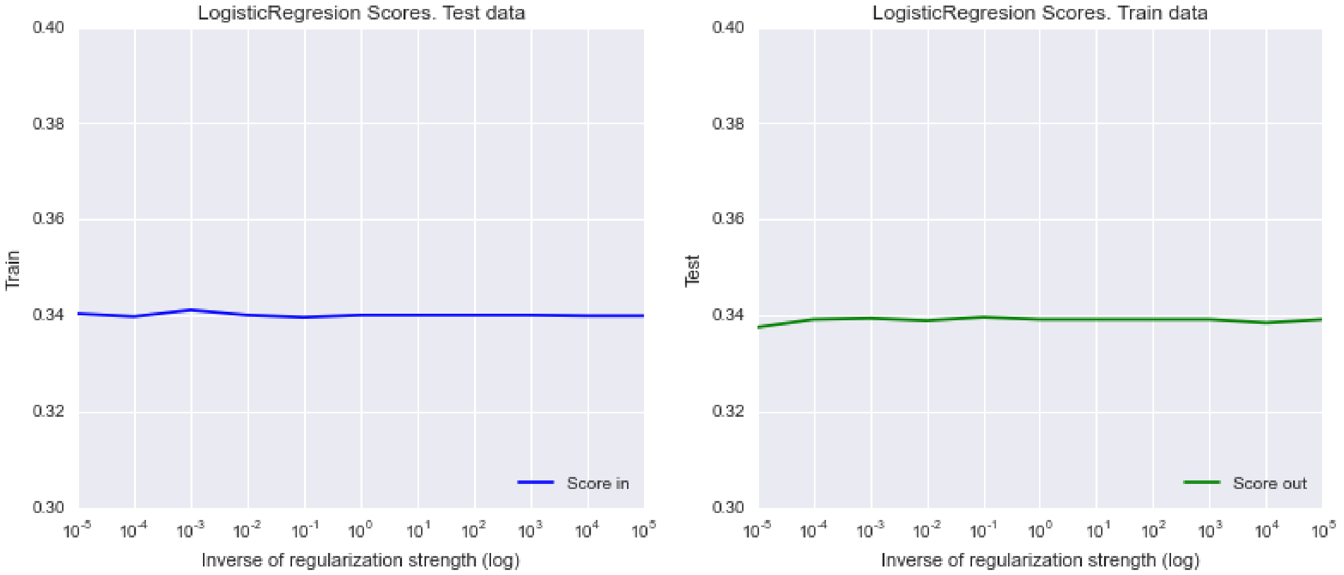
```
Feature_Reduction.append("Extra_tree")
Model.append("Linear Regression")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [202]:

```
#Logistic Regression
LR_measures = np.array([getLogisticRMesures(C = c,
                                    train_labels=y_train_rango_ET,
                                    train_data = X_train_ET,
                                    test_labels = y_test_rango_ET,
                                    test_data = X_test_ET)
                        for c in Cs])
```

In [203]:

```
pinta_Regression(LR_measures,0.3,0.4,0.3,0.4)
```



C= 1e-1

In [364]:

```
model = linear_model.LogisticRegression(C=1e-1)
model.fit(X_train_ET, y_train_rango_ET)
score = model.score(X_test_ET,y_test_rango_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.339605323543
Exec_time:  0.00224304199219 s
```

In [365]:

```
Feature_Reduction.append("Extra_tree")
Model.append("Logistic Regression")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

3.- PCA

    -edad

In [366]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_PCA, y_train_PCA)
score = model.score(X_test_PCA, y_test_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.0971155586696
Exec_time:  0.00174689292908 s
```

In [367]:

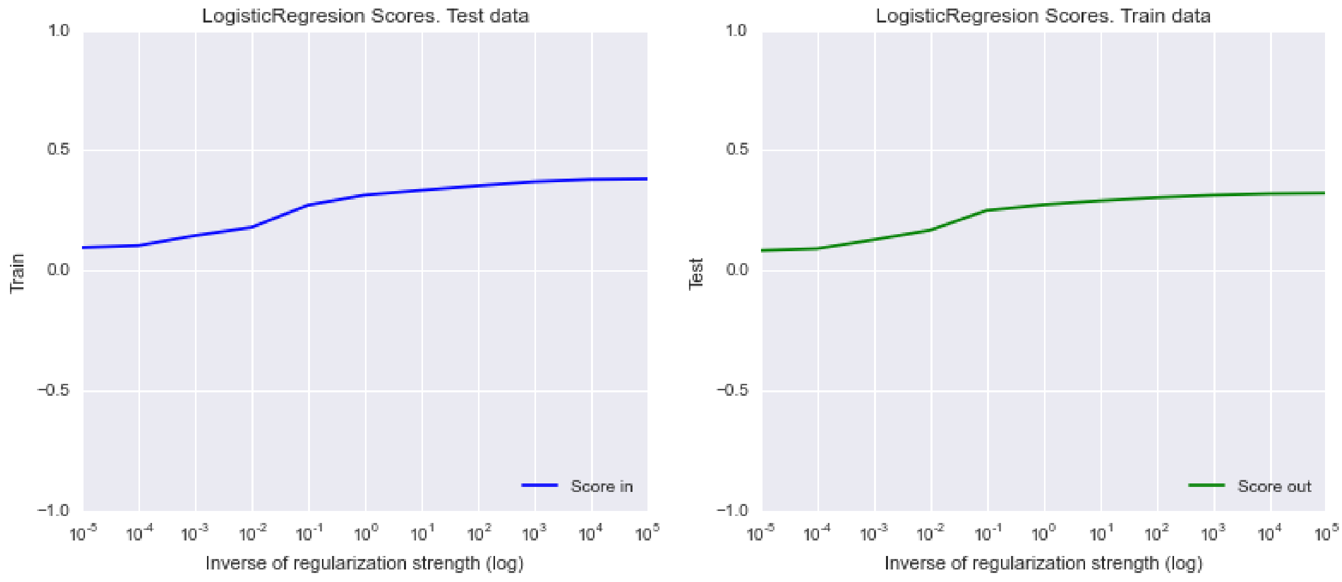
```
Feature_Reduction.append("PCA")
Model.append("Linear Regression")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [208]:

```
#Logistic Regression
LR_measures = np.array([getLogisticRMesures(C = c,
                                     train_labels=y_train_PCA,
                                     train_data = X_train_PCA,
                                     test_labels = y_test_PCA,
                                     test_data = X_test_PCA)
                        for c in Cs])
```

In [209]:

```
pinta_Regression(LR_measures)
```



c= 1e1

In [368]:

```
model = linear_model.LogisticRegression(C=1e1)
model.fit(X_train_PCA, y_train_PCA)
score = model.score(X_test_PCA,y_test_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.29004130335
Exec_time:  0.018100976944 s
```

In [369]:

```
Feature_Reduction.append("PCA")
Model.append("Logistic Regression")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

```
    - Mayor_edad
```

In [370]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_PCA, y_train_mayor_edad_PCA)
score = model.score(X_test_PCA, y_test_mayor_edad_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.0265939243556
Exec_time:  0.00161910057068 s
```

In [371]:

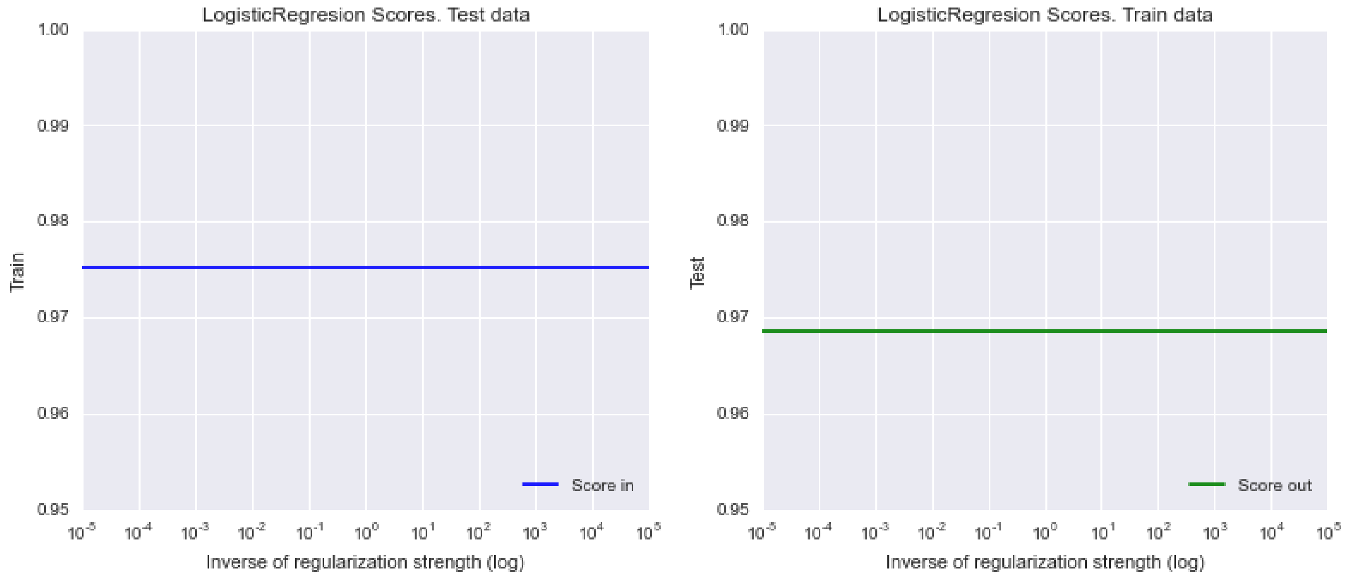
```
Feature_Reduction.append("PCA")
Model.append("Linear Regression")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [214]:

```
#Logistic Regression
LR_measures = np.array([getLogisticRMesures(C = c,
                                     train_labels=y_train_mayor_edad_PCA,
                                     train_data = X_train_PCA,
                                     test_labels = y_test_mayor_edad_PCA,
                                     test_data = X_test_PCA)
                        for c in Cs])
```

In [215]:

```
pinta_Regression(LR_measures,0.95,1,0.95,1)
```



C da igual

In [372]:

```
model = linear_model.LogisticRegression()
model.fit(X_train_PCA, y_train_mayor_edad_PCA)
score = model.score(X_test_PCA,y_test_mayor_edad_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.968563561267
Exec_time:  0.00169491767883 s
```

In [373]:

```
Feature_Reduction.append("PCA")
Model.append("Logistic Regression")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

```
    - Rango_edad
```

In [374]:

```
#Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train_PCA, y_train_rango_PCA)
score = model.score(X_test_PCA, y_test_rango_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.0988269109273
Exec_time:  0.00155091285706 s
```

In [375]:

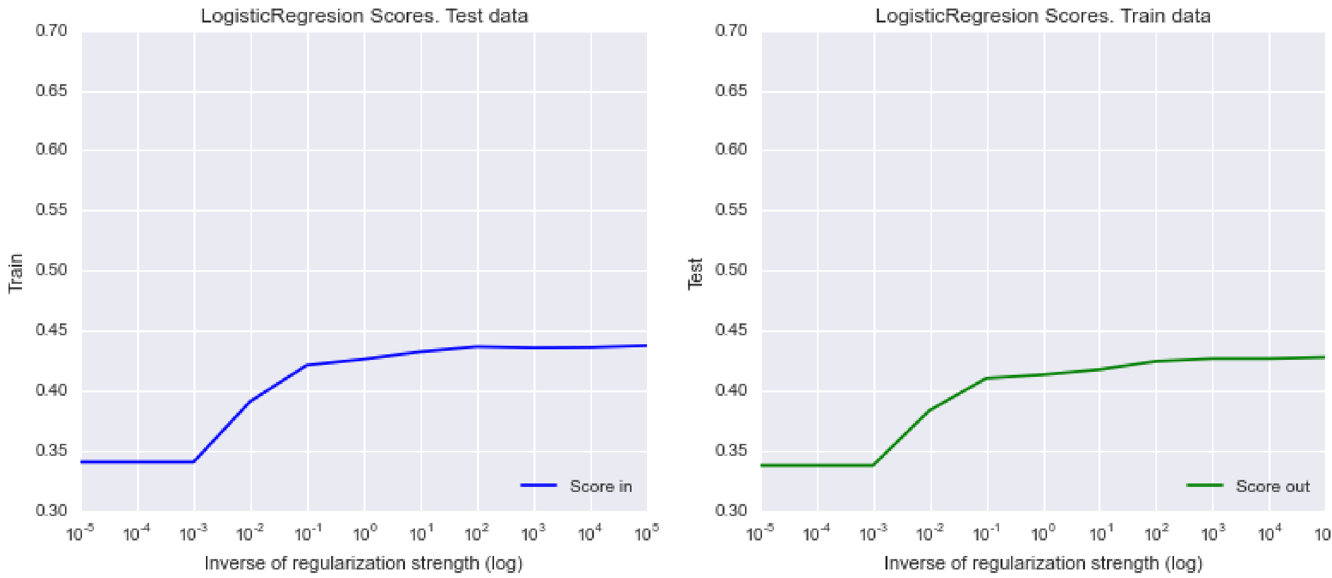
```
Feature_Reduction.append("PCA")
Model.append("Linear Regression")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [220]:

```
#Logistic Regression
LR_measures = np.array([getLogisticRMesures(C = c,
                                    train_labels=y_train_rango_PCA,
                                    train_data = X_train_PCA,
                                    test_labels = y_test_rango_PCA,
                                    test_data = X_test_PCA)
                        for c in Cs])
```

In [221]:

```
pinta_Regression(LR_measures,0.3,0.7,0.3,0.7)
```



C = 1e2

In [376]:

```
model = linear_model.LogisticRegression(C=1e2)
model.fit(X_train_PCA, y_train_rango_PCA)
score = model.score(X_test_PCA,y_test_rango_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.424506654429
Exec_time:  0.00485491752625 s
```

In [377]:

```
Feature_Reduction.append("PCA")
Model.append("Logistic Regression")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

9.- Support Vector Machines

In [224]:

```
from sklearn.svm import SVC
```

In [225]:

```
from sklearn.cross_validation import StratifiedShuffleSplit
```

In [226]:

```
def getSVMMesures(C = 1, kernel = 'rbf', max_iter = 1e3, tol = 1e-3,
                  train_size = None, n_experiments = 1,
                  train_labels = None, train_data = None, gamma = 0.0, degr
ee = 3, order = None):


    model = SVC(C = C, gamma = gamma, kernel = kernel, degree = degree, tol
= tol, max_iter = max_iter)

    test_size = train_size * 0.2
    sss = StratifiedShuffleSplit(train_labels, n_iter = n_experiments,
                                 train_size = train_size,
                                 test_size = test_size,
                                 random_state = np.random.random_integer
s(0,100000))

    modelsFitted = [model.fit(train_data[train_ix, :], train_labels[train_i
x])
                    for train_ix, test_ix in sss]

    scores = [(m.score(train_data[train_ix, :], train_labels[train_ix]),
              m.score(train_data[test_ix, :], train_labels[test_ix]))
              for m, (train_ix, test_ix) in zip(modelsFitted, sss)]

    meanScores = np.mean(scores, axis = 0)
    maxScores = np.max(scores, axis = 0)
    minScores = np.min(scores, axis = 0)

    return [model.C,
            model.kernel,
            model.max_iter,
            model.tol,
            train_size,
            meanScores[0],# mean E_in
            maxScores[0], # max E_in
            minScores[0], # min E_in
            meanScores[1],# mean E_out
            maxScores[1], # max E_out
            minScores[1], # min E_out
            order,
            model.gamma,
            model.degree,
            model] #model
```

In [227]:

```
def pintaSVM(SVM_Linear_measures_size):
    fig, axes = plt.subplots(ncols=2, figsize=(20, 10) )
    ax1, ax2 = axes.ravel()

    ax1.plot(SVM_Linear_measures_size[:,4], SVM_Linear_measures_size[:,5],
label = 'mean Score in', c = 'b')
    ax1.plot(SVM_Linear_measures_size[:,4], SVM_Linear_measures_size[:,6],
label = 'max Score in', c = 'g')
    ax1.plot(SVM_Linear_measures_size[:,4], SVM_Linear_measures_size[:,7],
label = 'min Score in', c = 'r')
    ax1.legend(loc = 4)
    ax1.grid()
    ax1.set_title('SVM: select the size of training Set (kernel = linear).
E_in')
    ax1.set_xlabel("% of Train Data Set")
    ax1.set_ylabel("Scores In")
    ax1.set_ylim(0.65,0.97);



    ax2.plot(SVM_Linear_measures_size[:,4], SVM_Linear_measures_size[:,8],
label = 'mean Score out', c = 'b')
    ax2.plot(SVM_Linear_measures_size[:,4], SVM_Linear_measures_size[:,9],
label = 'max Score out', c = 'g')
    ax2.plot(SVM_Linear_measures_size[:,4], SVM_Linear_measures_size[:,10],
label = 'min Score out', c = 'r')
    ax2.legend(loc = 4)
    ax2.grid()
    ax2.set_title("SVM: select the size of training Set (kernel = linear).
E_out")
    ax2.set_xlabel("% of Train Data Set")
    ax2.set_ylabel("Scores Out")
    ax2.set_ylim(0.65,0.97);
```

In [228]:

```
svmModel = SVC()
svmModel.fit(X_train_PCA, y_train_mayor_edad_PCA)
```

Out[228]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamm
a=0.0,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [229]:

```
svmModel.score(X_test_PCA, y_test_mayor_edad_PCA)
```

Out[229]:

```
0.96856356126663612
```

1.- Todas

    -edad

In [378]:

```
model = SVC()
model.fit(X_train_todas, y_train_todas)
score = model.score(X_test_todas, y_test_todas)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.535566773749
Exec_time:  11.7063238621 s
```

In [379]:

```
Feature_Reduction.append("Todas")
Model.append("SVM")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

    - mayor_edad

In [380]:

```
model = SVC()
model.fit(X_train_todas, y_train_mayor_edad_todas)
score = model.score(X_test_todas, y_test_mayor_edad_todas)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.977512620468
Exec_time:  1.54106593132 s
```

In [381]:

```
Feature_Reduction.append("Todas")
Model.append("SVM")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

```
- Rango Edad
```

In [382]:

```
model = SVC()
model.fit(X_train_todas, y_train_rango_todas)
score = model.score(X_test_todas, y_test_rango_todas)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.667278568151
Exec_time:  3.55187988281 s
```

In [383]:

```
Feature_Reduction.append("Todas")
Model.append("SVM")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

2.- Varianza

```
-edad
```

In [384]:

```
model = SVC()
model.fit(X_train_varianza, y_train_varianza)
score = model.score(X_test_varianza, y_test_varianza)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.52455254704
Exec_time:  11.015283823 s
```

In [385]:

```
Feature_Reduction.append("Varianza")
Model.append("SVM")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- mayor_edad

In [386]:

```
model = SVC()
model.fit(X_train_varianza, y_train_mayor_edad_V)
score = model.score(X_test_varianza, y_test_mayor_edad_V)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.977283157412
Exec_time:  1.40993118286 s
```

In [387]:

```
Feature_Reduction.append("Varianza")
Model.append("SVM")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

-rango_edad

In [388]:

```
model = SVC()
model.fit(X_train_varianza, y_train_rango_V)
score = model.score(X_test_varianza, y_test_rango_V)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_varianza)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.663836622304
Exec_time:  3.0750169754 s
```

In [389]:

```
Feature_Reduction.append("Varianza")
Model.append("SVM")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

3.- Extra Tree

-edad

In [390]:

```
model = SVC()
model.fit(X_train_ET, y_train_ET)
score = model.score(X_test_ET, y_test_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.487150068839
Exec_time:  11.675369978 s
```

In [391]:

```
Feature_Reduction.append("Extra_tree")
Model.append("SVM")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- mayor_edad

In [392]:

```
model = SVC()
model.fit(X_train_ET, y_train_mayor_edad_ET)
score = model.score(X_test_ET, y_test_mayor_edad_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.976365305186
Exec_time:  1.17367696762 s
```

In [393]:

```
Feature_Reduction.append("Extra_tree")
Model.append("SVM")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

-rango_edad

In [394]:

```
model = SVC()
model.fit(X_train_ET, y_train_rango_ET)
score = model.score(X_test_ET, y_test_rango_ET)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_ET)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.644102799449
Exec_time:  2.69380497932 s
```

In [395]:

```
Feature_Reduction.append("Extra_tree")
Model.append("SVM")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

4.- PCA

```
    - edad
```

In [396]:

```
model = SVC()
model.fit(X_train_PCA, y_train_PCA)
score = model.score(X_test_PCA, y_test_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.241395135383
Exec_time:  15.7033350468 s
```

In [397]:

```
Feature_Reduction.append("PCA")
Model.append("SVM")
Target.append("edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- mayor edad

In [398]:

```
model = SVC()
model.fit(X_train_PCA, y_train_mayor_edad_PCA)
score = model.score(X_test_PCA, y_test_mayor_edad_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.968563561267
Exec_time:  0.79682302475 s
```

In [399]:

```
Feature_Reduction.append("PCA")
Model.append("SVM")
Target.append("mayor_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

- rango_edad

In [400]:

```
model = SVC()
model.fit(X_train_PCA, y_train_rango_PCA)
score = model.score(X_test_PCA, y_test_rango_PCA)
print "Score: ",score

#Medimos el tiempo que tarda en predecir
start_time = time.time()
model.predict(X_predecir_PCA)
end_time = time.time()
total_time =  end_time - start_time
print "Exec_time: ",total_time , "s"
```

```
Score:  0.4056906838
Exec_time:  9.50265407562 s
```

In [401]:

```
Feature_Reduction.append("PCA")
Model.append("SVM")
Target.append("rango_edad")
Final_Score.append(score)
Parameters.append(model)
Exec_time.append(total_time)
```

In [402]:

```
print score
```

```
0.4056906838
```

10.- Resultados

In [403]:

```
Resultados = pd.DataFrame()
Resultados["feature_reduction"] = Feature_Reduction
Resultados["model"]= Model
Resultados["taget"] = Target
Resultados["score"] = Final_Score
Resultados["parameters"] = Parameters
Resultados["exec_time"] = Exec_time
```

In [404]:

```
Resultados.feature_reduction = ["Todas" if feature_reduction=='todas'
                                else feature_reduction
                                for feature_reduction in Resultados.featur
e_reduction]
```

In [405]:

```
print np.unique(["Todas" if feature_reduction=='todas'
                                else feature_reduction
                                for feature_reduction in Resultados.featur
e reduction])
```

```
['Extra_tree' 'PCA' 'Todas' 'Varianza']
```

In [413]:

```
pinta_edad = Resultados[Resultados.taget == "edad"].sort(['score','exec_tim
e'],ascending=(0,1)).head(5)
print pinta edad[['feature reduction','model','score','exec time']]
```

```
   feature_reduction            model     score  exec_time
15          Varianza   Random_forest  0.592244   0.002920
12             Todas   Random_forest  0.584442   0.004300
0              Todas  Decission_tree  0.572740   0.001324
3           Varianza  Decission_tree  0.572051   0.002111
18        Extra_tree   Random_forest  0.569986   0.002245
```

In [414]:

```
pinta_mayor_edad =  Resultados[Resultados.taget == "mayor_edad"].sort(['sco
re','exec_time'],ascending=(0,1)).head(5)
print pinta mayor edad[['feature reduction','model','score','exec time']]
```

```
   feature_reduction            model     score  exec_time
16          Varianza   Random_forest  0.980725   0.002550
13             Todas   Random_forest  0.980496   0.004208
7         Extra_tree  Decission_tree  0.977972   0.000901
49             Todas             SVM  0.977513   1.541066
52          Varianza             SVM  0.977283   1.409931
```

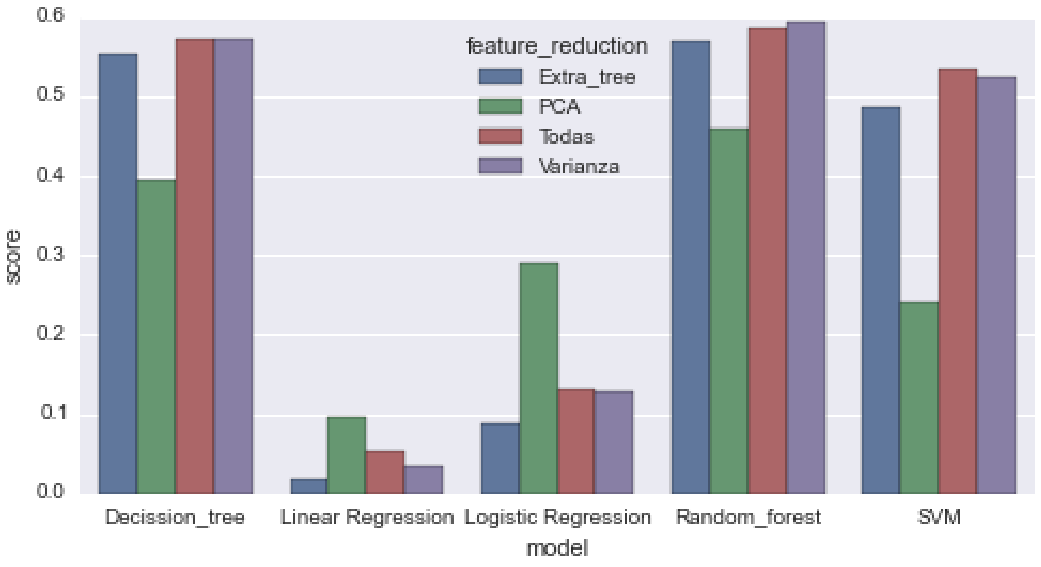
In [415]:

```
pinta_rango_edad =  Resultados[Resultados.taget == "rango_edad"].sort(['sco
re','exec_time'],ascending=(0,1)).head(5)
print pinta rango edad[['feature reduction','model','score','exec time']]
```

```
   feature_reduction            model     score  exec_time
17          Varianza   Random_forest  0.686324   0.002602
2              Todas  Decission_tree  0.684488   0.001598
14             Todas   Random_forest  0.680817   0.004075
5           Varianza  Decission_tree  0.673704   0.002036
50             Todas             SVM  0.667279   3.551880
```

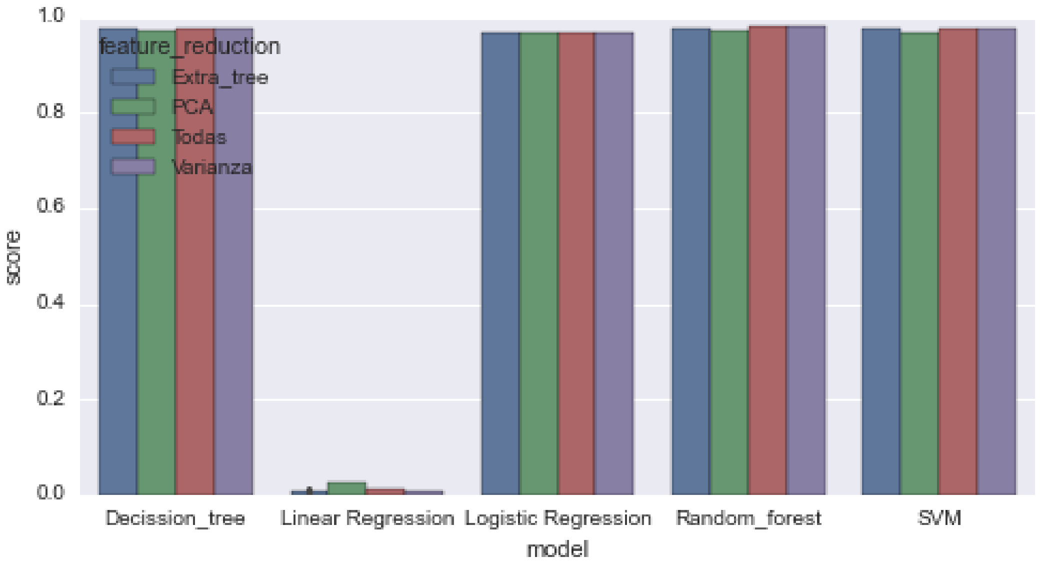
In [409]:

```
# Plot the feature importances of the forest
resul_edad = Resultados[Resultados.taget == "edad"]
resul_edad = resul_edad[resul_edad.score >0]
resul_mayor_edad = Resultados[Resultados.taget == "mayor_edad"]
resul_mayor_edad = resul_mayor_edad[resul_mayor_edad.score >0]
resul_rango_edad = Resultados[Resultados.taget == "rango_edad"]
resul_rango_edad = resul_rango_edad[resul_rango_edad.score>0]


sns.barplot(x="model", y="score", hue="feature_reduction", data=resul_eda
d);
```



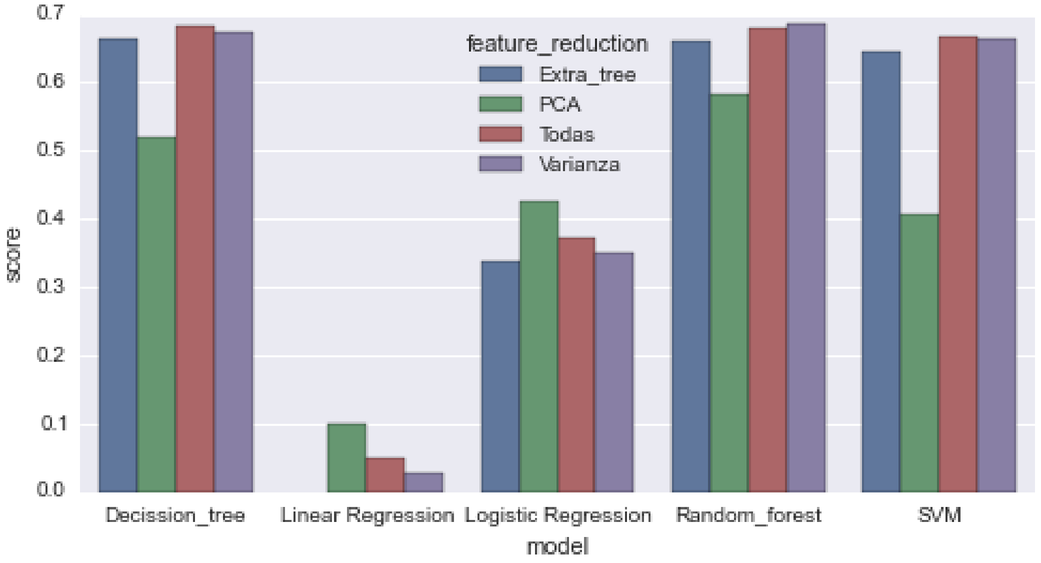
In [412]:

```
sns.barplot(x="model", y="score", hue="feature_reduction", data=resul_mayo
r_edad);
```



In [411]:

```
sns.barplot(x="model", y="score", hue="feature_reduction", data=resul_rang
o_edad);
```



In [ ]:

```

```