**SPAM E-mail Database**

**1) Las dimensiones del conjunto de datos.**

Es un dataset multivariado de clasificación. Sus atributos son números reales y enteros.

Tiene 4601 registros y 57 atributos

El dataset de Spam está formado por mensajes de email y categoriza los mensajes en Spam(1) o no Spam.

Los atributos del 1 al 55  indican la frecuencia de una palabra o carácter particular  en el  mensaje, y son valores reales comprendidos entre el 1 y el 100.

Las primeras 48 indican la frecuencia de la palabra en el texto.

Las 6 siguientes indican el porcentaje de caracteres en el mail que coinciden con uno concreto, es decir = 100 \* numero de repeticiones de un carácter/total de caracteres en el mail.

Los atributos del 55 al 57 miden la longitud de las secuencias de letras mayúsculas consecutivas. Son números reales del 1 al 100

55= longitud de media de secuencia ininterrumpida de letras mayúsculas

56= longitud de la mayor secuencia ininterrumpida de letras mayúsculas

57= número total de letras mayúsculas

El target del dataset es binómico, 0 o 1.

La distribución del dataset es la siguiente:

Spam  1813  (39.4%)  
 Non-Spam  2788  (60.6%)  
  
  
Estadísticas de los atributos:  
   Min: Max:   Average:  Std.Dev: Coeff.Var\_%:   
1  0    4.54   0.10455   0.30536  292            
2  0    14.28  0.21301   1.2906   606            
3  0    5.1    0.28066   0.50414  180            
4  0    42.81  0.065425  1.3952   2130           
5  0    10     0.31222   0.67251  215            
6  0    5.88   0.095901  0.27382  286            
7  0    7.27   0.11421   0.39144  343            
8  0    11.11  0.10529   0.40107  381            
9  0    5.26   0.090067  0.27862  309            
10 0    18.18  0.23941   0.64476  269            
11 0    2.61   0.059824  0.20154  337            
12 0    9.67   0.5417    0.8617   159            
13 0    5.55   0.09393   0.30104  320            
14 0    10     0.058626  0.33518  572            
15 0    4.41   0.049205  0.25884  526            
16 0    20     0.24885   0.82579  332            
17 0    7.14   0.14259   0.44406  311            
18 0    9.09   0.18474   0.53112  287            
19 0    18.75  1.6621    1.7755   107            
20 0    18.18  0.085577  0.50977  596            
21 0    11.11  0.80976   1.2008   148            
22 0    17.1   0.1212    1.0258   846            
23 0    5.45   0.10165   0.35029  345            
24 0    12.5   0.094269  0.44264  470            
25 0    20.83  0.5495    1.6713   304            
26 0    16.66  0.26538   0.88696  334            
27 0    33.33  0.7673    3.3673   439            
28 0    9.09   0.12484   0.53858  431            
29 0    14.28  0.098915  0.59333  600            
30 0    5.88   0.10285   0.45668  444            
31 0    12.5   0.064753  0.40339  623            
32 0    4.76   0.047048  0.32856  698            
33 0    18.18  0.097229  0.55591  572            
34 0    4.76   0.047835  0.32945  689            
35 0    20     0.10541   0.53226  505            
36 0    7.69   0.097477  0.40262  413            
37 0    6.89   0.13695   0.42345  309            
38 0    8.33   0.013201  0.22065  1670           
39 0    11.11  0.078629  0.43467  553            
40 0    4.76   0.064834  0.34992  540            
41 0    7.14   0.043667  0.3612   827            
42 0    14.28  0.13234   0.76682  579            
43 0    3.57   0.046099  0.22381  486            
44 0    20     0.079196  0.62198  785            
45 0    21.42  0.30122   1.0117   336            
46 0    22.05  0.17982   0.91112  507            
47 0    2.17   0.0054445 0.076274 1400           
48 0    10     0.031869  0.28573  897            
49 0    4.385  0.038575  0.24347  631            
50 0    9.752  0.13903   0.27036  194            
51 0    4.081  0.016976  0.10939  644            
52 0    32.478 0.26907   0.81567  303            
53 0    6.003  0.075811  0.24588  324            
54 0    19.829 0.044238  0.42934  971            
55 1    1102.5 5.1915    31.729   611            
56 1    9989   52.173    194.89   374            
57 1    15841  283.29    606.35   214            
58 0    1      0.39404   0.4887   124            
<https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.DOCUMENTATION>

**2) La utilización de diferentes modelos, no sólo lineales, y su explicación dentro del informe (utilización NO CIEGA de la librería).**

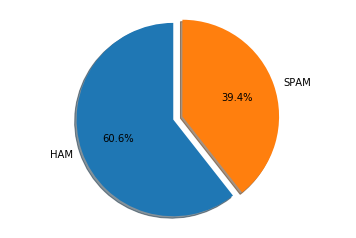
**Näive Bayes Gaussian**

El algoritmo de clasificación Naive Bayes es un clasificador probabilístico. Se basa en modelos de probabilidades que incorporan fuertes suposiciones de independencia.

Esto significa que registra la frecuencia con la que Spam o Ham  aparece junto a un valor de un campo de entrada.

Es un algoritmo de clasificación rapido y simple, suele dar buenos resultados para dataset de grandes dimensiones.

Nuestro dataset es binario, Spam/Ham y tiene el siguiente porcentaje de targets:



Para aplicar el método de GNB lo hacemos en 20 iteraciones.

En cada iteración estratificar el dataset para que haya una proporción fija de Spam/Ham  en la parte de Train y de Test, aplicamos el método GNB, entrenamos el módelo y calculamos la predicción de fallos sobre Ytest e Ytrain.

Después calculamos la media de fallos de todas las iteraciones para ver el resultado.

**CODIGO**

|  |
| --- |
| import matplotlib.pyplot as plt  from sklearn.naive\_bayes import GaussianNB  from sklearn.model\_selection import cross\_val\_score  from sklearn.model\_selection import  train\_test\_split  #importar datos del CSV  data=pd.read\_csv('C:\\Users\\Esther\\Desktop\\Master\\Modulo 8\\SPAM\\spambasedata.csv', header=None)  #data = pd.read\_csv('spambase.data', header=None)  data.rename(columns={57:'is\_spam'}, inplace=True)  ham = data[data['is\_spam'] == 0]  spam = data[data['is\_spam'] == 1]  nospam = data[data['is\_spam'] == 0]  X = data  print (X.shape)  Y = X.pop('is\_spam')  ############PIE CHAR ###################  # Pie chart, where the slices will be ordered and plotted counter-clockwise:  Y0=sum(Y==0)  Y1=sum(Y==1)  labels = 'HAM','SPAM'  sizes = [Y0, Y1]  explode = (0, 0.1)  fig1, ax1 = plt.subplots()  ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',         shadow=True, startangle=90)  ax1.axis('equal')  plt.show()  ################### GNB ############################  #Dividimos el dataset en Train(3/5) y test(2/5) stratificando los datos de Spam y Ham  #Aplico GNB  #REalizo un bucle en el que entreno mi modelo con Xtrain  #y saco una media de errores en cada iteracion comparando Ytest con Ypred  miGNB = GaussianNB()  i = 0  fallos = []  fallostrain = []  for i in range(1,40):     spam\_train, spam\_test = train\_test\_split(spam, train\_size=0.6)     ham\_train, ham\_test = train\_test\_split(ham, train\_size=0.6)     Xtrain = ham\_train.append(spam\_train)   Ytrain = Xtrain.pop('is\_spam')     Xtest = ham\_test.append(spam\_test)     Ytest = Xtest.pop('is\_spam')     Ypred=miGNB.fit(Xtrain, Ytrain).predict(Xtest)     Ypredtrain=miGNB.fit(Xtrain, Ytrain).predict(Xtrain)     fallos.append(sum(Ypred!=Ytest))     fallostrain.append(sum(Ypredtrain!=Ytrain))      print(fallos)  print(fallostrain)  print ("Num. medio de errores de: " + str(100\*np.mean(fallos)/len(Ytest)))  print ("Dev. Std. de errores de: " + str(100\*np.std(fallos)/len(Ytest)))  print ("Num. medio de errores de Train: " + str(100\*np.mean(fallostrain)/len(Ytrain)))  print ("Dev. Std. de errores de train: " + str(100\*np.std(fallostrain)/len(Ytrain))) |

**RESULTADOS**

|  |
| --- |
| **Fallos Test**  [305, 340, 341, 338, 343, 322, 314, 340, 337, 325, 325, 329, 350, 355, 341, 328, 319, 352, 306, 330, 326, 322, 320, 346, 339, 337, 320, 336, 342, 356, 330, 349, 312, 313, 347, 318, 331, 347, 371]  Num. medio de errores de: 18.0990562098  Dev. Std. de errores de: 0.795946323944  **Fallos Train**  [486, 470, 496, 496, 494, 505, 473, 489, 490, 488, 467, 498, 465, 486, 441, 464, 517, 484, 517, 480, 463, 499, 447, 500, 477, 493, 478, 502, 469, 500, 482, 485, 490, 490, 487, 472, 511, 456, 466]  Num. medio de errores de Train: 17.539799816  Dev. Std. de errores de train: 0.631760715085 |

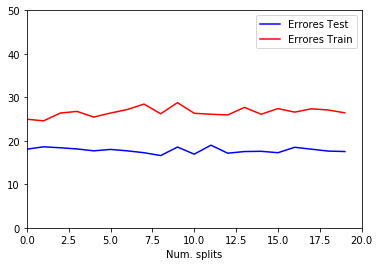
**Grafica con los errores de train y test**

Ploteamos el porcentaje de errores en cada iteración para train y para test.

Los datos de error están expresados en porcentaje.

**CODIGO**

|  |
| --- |
| #Grafica de errores en cada iteracion  #Calculamos el numero de fallos de cada iteracion sobre 100  j=0  d = decimal.Decimal('1.1')  fallos100=np.array([d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d],dtype=np.dtype(decimal.Decimal))  for j in range(0,20):     print[j]     print(fallos[j])     d=((fallos[j]) \* (100)/float(len(Ytest)))     fallos100[j] = d  print (fallos100)  j=0  d = decimal.Decimal('1.1')  fallostrain100=np.array([d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d],dtype=np.dtype(decimal.Decimal))  for j in range(0,20):     print[j]     print(fallostrain[j])     d=((fallostrain[j]) \* (100)/float(len(Ytest)))     fallostrain100[j] = d  print (fallostrain100)  plt.axis([0, 20, 0, 50])  plt.plot(range(0,20),fallos100, 'b-',          label='Errores Test')  plt.plot(range(0,20), fallostrain100, 'r-',          label='Errores Train')  plt.legend(loc='down right')  plt.xlabel('Num. splits')  plt.title('Análisis errores')  plt.style.context('seaborn-whitegrid') |

￼￼

El número de errores es mas alto para Train que para Test, lo que significa que el modelo no se ha entrenado bien con este algoritmo. Ya que un módelo bien entrenado debería tener mas aciertos con la parte de Train que de Test y aqui ocurre todo lo contrario.

El porcentaje de errores es muy alto, entorno a un 18 para Test y entorno a un 25 para Train.

Este algoritmo no es válido para crear un modelo de predicción en este dataset.

**REDES NEURONALES**

Usamos el algoritmo MLPClassifier.

El parámetro que se le pasa es “hidden\_layers”, que representa el número de neuronas en la capa oculta.

Usamos GridSearch para encontrar el mejor algoritmo para mi dataset. Establecemos 3 hidden layers y variamos en numero de neuronas desde 1 a 10.

**CODIGO**

|  |
| --- |
| #Redes neuronales  import numpy as np  from sklearn.neural\_network import MLPClassifier  from matplotlib import pyplot as plt  from sklearn.decomposition import PCA  from sklearn.model\_selection import learning\_curve  #importar datos del CSV  data=pd.read\_csv('C:\\Users\\Esther\\Desktop\\Master\\Modulo 8\\SPAM\\spambasedata.csv', header=None)  #data = pd.read\_csv('spambase.data', header=None)  data.rename(columns={57:'is\_spam'}, inplace=True)  ham = data[data['is\_spam'] == 0]  spam = data[data['is\_spam'] == 1]  nospam = data[data['is\_spam'] == 0]  X = data  print (X.shape)  Y = X.pop('is\_spam')  #Aplicamos Grid Search para comparar diferentes scores  #con 3 hidden layyers y de 1 a 10 neuronas en cada hidden layer  cls = MLPClassifier( max\_iter=100, alpha=1e-4,  solver='sgd', verbose=10, tol=1e-4, random\_state=1,  learning\_rate\_init=0.01)  mi\_param\_grid={'hidden\_layer\_sizes':[(1,1,1),(2,2,2),(3,3,3),(4,4,4),(5,5,5),(6,6,6),(7,7,7),(8,8,8),(9,9,9),(10,10,10)]}  migscv=GridSearchCV(cls,mi\_param\_grid,cv=10,verbose=2)  migscv.fit(Xtrain,Ytrain)  migscv.best\_estimator\_ |

**RESULTADO**

|  |
| --- |
| migscv.best\_estimator\_  Out[137]:  MLPClassifier(activation='relu', alpha=0.0001, batch\_size='auto', beta\_1=0.9,  beta\_2=0.999, early\_stopping=False, epsilon=1e-08,  hidden\_layer\_sizes=(5, 5, 5), learning\_rate='constant',  learning\_rate\_init=0.01, max\_iter=100, momentum=0.9,  nesterovs\_momentum=True, power\_t=0.5, random\_state=1, shuffle=True,  solver='sgd', tol=0.0001, validation\_fraction=0.1, verbose=10,  warm\_start=False) |

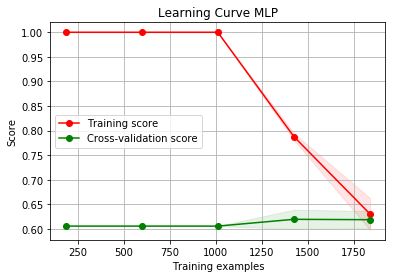
Grafica curva aprendizaje

Ploteamos la curva de aprendizaje para el mejor algoritmo encontrado:

|  |
| --- |
| #Aplicamos MLP con 3 hidden layers y 5 neuronas cada una  def plot\_learning\_curve(estimator, title, X, y, ylim=None, cv=None,  n\_jobs=1, train\_sizes=np.linspace(.1, 1.0, 5)):  plt.figure()  plt.title(title)  if ylim is not None:  plt.ylim(\*ylim)  plt.xlabel("Training examples")  plt.ylabel("Score")  train\_sizes, train\_scores, test\_scores = learning\_curve(  estimator, X, y, cv=cv, n\_jobs=n\_jobs, train\_sizes=train\_sizes)  train\_scores\_mean = np.mean(train\_scores, axis=1)  train\_scores\_std = np.std(train\_scores, axis=1)  test\_scores\_mean = np.mean(test\_scores, axis=1)  test\_scores\_std = np.std(test\_scores, axis=1)  plt.grid()  plt.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,  train\_scores\_mean + train\_scores\_std, alpha=0.1,  color="r")  plt.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,  test\_scores\_mean + test\_scores\_std, alpha=0.1, color="g")  plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r",  label="Training score")  plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g",  label="Cross-validation score")  plt.legend(loc="best")  plt.show()  print "Fitting..."  cls.fit(Xtrain, Ytrain)  plot\_learning\_curve(cls,"Learning Curve MLP", Xtrain, Ytrain)  print("Training set score: %f" % cls.score(Xtrain, Ytrain))  print("Test set score: %f" % cls.score(Xtest, Ytest)) |

**RESULTADOS**

|  |
| --- |
| Training set score: 0.609641  Test set score: 0.610206 |



Visualizacion PCA

GNB

gridsearch

Visualizacion

SVM (Laura)

Redes neuronales(Esther)

DT(Laura)

RDF