In [195]:
```
# ubicacion de los sismos, fuentes sismologicas, ley gutenberg-richter
```

In [196]:
```
#Librerias a usar

from time import time
import matplotlib
import matplotlib.pyplot as plt
import sklearn as sk
import numpy as np
import pandas as pd
import seaborn as sb
from datetime import datetime, timedelta
import joblib
from sklearn.linear_model import Ridge

import math
```

In [197]:
```
print(matplotlib.__version__)
print(sk.__version__)
print(np.__version__)
print(pd.__version__)
print(sb.__version__)
#versiones a usar
#3.4.2
#0.24.2
#1.20.2
#1.2.5
#0.11.1
```

```
3.4.2
0.24.2
1.20.2
1.2.5
0.11.1
```

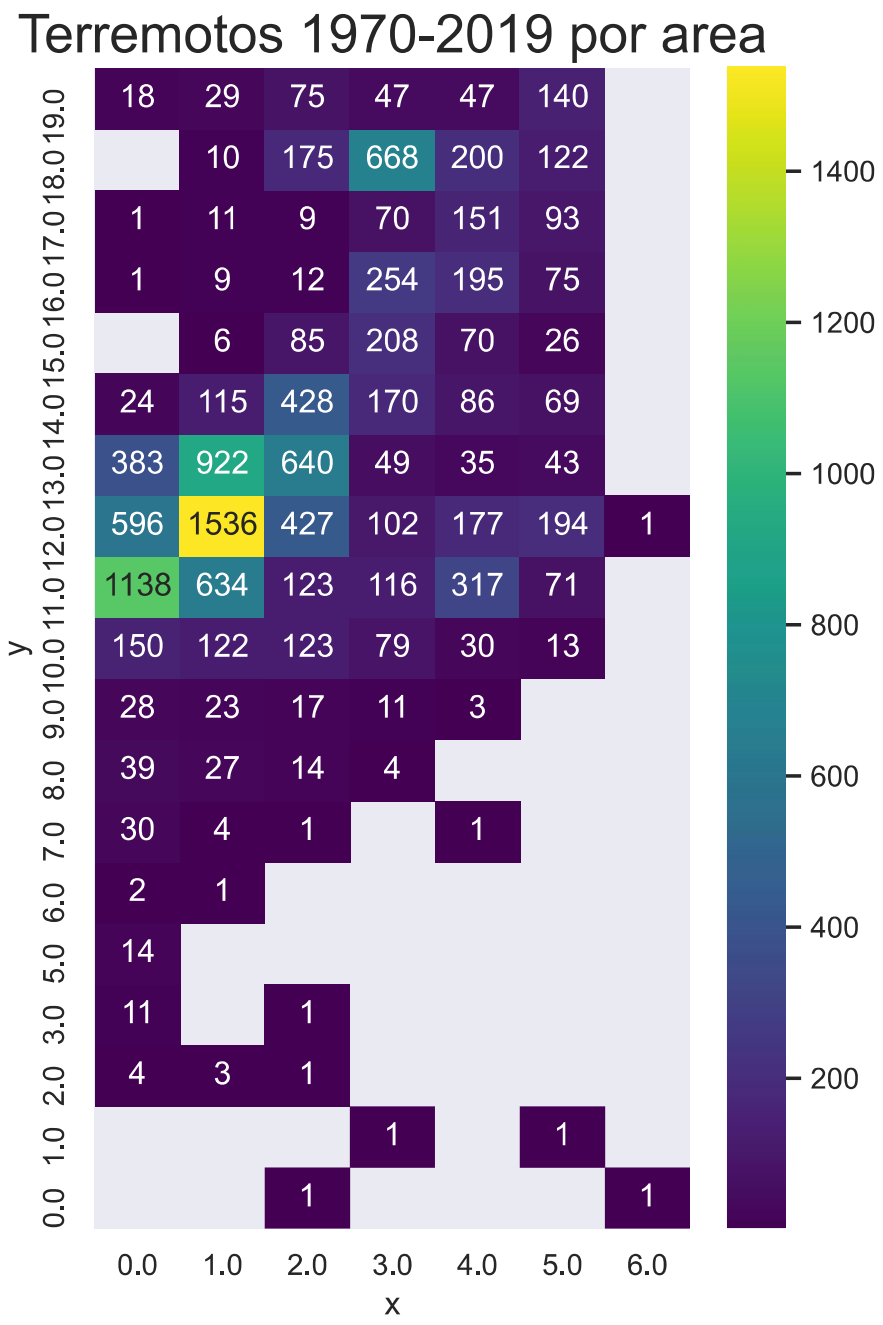
In [198]:
```
#Se cargan los datos a usar

data = pd.read_csv('data_chile.csv', sep=',') #1970-2019
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11963 entries, 0 to 11962
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   time             11963 non-null  object 
 1   latitude         11963 non-null  float64
 2   longitude        11963 non-null  float64
 3   depth            11963 non-null  float64
 4   mag              11963 non-null  float64
 5   magType          11963 non-null  object 
 6   nst              4130 non-null   float64
 7   gap              6138 non-null   float64
 8   dmin             2925 non-null   float64
 9   rms              8450 non-null   float64
 10  net              11963 non-null  object 
 11  id               11963 non-null  object 
 12  place            11934 non-null  object 
 13  horizontalError  2533 non-null   float64
 14  depthError       5344 non-null   float64
 15  magError         3007 non-null   float64
 16  magNst           7110 non-null   float64
 17  tipo sismicidad  11963 non-null  int64  
dtypes: float64(12), int64(1), object(5)
memory usage: 1.6+ MB
```

In [199]:
```
#generamos el area a usar

lat_min = math.floor(data['latitude'].min()) #eje y
lat_max = math.ceil(data['latitude'].max())
q_lat = 2 #si se cambia el la fuente de datos, buscar un numero que se acomod
e.
n_lat = abs(math.ceil(abs(lat_max)-abs(lat_min))/q_lat)

lon_min = math.floor(data['longitude'].min()) #eje x
lon_max = math.ceil(data['longitude'].max())
q_lon = 0.5 #si se cambia el la fuente de datos, buscar un numero que se acomo
de.
n_lon = abs(math.ceil(abs(lon_max)-abs(lon_min))/q_lon)

print(lat_max,lat_min)
print(n_lat)
print(lon_max,lon_min)
print(n_lon)
data['x'] = np.nan
data['y'] = np.nan

for i in range(0,int(n_lat)+1):
    data.loc[(data['latitude']>=lat_min) & (data['latitude']<lat_min+q_lat),
'y'] = i
    for j in range(0,int(n_lon)+1):
        data.loc[(data['y']==i) & (data['longitude']>=lon_min) & (data['longit
ude']< lon_min+q_lon),'x'] = j
        lon_min=lon_min+q_lon
    lon_min=data['longitude'].min()
    lat_min=lat_min+q_lat

data[['y','x']].info()
```

```
-17 -57
20.0
-69 -73
8.0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11963 entries, 0 to 11962
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   y       11963 non-null  float64
 1   x       11963 non-null  float64
dtypes: float64(2)
memory usage: 187.0 KB
```

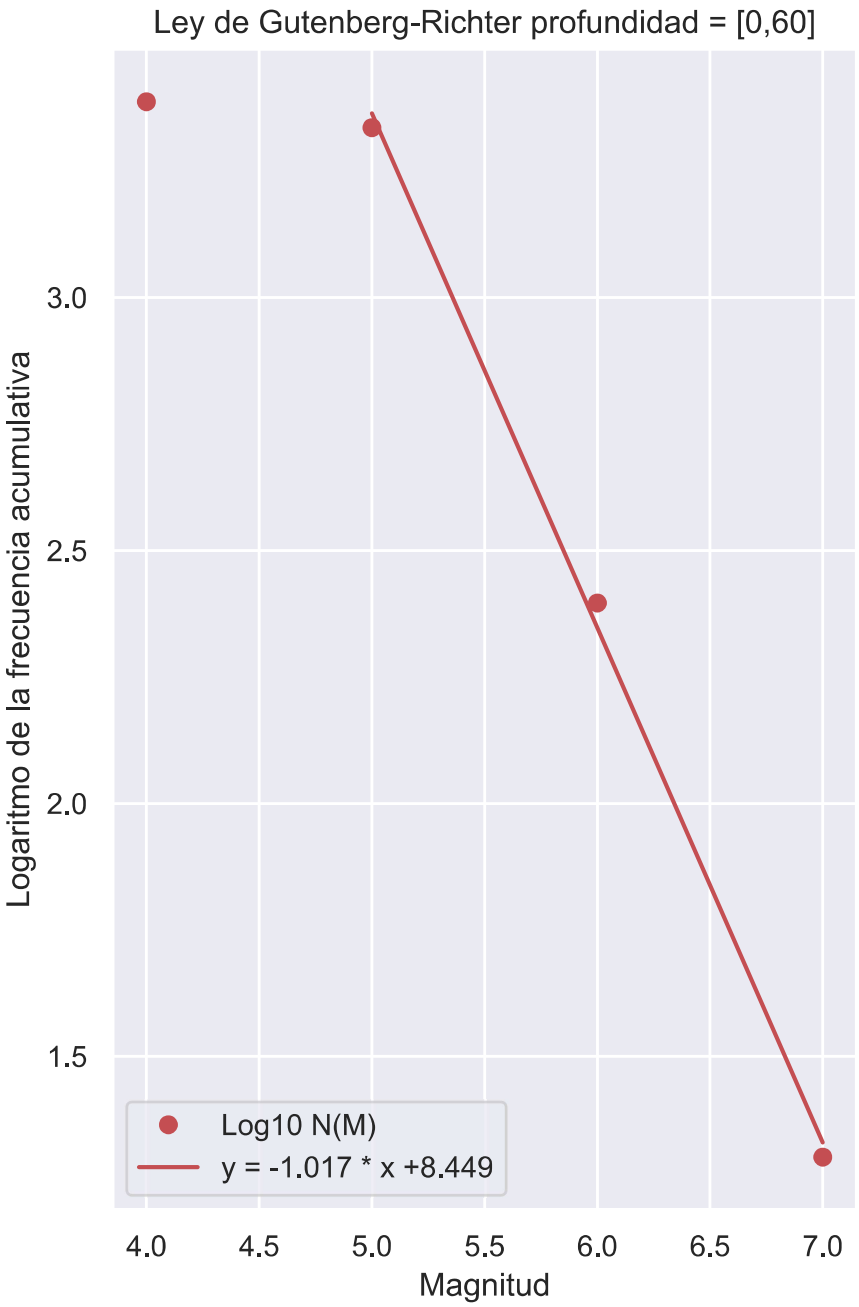
In [200]:
```
# Grafica de los sismos por area, la cual se genero anteriormente.

heat = data.groupby(['x','y']).size().unstack(level=0)
sb.set(rc={'figure.figsize':(5,8)})
ax = sb.heatmap(heat, fmt="g", cmap='viridis', annot=True)
ax.set_title('Terremotos 1970-2019 por area', fontsize =20)
ax.invert_yaxis()
plt.savefig("mapa_por_area.jpg", bbox_inches='tight')
```

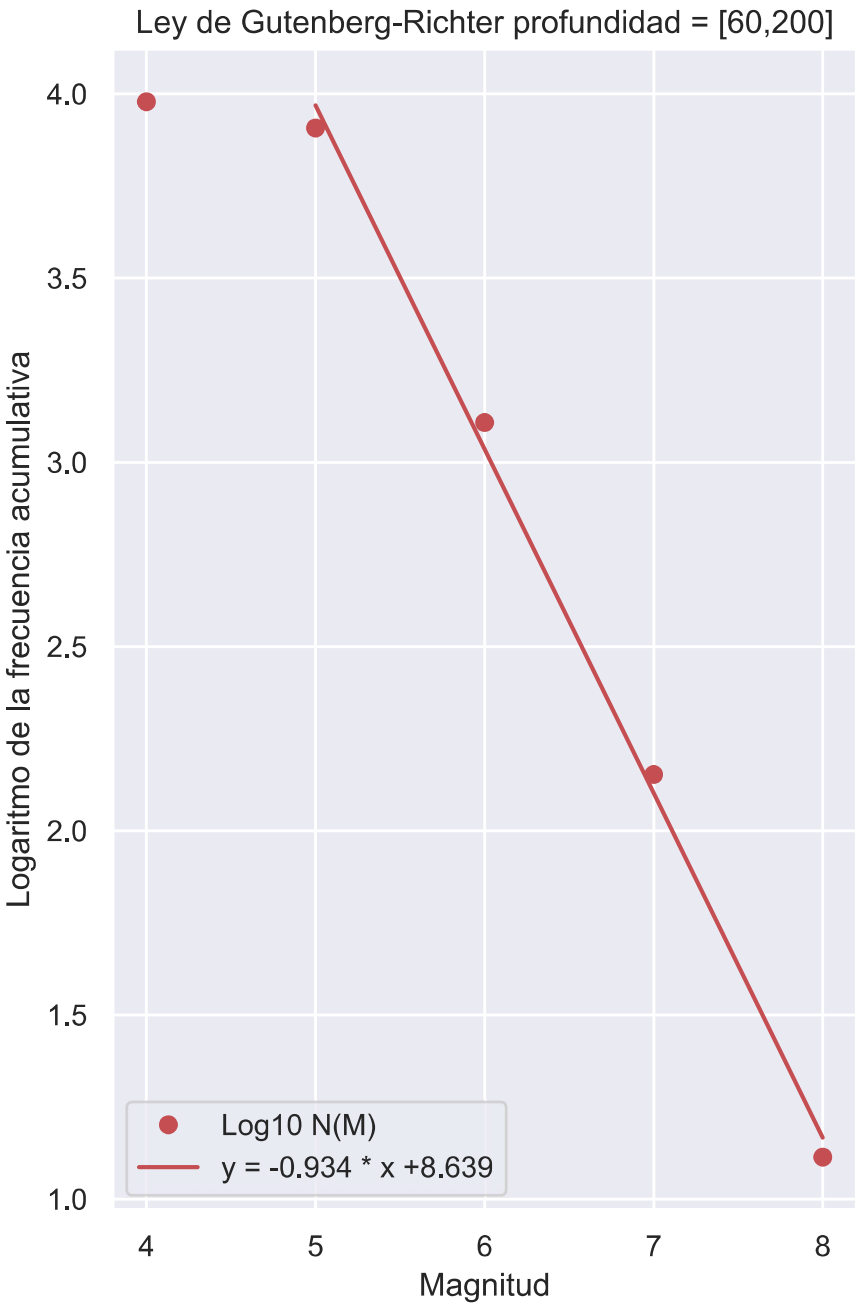


In [201]:
```
#calcular ley de Gutenberg-Richter

n_min=math.floor(data['mag'][(data['tipo sismicidad']==0)].min())
n_max=math.ceil(data['mag'][(data['tipo sismicidad']==0)].max())
n_1=[]
n_0=[]
aux=0
for i in range(n_min,n_max):
    a=data['mag'][(data['tipo sismicidad']==0) & (data['mag']>=i) & (data['ma
g']<i+1)].count()
    n_0.append([a,i])
    aux=a+aux
n_0.append([aux,'total'])

n_min=math.floor(data['mag'][(data['tipo sismicidad']==1)].min())
n_max=math.ceil(data['mag'][(data['tipo sismicidad']==1)].max())
aux=0
for i in range(n_min,n_max):
    a=data['mag'][(data['tipo sismicidad']==1) & (data['mag']>=i) & (data['ma
g']<i+1)].count()
    n_1.append([a,i])
    aux = a+aux
n_1.append([aux,'total'])

m_0=[[n_0[len(n_0)-1][0],n_0[0][1],math.log10( n_0[len(n_0)-1][0] )]]


for i in range(1,len(n_0)-1):
    m_0.append([
        n_0[i-1][0],
        n_0[i][1],
        math.log10( n_0[i-1][0] )
    ])

m_1=[[n_1[len(n_1)-1][0],n_1[0][1],math.log10( n_1[len(n_1)-1][0] )]]


for i in range(1,len(n_1)-1):
    m_1.append([
        n_1[i-1][0],
        n_1[i][1],
        math.log10( n_1[i-1][0] )
    ])


print(m_0,m_1)
```

```
[[2438, 4, 3.387033701282363], [2167, 5, 3.335858911319818], [249, 6, 2.39619
93470957363], [20, 7, 1.3010299956639813]] [[9525, 4, 3.978864984347657], [80
84, 5, 3.9076263048432662], [1282, 6, 3.1078880251827985], [142, 7, 2.1522883
443830563], [13, 8, 1.1139433523068367]]
```

In [202]:
```
x = [i[1] for i in m_0]
y = [i[2] for i in m_0]
z = [round(i,3) for i in np.polyfit(x[1:], y[1:],1)]
z_puntos = [i[1]*z[0] + z[1] for i in m_0[1:]]

z_f ='y = '+str(z[0])+' * x +'+str(z[1])

plt.plot(x,y, 'or',label='Log10 N(M)' )
plt.plot(x[1:],z_puntos,'-r', label=z_f)
plt.xlabel('Magnitud')
plt.ylabel('Logaritmo de la frecuencia acumulativa')
plt.title('Ley de Gutenberg-Richter profundidad = [0,60]')
plt.legend(loc='lower left')
plt.savefig("ley_GR_0_60.jpg", bbox_inches='tight')
plt.show()


x2 = [i[1] for i in m_1]
y2 = [i[2] for i in m_1]
z2 = [round(i,3) for i in np.polyfit(x2[1:], y2[1:],1)]
z2_puntos = [i[1]*z2[0] + z2[1] for i in m_1[1:]]

z2_f ='y = '+str(z2[0])+' * x +'+str(z2[1])

plt.plot(x2,y2, 'or',label='Log10 N(M)' )
plt.plot(x2[1:],z2_puntos,'-r', label=z2_f)
plt.xlabel('Magnitud')
plt.ylabel('Logaritmo de la frecuencia acumulativa')
plt.title('Ley de Gutenberg-Richter profundidad = [60,200]')
plt.legend(loc='lower left')
plt.savefig("ley_GR_60_200.jpg", bbox_inches='tight')
plt.show()
```


Ley de Gutenberg-Richter profundidad = [0,60]
Logaritmo de la frecuencia acumulativa
3.0
2.5
2.0
1.5
Log10 N(M)
y = -1.017 * x +8.449
4.0
4.5
5.0
5.5
6.0
6.5
7.0
Magnitud



In [203]:
```
data['ley GR'] = np.nan
data.loc[data['tipo sismicidad']==0,'ley GR'] = data.loc[data['tipo sismicidad']==0,'mag']*z_puntos[0]+z_puntos[1]
data.loc[data['tipo sismicidad']==1,'ley GR'] = data.loc[data['tipo sismicidad']==1,'mag']*z2_puntos[0]+z2_puntos[1]
```

In [204]:
```
data['ley GR'].describe()
```

Out[204]:
```
count    11963.000000
mean        20.203456
std          2.319550
min         15.803000
25%         18.911000
50%         20.101700
75%         21.292400
max         35.977700
Name: ley GR, dtype: float64
```

In [205]:
```
data.to_csv('data_chile_1970_2019.csv', index=False)
```

In [206]:
```
# Imports necesarios
from sklearn import linear_model
from sklearn import svm
from sklearn.linear_model import Ridge
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from time import time
from sklearn.neural_network import MLPClassifier
```

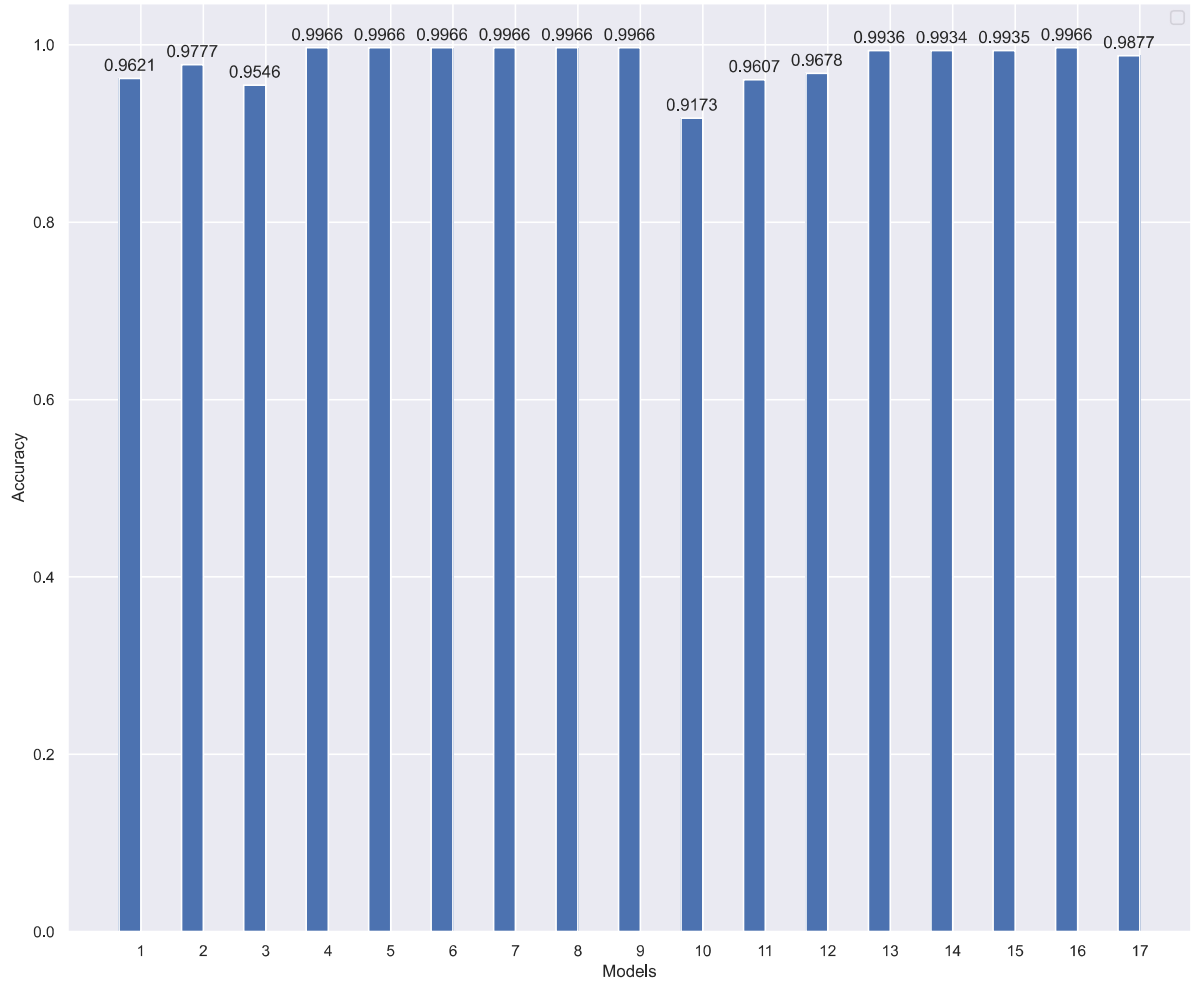
In [207]:
```
# generamos nuestros datos de entrenamiento

X = data[['ley GR','x','y','depth','tipo sismicidad']]
y = data['mag']

lab_enc_train = preprocessing.LabelEncoder()

X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.30, random_state=42)

y_encoded_train = lab_enc_train.fit_transform(y_train)
y_encoded_test = lab_enc_train.fit_transform(y_test)
```

In [208]:
```
# Modelos a utilizar

classifiers = [
    svm.SVR(kernel='rbf'),
    svm.SVR(kernel='linear'),
    svm.SVR(kernel='poly'),
    linear_model.BayesianRidge(n_iter=350),
    linear_model.BayesianRidge(n_iter=700),
    linear_model.BayesianRidge(n_iter=1400),
    linear_model.ARDRegression(n_iter=700),
    linear_model.ARDRegression(n_iter=1700),
    linear_model.ARDRegression(n_iter=3700),
    linear_model.PassiveAggressiveRegressor(max_iter=1000),
    linear_model.PassiveAggressiveRegressor(max_iter=3500),
    linear_model.PassiveAggressiveRegressor(max_iter=7000),
    linear_model.TheilSenRegressor(max_iter=1000),
    linear_model.TheilSenRegressor(max_iter=3500),
    linear_model.TheilSenRegressor(max_iter=7000),
    linear_model.LinearRegression()]
```

In [209]:
```
# Entrenamiento de los modelos anteriores, score y tiempo de entrenamiento

models=[]
for item in classifiers:
    clf = item
    start_time = time()
    clf.fit(X_train, y_train)
    time_model = time() - start_time

    models.append([clf,str(item),time_model,clf.score(X_test,y_test)])


# Modelo MLP, score y tiempo de entrenamiento
clf = MLPClassifier(solver='adam',
                    hidden_layer_sizes=(15, 9),
                    max_iter=5500)
start_time = time()
clf.fit(X_train, y_encoded_train)
time_model = time() - start_time
models.append([clf,str(item),time_model,clf.score(X_test, y_encoded_test)])
```

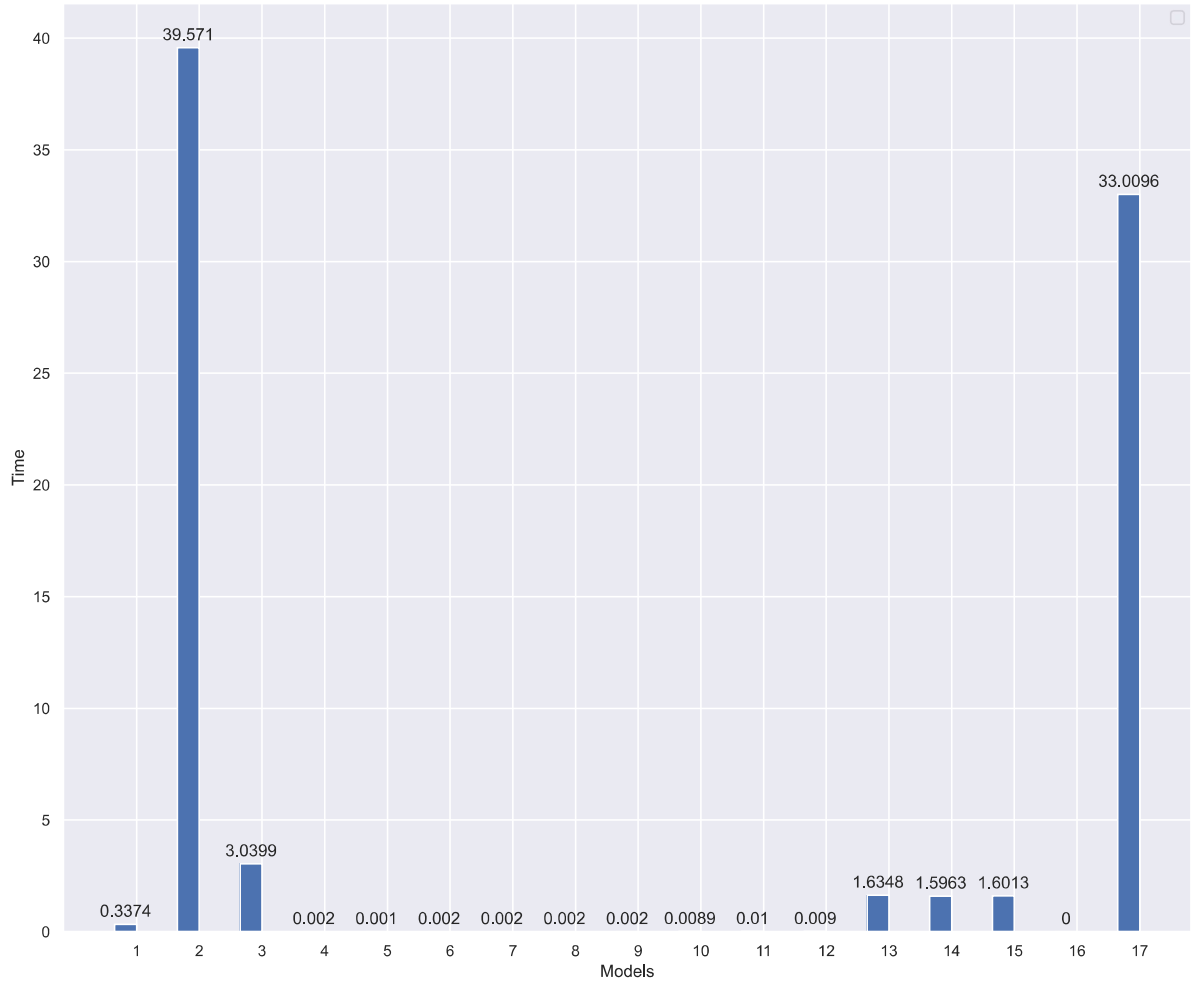
In [210]:
```
acc_relu=[
     round(i[3],4) for i in models
]

acc_lab=[
         i for i in range(1,len(models)+1)
]


""" Plot """
def plot(acc_relu,labels):
  x = np.arange(len(labels))  # the label locations
  width = 0.35  # the width of the bars

  fig, ax = plt.subplots(figsize=(12, 10))
  rects1 = ax.bar(x - width/2, acc_relu, width)

  # Add some text for labels, title and custom x-axis tick labels, etc.
  ax.set_ylabel('Accuracy')
  ax.set_xlabel('Models')
  ax.set_xticks(x)
  ax.set_xticklabels(labels)
  ax.legend()

  ax.bar_label(rects1, padding=3)

  fig.tight_layout()
  plt.savefig("models_scores.jpg", bbox_inches='tight')
  plt.show()
plot(acc_relu,acc_lab)
```

```
No handles with labels found to put in legend.
```



In [211]:
```
acc_relu=[
     round(i[2],4) for i in models
]

acc_lab=[
         i for i in range(1,len(models)+1)
]


""" Plot """
def plot2(acc_relu,labels):
  x = np.arange(len(labels))  # the label locations
  width = 0.35  # the width of the bars

  fig, ax = plt.subplots(figsize=(12, 10))
  rects1 = ax.bar(x - width/2, acc_relu, width)

  # Add some text for labels, title and custom x-axis tick labels, etc.
  ax.set_ylabel('Time')
  ax.set_xlabel('Models')
  ax.set_xticks(x)
  ax.set_xticklabels(labels)
  ax.legend()

  ax.bar_label(rects1, padding=3)

  fig.tight_layout()
  plt.savefig("models_time.jpg", bbox_inches='tight')
  plt.show()
plot2(acc_relu,acc_lab)
```

```
No handles with labels found to put in legend.
```



In [212]:
```
from sklearn.decomposition import PCA
```

In [213]:
```
classifiers_pca = [
    PCA(n_components=i) for i in [2,3,4]
]
```

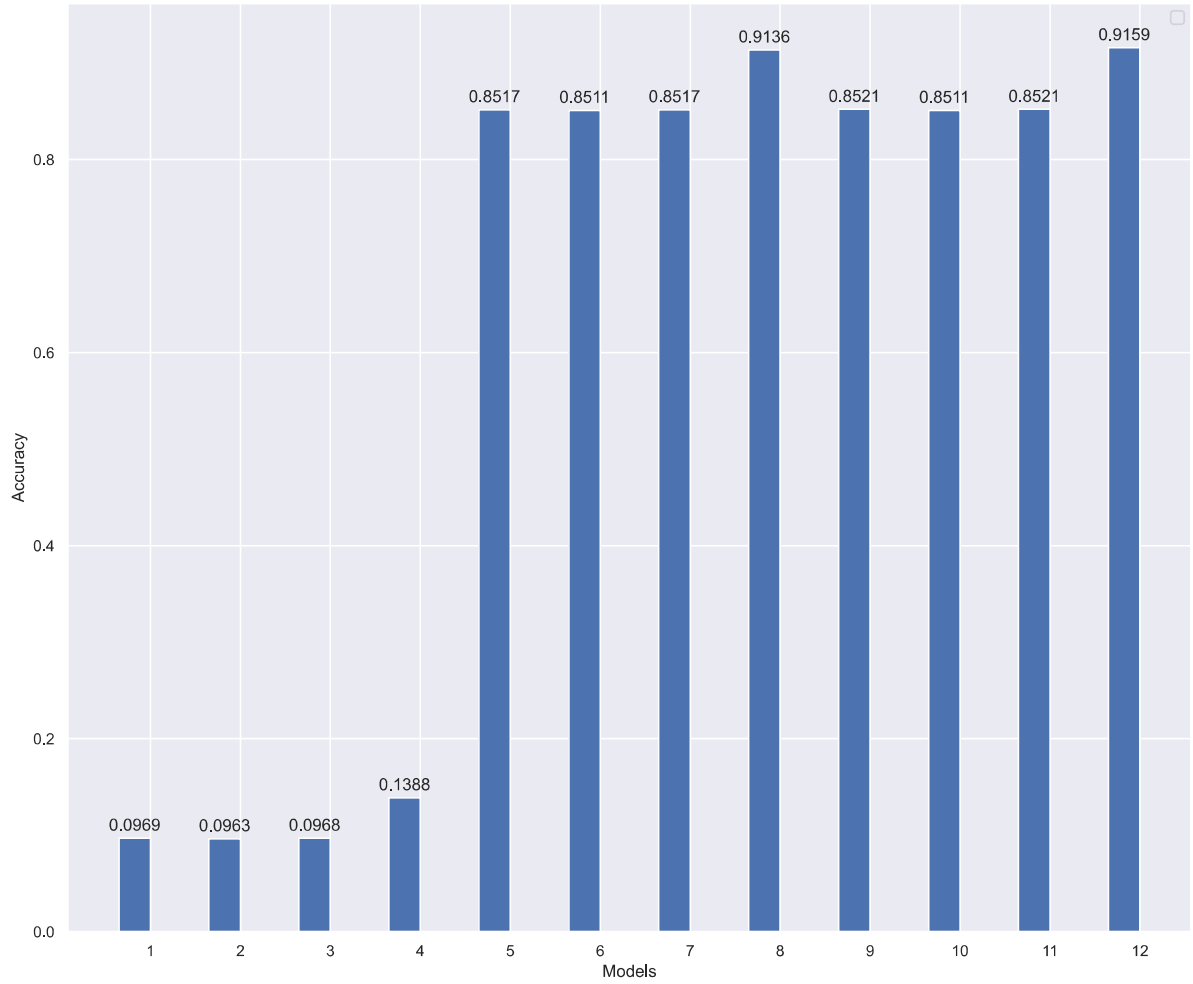
In [214]:
```
models_pca=[]
for item in classifiers_pca:
    clf = item
    start_time = time()
    transform_train=clf.fit_transform(X_train)
    time_model = time() - start_time
    transform_test=clf.fit_transform(X_test)

    models_pca.append([clf,str(item),time_model,clf.score(X_train),transform_t
rain,transform_test])
```

In [215]:
```
classifiers = [
    linear_model.BayesianRidge(n_iter=350),
    linear_model.ARDRegression(n_iter=700),
    linear_model.LinearRegression()]
```

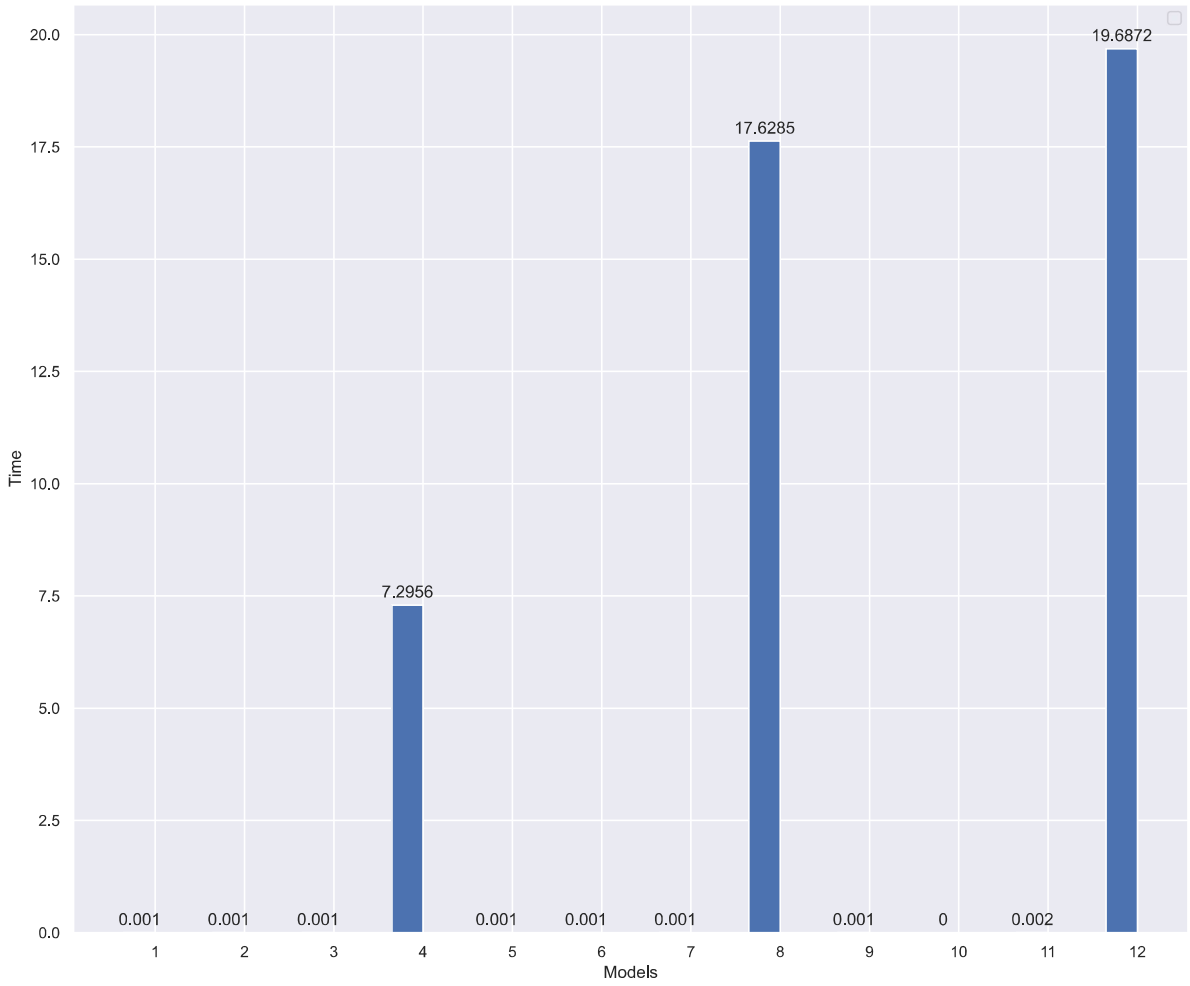
In [216]:
```
# Entrenamiento de los modelos anteriores, score y tiempo de entrenamiento

models1=[]
for i in models_pca:
    for item in classifiers:
        clf = item
        start_time = time()
        clf.fit(i[4], y_train)
        time_model = time() - start_time

        models1.append([clf,str(item),time_model,clf.score(i[5],y_test)])


    # Modelo MLP, score y tiempo de entrenamiento
    clf = MLPClassifier(solver='adam',
                        hidden_layer_sizes=(15, 9),
                        max_iter=5500)
    start_time = time()
    clf.fit(i[4], y_encoded_train)
    time_model = time() - start_time
    models1.append([clf,str(item),time_model,clf.score(i[5], y_encoded_test)])
```

In [217]:
```
acc_relu=[
     round(i[3],4) for i in models1
]

acc_lab=[
         i for i in range(1,len(models1)+1)
]

plot(acc_relu,acc_lab)



acc_relu=[
     round(i[2],4) for i in models1
]

acc_lab=[
         i for i in range(1,len(models1)+1)
]

plot2(acc_relu,acc_lab)
```

```
No handles with labels found to put in legend.
```




```
No handles with labels found to put in legend.
```


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Time
19.6872
17.6285
7.2956
0.001
0.001
0.001
0.001
0.001
0.001
0.001
0
0.002
1
2
3
4
5
6
7
8
9
10
11
12
Models