# Proyecto Final Aprendizaje Automatico

## Prediccion de puntuacion de un producto en la aplicacion Wish

Autores: Daniel López Acero y Antonio Luis Suarez Solis

### ¿En que consite nuestro proyecto?

El objetivo de nuestro proyecto es tratar de predecir si la nota que tendra un producto puesto en venta en la aplicacion Wish sera buena o mala, por medio de la aplicacion de distintas tecnicas de aprendizaje automatico aprendidas y utilizada durante el curso a una base de datos obtenia de Kaggle.com

Bajo nuestro criterio, una buena calificación se encontraria por encima del 3.7 sobre 5, correspondiendo a un 7.5 aproximadamente sobre 10.

Dicha base de datos es esta: https://www.kaggle.com/jmmvutu/summer-products-and-sales-in-ecommerce-wish (https://www.kaggle.com/jmmvutu/summer-products-and-sales-in-ecommerce-wish)

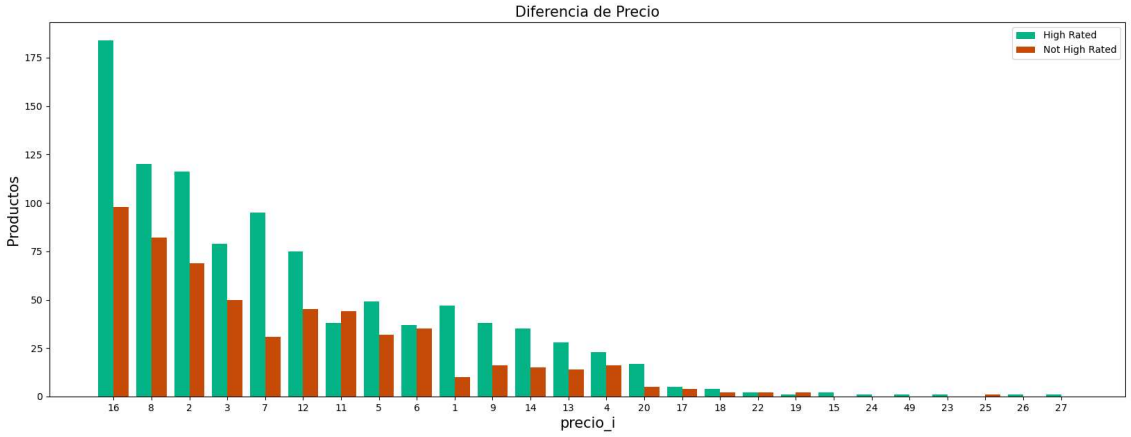
Antes de ponernos a trabjar, nuestro primer objetivo es realizar un analisis a la base de datos y realizar observaciones. Esta base de datos cuenta con muchisimos datos de los cuales utilizaremos los mas apropiados y adecuados para desempeñar nuestro objetivo

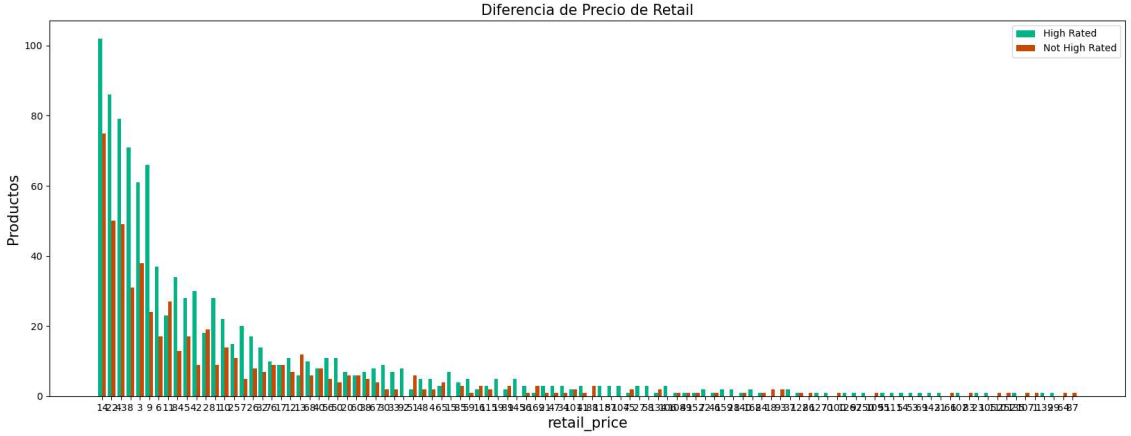
### Datos:

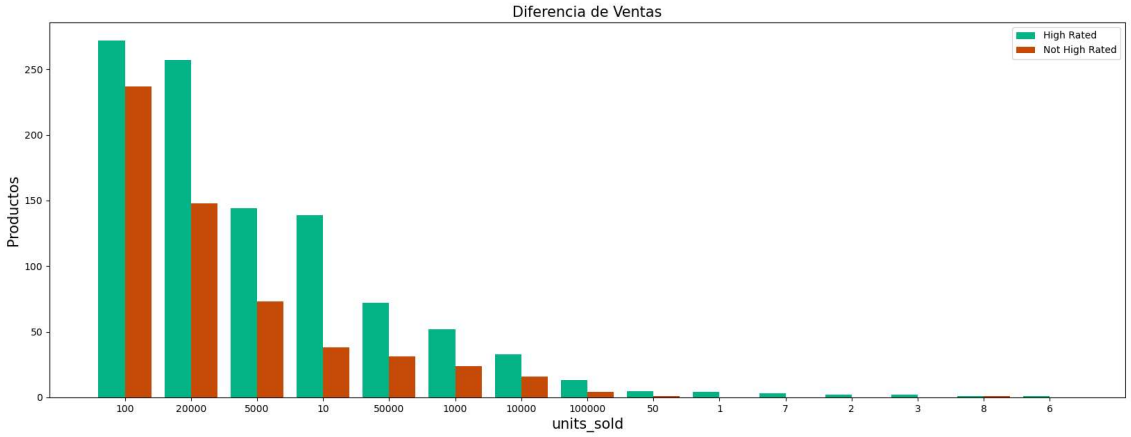
- Price (Precio del producto)
- retail_price (Precio del producto en otras paginas webs)
- units_sold (Cantidad de productos vendidos)
- uses_ad_boosts (Si el usuario ha utilizado un boost de publicidad o no)
- rating (Nota media del producto)
- rating_count (Cantidad de valoraciones del producto)

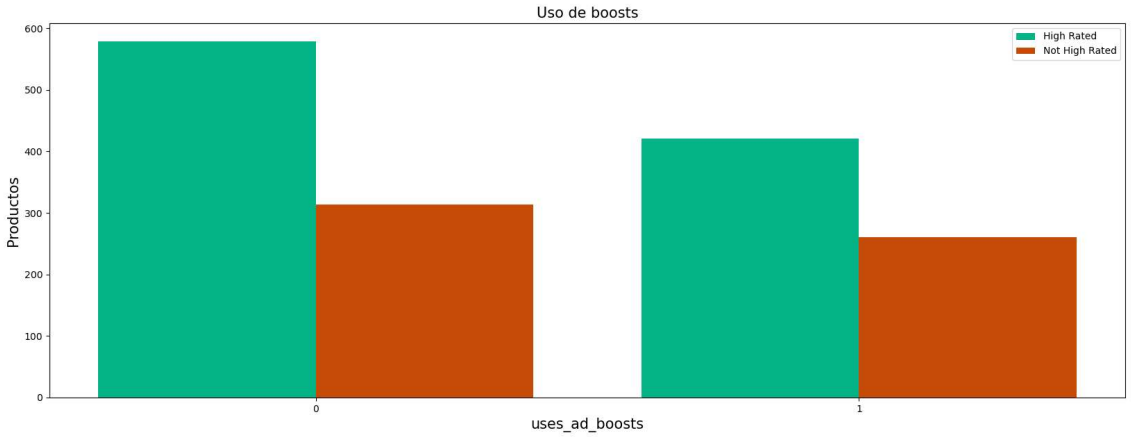
Vistos los datos que queremos utilizar comenzamos comparandolos, para ello lo que hicimos fue dividir los datos entre los que estan bien valorados y los que no lo estan

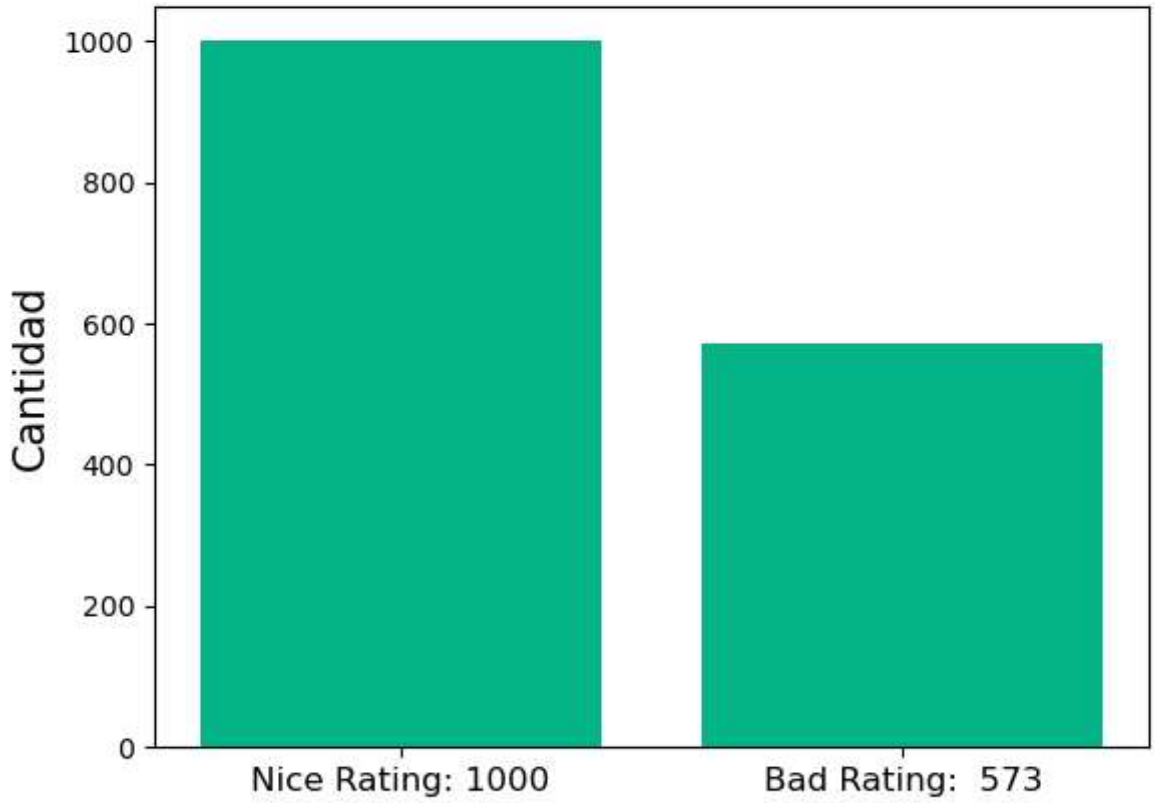
### GRAFICAS COMPARATIVAS


Diferencia de Precio
High Rated
Not High Rated
Productos
precio_i


Diferencia de Precio de Retail
High Rated
Not High Rated
Productos
retail_price


Diferencia de Ventas
High Rated
Not High Rated
Productos
units_sold


Uso de boosts
High Rated
Not High Rated
Productos
uses_ad_boosts




# Imports

In [32]:

```
import pandas as pd
from pandas import DataFrame
import time
import numpy as np
import random
import matplotlib.pyplot as plt
import scipy.optimize as opt
from sklearn import metrics
from sklearn import preprocessing
from scipy.optimize import minimize as sciMin
from scipy.io import loadmat
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn import svm
```

# Lectura de datos y descarte de campos innecesarios

In [33]:

```
data = pd.read_csv('summer-products-with-rating-and-performance_2020-08.csv')
data.drop(labels = ["title","title_orig","currency_buyer", 'rating_five_count', 'rating_fou

def rating_bynary_trans(rate): # Funcion que establece la división entre una buena puntuaci
    if rate < 3.7:
        return 0
    else:
        return 1

data["NiceRating"] = data["rating"].map(rating_bynary_trans)
data = data.reset_index(drop=True)

models_times = np.arange(4, dtype = float)


i = 0
```

# Funciones útiles para el tratamiento de datos

In [34]:

```
def graficaComparativa(campo,titulo,ticklabels): # Función que crea una gráfica de 'x' camp
    global i
    cantidades = data[campo].value_counts()
    lables = cantidades.axes[0].tolist()
    index = np.arange(len(lables))

    badRating = []
    niceRating = []

    for shapes in lables:
        quantity = len(data[data[campo] == shapes].index)
        highRated = len(data[(data[campo] == shapes) & (data['NiceRating'] == 1)].index)
        niceRating.append(highRated)
        badRating.append(quantity-highRated)

    ancho = 0.4
    fig, ax = plt.subplots(figsize=(20,7))
    ax.bar(index, niceRating, ancho, color='#04B486')
    ax.bar(index+ancho, badRating, ancho, color='#C74B08')
    ax.set_xlabel(campo, fontsize = 15)
    ax.set_ylabel('Productos',fontsize=15)
    ax.set_title(titulo,fontsize=15)
    ax.set_xticks(index+ancho/2)
    ax.set_xticklabels(ticklabels, fontsize=10)
    ax.legend(['High Rated', 'Not High Rated'])
    i = i + 1
    plt.show()
```

In [35]:

```
def comparacion(): # Funcion que crea una gráfica comparando las puntuaciones respecto a si
    NiceRating = []
    BadRating = []

    for cl in data["NiceRating"]:
        if cl==0:
            BadRating.append(cl)
        else:
            NiceRating.append(cl)

    xBars = ['Nice Rating: ' + str(len(NiceRating)), 'Bad Rating:  ' + str(len(BadRating))]
    ancho = 0.8
    fig, ax = plt.subplots(figsize=(8,7))
    index = np.arange(len(xBars))
    plt.bar(index, [len(NiceRating), len(BadRating)], ancho, color='blue')
    plt.ylabel('Cantidad', fontsize=15)
    plt.xticks(index, xBars, fontsize=12, rotation=30)
    plt.show()



def tranforData(d): # Funcion que redondea los valores de ciertos campos a números enteros.
    d["precio_i"] = d["price"].astype(int)
    d["rating_i"] = d["rating"].astype(int)
    d["ratingcount_i"] = d["rating_count"].astype(int)
    d["retail_price_i"] = d["retail_price"].astype(int)
    return d

data = tranforData(data)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1573 entries, 0 to 1572
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   price           1573 non-null   float64
 1   retail_price    1573 non-null   int64
 2   units_sold      1573 non-null   int64
 3   uses_ad_boosts  1573 non-null   int64
 4   rating          1573 non-null   float64
 5   rating_count    1573 non-null   int64
 6   NiceRating      1573 non-null   int64
 7   precio_i        1573 non-null   int32
 8   rating_i        1573 non-null   int32
 9   ratingcount_i   1573 non-null   int32
 10  retail_price_i  1573 non-null   int32
dtypes: float64(2), int32(4), int64(5)
memory usage: 110.7 KB
```

In [36]:

```
# Llamada a las graficas mostradas anteriormente para su salida en ejecución.



#graficaComparativa('precio_i', 'Diferencia de Precio',data["precio_i"].unique())
#graficaComparativa('retail_price', 'Diferencia de Precio de Retail',data["retail_price"].u
#graficaComparativa('units_sold', 'Diferencia de Ventas',data["units_sold"].unique())
#graficaComparativa('ratingcount_i', 'Diferencia de numero de ratings',data["ratingcount_i"
#graficaComparativa('uses_ad_boosts', 'Uso de boosts',data["uses_ad_boosts"])
```

In [37]:

```
# LLamada a comparación para mostrar la grafica durante la ejecución

#comparacion()
```

## Preparación de datos para tratamiento

In [38]:

```
dataRate = data['NiceRating']
dataRating = DataFrame(dataRate)
dataRating.columns = ['NiceRating']
#eliminamos las....
data.drop(labels = ['precio_i','rating_i','ratingcount_i','retail_price_i','uses_ad_boosts'
                    'rating','rating_count','NiceRating'], axis = 1, inplace = True)


labelEncode = preprocessing.LabelEncoder()

YArr = labelEncode.fit_transform(dataRate.values.ravel())
#print((YArr))

#data.drop(labels = ['NiceRating','rating_i',], axis = 1, inplace = True)

dataFeat = DataFrame(data)

#XArr = pd.get_dummies(dataFeat).values
XArr = pd.get_dummies(dataFeat.astype(str)).values
```

## Regresión logística

In [39]:

```
#Regresion Logistica:
m = len(YArr)

def sigmoide(value):
    s = 1/(1+np.exp(-value))
    return s

#FUNCIÓN DE COSTE
def coste(O, X, Y):
    H = sigmoide(np.dot(X,O))
    logH = np.log(H)
    logHT = logH.T
    logAux = np.log((1- H))
    logAuxT = logAux.T
    YT = Y.T
    suma = (-1/m)* (np.dot(YT, logH) + np.dot((1-YT), logAux))
    return suma

#FUNCIÓN DE GRADIENTE
def gradiente(O, X, Y):
    return (X.T.dot((sigmoide(X.dot(O))) - Y))/m

#FUNCIÓN DE COSTE REGULARIZADA (lambda)
def coste2(O, X, Y, lam):
    sol = (coste(O, X, Y) + (lam/(2*m))*(O**2).sum())
    return sol

#FUNCIÓN DE GRADIENTE REGULARIZADA (lambda)
def gradiente2(O, X, Y, lam):
    AuxO = np.hstack([np.zeros([1]), O[1:,]])
    return (((X.T.dot(sigmoide(X.dot(O))-Y))/m) + (lam/m)*O)

X = XArr.copy()
X = np.insert(X, 0, 1, axis = 1)


#Utilizando las funciones descritas encima y cronometrando el tiempo para la eficiencia, ll
# de gradiente regularizada para obtener las thetas optimas.

start = time.time()
thetas = np.ones(len(X[0]))
result = opt.fmin_tnc(func = coste2, x0 = thetas, fprime = gradiente2, args = (X, YArr, 0.1
thetas_opt = result[0]
end = time.time()
print("EXE TIME:", end - start, "seconds")
print("OPT THETAS:\n", thetas_opt)


#Evaluación de los resultados obtenidos en las predicciones con las thetas óptimas
def evalua(thetas, X, y):
    thetasMat = np.matrix(thetas)
    z = np.dot(thetasMat,X.transpose())
    resultados = sigmoide(z)
    resultados[resultados >= 0.5] = 1
    resultados[resultados < 0.5] = 0
    admitidosPred = sum(np.where(resultados == y)).shape[0]
    return (admitidosPred / len(y)) * 100
```

```
prediction = evalua(thetas_opt, X, YArr)
models_times[0] = (prediction)
print(models_times)
print("PREDICTIONS RESULT:",prediction)
```

```
EXE TIME: 1.4590749740600586 seconds
OPT THETAS:
 [ 1.50981381 -0.40254397 -2.90873237  1.5619196  -2.06145513 -1.19288781
  1.23550966  0.70531971  2.72446686  1.24098618 -3.17774423  1.24098618
  0.66031073 -0.94761631 -2.06145513  2.2179489  -0.59406783  2.13569302
 -2.06145513 -1.13324254  0.14348762 -0.17967897  0.61711742  1.6014568
  0.67244962 -0.05788682  1.83811698 -0.49914241 -0.01364691 -0.58376136
 -0.44128297 -1.27944777  2.02039435 -0.29411776  0.89820758  0.03053103
 -2.36346953  1.13063379  1.39530149 -2.63844026 -1.3177004  -2.19405054
  2.07666513  1.55256653 -0.97772668 -0.46951678  0.71308237  1.28700871
 -2.86367779  2.56559448  0.32890033 -0.08701279 -0.15588168 -0.80439066
 -0.06655869  2.60617893 -1.02561654 -4.016718   -1.09552321  1.46993219
  1.69268907  0.58134547 -2.64509262 -1.472985    0.74279623 -0.56724612
 -0.81544157 -1.472985   -0.62507327 -0.72452566 -0.2396794  -2.86329798
 -2.12409205  1.79948334  2.19583243  0.01468804  2.14750478  0.56088363
  1.3805279  -0.07898288  0.06346918 -1.40429118  1.69260841  1.83537636
  2.01415792  2.49639891  2.26394328 -3.08066206  1.41435727  1.75496039
  1.87422334  1.87605201  0.16975026  0.79070483  1.72044683  0.28806983
  0.28986555 -0.48367406 -3.89831989  1.55514487  0.23618943 -2.01156591
  2.06536848 -1.8092418   0.14736355 -1.37886849  0.49180628  0.18646302
 -0.25805698 -0.96272883  0.38115561 -2.27050643  1.7271508   1.72275277
  0.32854351 -3.32622062  2.29028427  2.08158878  1.05499449 -1.30684786
 -3.33637746 -2.12409205  1.46400574 -1.84130739 -0.15671974 -0.16779775
  0.39811508  0.57461727 -1.44330633 -0.17328914 -1.5683287   1.0161145
  2.39737815 -2.67482675  1.17773817  1.86854809  0.62145089 -0.08634329
  1.0161145  -0.20222768  1.26359523  1.39058986 -0.594871    1.78759365
 -0.71743872  1.06125751  0.78915045 -2.36378394  0.35789743 -2.26655782
 -2.0164342   0.35604616  0.64115086  1.73027486  1.006465    1.68630898
 -0.77227347  1.72579091 -1.35344746 -0.2577292  -1.56777687 -1.64611663
  0.43867667  2.0638834   0.34502059  0.64734256 -2.06760123 -0.82130116
  0.08515608  1.8304347   0.60583953 -0.89816041 -2.23715712  1.42523653
 -0.17248512  0.31438959 -1.17138763  1.34597832  0.07535445 -1.55735395
  2.10899686 -0.65626403 -0.91453944  0.27412964  0.89537431 -1.26808482
  0.72558692 -0.77392106 -1.11529912 -0.36188132 -0.21547465 -0.72490144
  1.27908143  2.31887965  2.61454355  0.93015002  0.32321216 -0.1780877
 -0.31193963  2.21630508 -0.95342807  0.86082042 -0.61949931 -2.28307868
 -0.6517167   0.13493617  0.2864052  -2.74717848 -2.75530854  2.45656974
  1.19942254 -0.79616226 -0.76128283 -1.55659641 -0.70282166 -2.30860556
  1.94846053 -0.14446506 -0.44018678 -0.09564258  0.39801604 -0.67686784
 -0.73862347  0.21550756  1.57124491  1.01703805 -0.34074752 -1.44788926
  0.23311557 -1.02525395  1.88950499  0.78915047  2.34227842 -0.54813263
 -1.4439294  -1.01920456 -0.16881382  1.12875081 -1.11264753 -0.66012117
  1.10075286 -0.28502229 -0.83806106 -0.22641441  0.73829177  0.78607362
  1.701287  ]
[72.53655435  1.          2.          3.        ]
PREDICTIONS RESULT: 72.53655435473617
```

## Redes Neuronales

In [40]:

```
lambda_ = 1
def addColumn(mat):
    return np.hstack([np.ones([np.shape(mat)[0],1]), mat])

def sig_derivated(value):
    return sigmoide(value) * (1- sigmoide(value))

def random_weights(_in, _out):
    weights = np.random.uniform(-0.12,0.12, (_out, 1+_in))
    return weights

def neuralCosteReg(_x, _y, a3, numTags, th1, th2):
    m = np.shape(_x)[0]
    aux1 = -_y*(np.log(a3))
    aux2 = (1-_y) * (np.log(1-a3))
    aux3 = aux1-aux2
    aux4 = np.sum(th1**2) + np.sum(th2**2)
    return (1/m)* np.sum(aux3) + (lambda_/(2*m)) * aux4
```

In [41]:

```
def forward_prop(_x, th1, th2): #Funcion para el calculo de h = h(subtheta) (x(i)).
    m = np.shape(_x)[0]
    a1 = np.hstack([np.ones([m,1]), _x])
    z2 = np.dot(a1, th1.T)
    a2 = np.hstack([np.ones([m,1]), sigmoide(z2)])
    z3 = np.dot(a2, th2.T)
    h = sigmoide(z3)
    return h, a1, z2, a2,z3

def prop(a1, th1, th2):
    a1 = addColumn(a1)
    a2 = sigmoide(np.dot(a1, np.transpose(th1)))
    a2 = addColumn(a2)
    a3 = sigmoide(np.dot(a2, np.transpose(th2)))
    return a1,a2,a3
```

In [42]:

```
def back_prop(params_rn, num_entradas, num_ocultas, numTags, _x, _y, _reg): #Calculo del gr
    thet1 = np.reshape(params_rn[:num_ocultas*(num_entradas + 1)],(num_ocultas, (num_entrad
    thet2 = np.reshape(params_rn[num_ocultas*(num_entradas + 1):],(numTags, (num_ocultas +

    a1, a2, a3 = prop(_x, thet1, thet2)
    m = np.shape(_x)[0]
    delta3 = a3 - _y

    delta_mat1 = np.zeros(np.shape(thet1))
    delta_mat2 = np.zeros(np.shape(thet2))
    aux1 = np.dot(delta3, thet2)
    aux2 = addColumn(sig_derivated(np.dot(a1, np.transpose(thet1))))
    delta2 = aux1*aux2
    delta2 = np.delete(delta2, [0], axis = 1)

    delta_mat1 = delta_mat1+ np.transpose(np.dot(np.transpose(a1), delta2))
    delta_mat2 = delta_mat2 + np.transpose(np.dot(np.transpose(a2), delta3))

    delta_mat1  = (1/m)*delta_mat1
    delta_mat1[:, 1:] = delta_mat1[:, 1:] + (_reg/m)*thet1[:, 1:]

    delta_mat2  = (1/m)*delta_mat2
    delta_mat2[:, 1:] = delta_mat2[:, 1:] + (_reg/m)*thet2[:, 1:]
    coste = neuralCosteReg(_x, _y,a3,  numTags, thet1,thet2)
    gradient = np.concatenate((np.ravel(delta_mat1), np.ravel(delta_mat2)))
    return coste, gradient
```

In [43]:

```
def evaluateLearning(_y, _out): #Calculo de aciertos
    checker = (_out >0.7)
    count = np.size(np.where(checker[:, 0] == _y[:, 0]))
    fin = count/np.shape(_y)[0]*100
    return fin, checker
def NeuralNet(_data): #Función principal para llamar a la red neuronal.
    _X = XArr.copy()
    _y = YArr.copy()

    _y = np.reshape(_y, (np.shape(_y)[0], 1))

    num_entradas = np.shape(_X)[1]
    num_ocultas = 25
    numTags = 1

    the1 = random_weights(num_entradas, num_ocultas)
    the2 = random_weights(num_ocultas, numTags)

    theta_vector = np.concatenate((np.ravel(the1), np.ravel(the2)))
    start = time.time()
    thetas = sciMin(fun = back_prop, x0 =theta_vector, args = (num_entradas, num_ocultas, n
    method = 'TNC', jac = True, options = {'maxiter': 70}).x
    end = time.time()

    the1 = np.reshape(thetas[:num_ocultas*(num_entradas+1)], (num_ocultas, (num_entradas+1)
    the2 = np.reshape(thetas[num_ocultas* (num_entradas+1):], (numTags, (num_ocultas+1)))

    a,c = evaluateLearning(_y, forward_prop(_X, the1, the2)[0])

    print("-----------------------------------------------\n")
    print("\n TRAINING EXECUTION TIME:", end - start, "seconds")

    print("PREDICTION NEURAL_NETWORK: " + str(a) + " %")
    models_times[1] = a
    print("-----------------------------------------------\n")
NeuralNet(data)
```

```
-----------------------------------------------


 TRAINING EXECUTION TIME: 1.0976526737213135 seconds
PREDICTION NEURAL_NETWORK: 64.52638270820088 %
-----------------------------------------------
```

## SVM

In [47]:

```
#lineal
XSvm = XArr.copy()
YSvm = YArr.copy()

X_Train, X_test, Y_Train, Y_test = train_test_split(XSvm, YSvm, test_size=0.33, random_stat

#SVM de tipo lineal
Coef = 1.0
svmLineal = svm.SVC(kernel='linear',C=Coef)

#Entrenamiento de las "redes". Similar a buscar las thetas óptimas
start = time.time()
svmFitted = svmLineal.fit(X_Train, Y_Train)
end = time.time()
print("\nTRAINING EXECUTION TIME:", end - start, "seconds")
#predecimos Y a partir de la x "entrenada"
predictY = svmLineal.predict(X_test)

def evalua(results, Y):
    numAciertos = 0
    for i in range(len(Y_test)):
        if results[i] == Y[i]: numAciertos += 1
    return (numAciertos/(len(Y_test)))*100

success = evalua(predictY, Y_test)
models_times[2] = success
print("\nPREDICTIONS SVM LINEAL",success)
```

```
TRAINING EXECUTION TIME: 0.7210719585418701 seconds

PREDICTIONS SVM LINEAL 59.80769230769231
```

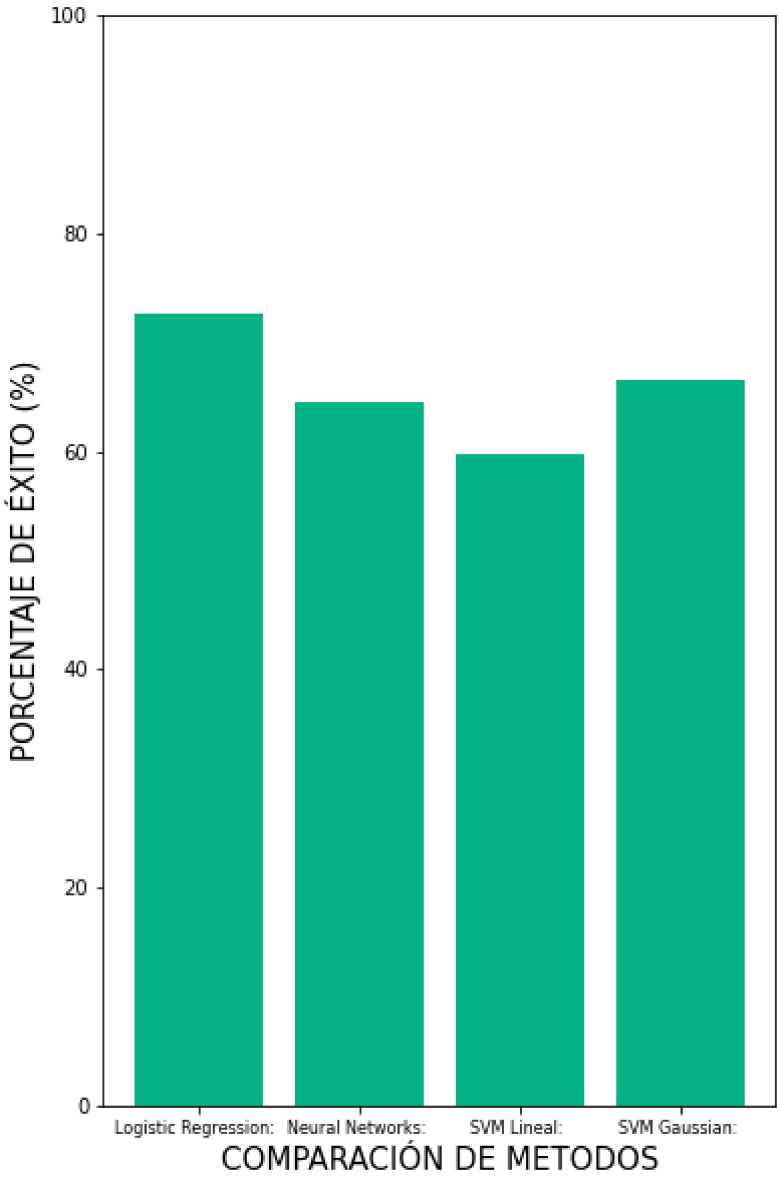
In [51]:

```
#gausiano
XSvmG = XArr.copy()
YSvmG = YArr.copy()

X_Train, X_test, Y_Train, Y_test = train_test_split(XSvmG, YSvmG, test_size=0.33, random_st

Coef=1
sigma = 0.1
svmGauss = svm.SVC(C = Coef, kernel = 'rbf', gamma = 1/(2*sigma **2))

start = time.time()
svmGaussFitted = svmGauss.fit(X_Train, Y_Train)
end = time.time()
print("\nTRAINING EXECUTION TIME:", end - start, "seconds")
predictY = svmGauss.predict(X_test)

def evalua(results, Y):
    numAciertos = 0
    for i in range(len(Y_test)):
        if results[i] == Y[i]: numAciertos += 1
    return (numAciertos/(len(Y_test)))*100

success = evalua(predictY, Y_test)
models_times[3] = success
print("\nPREDICTIONS SVM GAUSS",success)
```

```
TRAINING EXECUTION TIME: 0.8068490028381348 seconds

PREDICTIONS SVM GAUSS 66.53846153846153
```

In [46]:

```
#comparacion final
xBars = ['Logistic Regression: ',
         'Neural Networks: ' ,
         'SVM Lineal: ' ,
         'SVM Gaussian: ']
ancho = 0.8
fig, ax = plt.subplots(figsize=(6,10))
index = np.arange(len(xBars))
plt.bar(index, [models_times[0], models_times[1], models_times[2], models_times[3]], ancho,
plt.xlabel('COMPARACIÓN DE METODOS', fontsize=15)
plt.ylabel('PORCENTAJE DE ÉXITO (%)', fontsize=15)
plt.xticks(index, xBars, fontsize=8, rotation=0)
plt.ylim((0, 100))
plt.show()
```



# Analisis:

## Regresión logística:

Basandonos en lo explicado en clase, hemos calculdo las thetas óptimas para llevar acabo las predicciones. Su porcentaje de acierto es de 72,5%. Podemos concluir que la predicción no es especialmente fiable.

Resultado: 72,53% en 1,69 s

## Red Neuronal:

Hemos implementado metodos para el calculo de la derivada de la función sigmoide, un algoritmo de propagación hacia atras para calcular el gradiente, obteniendo asi las thetas optimas. AL contrario de lo que cabría esperar, su porcentaje de exito es de 62%, por lo que deja mucho que desear como predictor.

Resultado: 62,74% en 1.06 s

## SVM:

Hemos usado la libreria sklearn tal y como se especificó en la asignatura.

### Linear:

Resultados: 59,8% en 0,72 s

### Gaussiano:

Resultados: 66,53% en 0,80 s

# Conclusiones:

Una vez obtenidos los datos y con las gráficas comparativas a disposición, llegamos a la conclusión de que aun asumiendo que este tipo de enfoques e información a nuestro alcance pueden no ser los ideales para analizar este campo creemos que los resultados obtenidos por los predictores dejan mucho que desear y podrian ser mejorados en gran medida. No creemos que el tema a analizar sea uno sencillo en vista del abanico de opiniones y gustos que uno puede encontrar en la gente, incrementado mas si cabe en el ambito de la ropa.

Si tuviesemos que aventurarnos a decir que campo de los usados en el proyecto tiene un mayor peso en la calificación nos debatimos entre la relación entre el precio y el retail price(precio en otras paginas/tiendas) o las unidades vendidas.

Para finalizar, si bien la Regresión Logística es el método que mayor acierto nos proporciona, es seguido de cerca por las SVM(Gauss) con solo un 6% menos de acierto y una reducción en el tiempo del 52.66%.