UNIDAD 4 - CLASIFICADOR NAIVE BAYES

Docente: Ruth Rosario Chirinos Contreras

Fecha de Entrega: 13/Sept/2023 19:00 PM

Clasificacion de SMS Spam con Naive Bayes

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from collections import Counter
5 from sklearn import feature_extraction, model_selection, naive_bayes, metrics
7 import warnings
8 warnings.filterwarnings("ignore")
9 %matplotlib inline
```

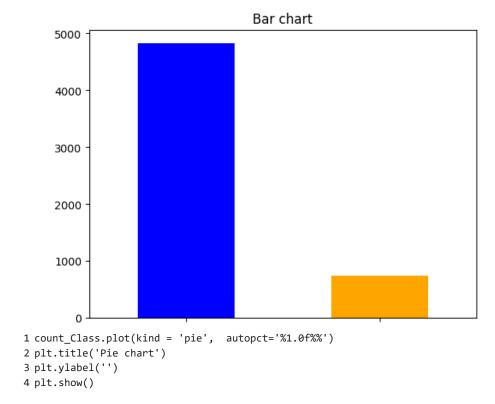
▼ Explorando el Dataset

```
1 data = pd.read_csv(r'spam.csv', encoding='latin-1')
2 data.head(n=10)
3 # 'ham': es e-mail que no es Spam
```

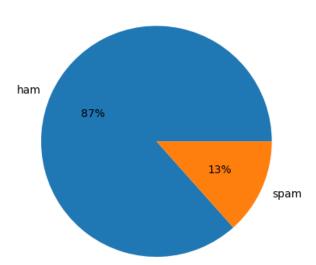
Unnamed: 4	Unnamed: 3	Unnamed: 2	v2	v1	
NaN	NaN	NaN	Go until jurong point, crazy Available only	ham	0
NaN	NaN	NaN	Ok lar Joking wif u oni	ham	1
NaN	NaN	NaN	Free entry in 2 a wkly comp to win FA Cup fina	spam	2
NaN	NaN	NaN	U dun say so early hor U c already then say	ham	3
NaN	NaN	NaN	Nah I don't think he goes to usf, he lives aro	ham	4
NaN	NaN	NaN	FreeMsg Hey there darling it's been 3 week's n	spam	5
NaN	NaN	NaN	Even my brother is not like to speak	ham	6

▼ Distribucion spam/non-spam plots

```
1 count_Class=pd.value_counts(data["v1"], sort= True)
2 count_Class.plot(kind= 'bar', color= ["blue", "orange"])
3 plt.title('Bar chart')
4 plt.show()
```



Pie chart



▼ Analisis de Texto

Queremos encontrar las frecuencias de las palabras en los mensajes spam y no spam. Las palabras de los mensajes serán características del modelo.

Usamos la función Contador.

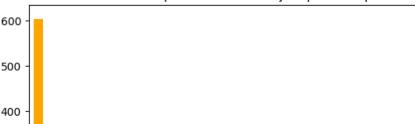
```
1 #data[
          data['v1']=='ham'
                                ]
3 # SMS normales
4 count1 = Counter(" ".join(data[data['v1']=='ham']["v2"]).split()).most_common(20)
5 df1 = pd.DataFrame.from_dict(count1)
```

```
6 df1 = df1.rename(columns={0: "words in non-spam", 1 : "count"})
 8 # SMS con spam
9 count2 = Counter(" ".join(data[data['v1']=='spam']["v2"]).split()).most_common(20)
10 df2 = pd.DataFrame.from dict(count2)
11 df2 = df2.rename(columns={0: "words in spam", 1 : "count_"})
 1 df1.plot.bar(legend = False)
 2 y_pos = np.arange(len(df1["words in non-spam"]))
 3 plt.xticks(y_pos, df1["words in non-spam"])
 4 plt.title('Palabras mas frequentes en mensajes que no son spam')
 5 plt.xlabel('words')
 6 plt.ylabel('number')
 7 plt.show()
```

Palabras mas frequentes en mensajes que no son spam 1600 1400 1200 1000 800 600 400 200 0 and me of for that n .<u>s</u> Ħ О 2 .⊑ words

```
1 df2.plot.bar(legend = False, color = 'orange')
2 y_pos = np.arange(len(df2["words in spam"]))
3 plt.xticks(y_pos, df2["words in spam"])
4 plt.title('Palabras mas frequentes en mensajes que son spam')
5 plt.xlabel('words')
6 plt.ylabel('number')
7 plt.show()
```





Podemos ver que la mayoría de las palabras frecuentes en ambas clases son palabras vacías como 'a', 'a', 'o', etc.

Con palabras vacías nos referimos a las palabras más comunes de un idioma, no existe una lista única y universal de palabras vacías.

Ingenieria de Caracteristicas (Feature engineering)

El preprocesamiento de texto, la tokenización y el filtrado de palabras vacías se incluyen en un componente de alto nivel que es capaz de crear un diccionario de características y transformar documentos en vectores de características.

Eliminamos las palabras vacías para mejorar los análisis

```
1 f = feature_extraction.text.CountVectorizer(stop_words = 'english')
2 X = f.fit_transform(data["v2"])
3 np.shape(X)
    (5572, 8404)
```

Hemos creado más de 8400 funciones nuevas. La nueva característica j en la fila i es igual a 1 si la palabra w_i aparece en el ejemplo de texto i. Si no aparece sera 0.

Analisis Predictivo (Predictive Analysis)

Mi objetivo es predecir si un nuevo SMS es spam o no. Supongo que es mucho peor clasificar erróneamente lo que no es spam que clasificar erróneamente un spam. (No quiero tener falsos positivos)

La razón es que normalmente no reviso los mensajes de spam.

Las dos situaciones posibles son:

1. Nuevos SMS spam en mi bandeja de entrada. (Falso negativo).

RESULTADO: Lo borro.

2. Nuevos SMS que no son spam en mi carpeta de spam (falso positivo).

RESULTADO: Probablemente no lo leo.

¡¡¡Prefiero la primera opción!!!

Primero transformamos la variable spam/no spam en una variable binaria, luego dividimos nuestro conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba.

```
1 data["v1"]=data["v1"].map({'spam':1,'ham':0})
2 X_train, X_test, y_train, y_test = model_selection.train_test_split(X, data['v1'], test_size=0.33, random_state=42)
3 print([np.shape(X_train), np.shape(X_test)])
```

```
[(3733, 8404), (1839, 8404)]
```

Clasificador Multinomial naive bayes

Entrenamos diferentes modelos bayes cambiando el parámetro de regularización α .

Evaluamos la exactitud, recuperación y precisión del modelo con el conjunto de prueba.

```
1 list_alpha = np.arange(1/100000, 20, 0.11)
 2 score_train = np.zeros(len(list_alpha))
 3 score_test = np.zeros(len(list_alpha))
 4 recall_test = np.zeros(len(list_alpha))
 5 precision_test= np.zeros(len(list_alpha))
 6 count = 0
 7 for alpha in list_alpha:
      #Definimos el modelo
 9
      bayes = naive_bayes.MultinomialNB(alpha=alpha)
10
      #Entrenamos el modelo
      bayes.fit(X_train, y_train)
11
    #Obtenemos los scores
     score_train[count] = bayes.score(X_train, y_train)
14
      score_test[count] = bayes.score(X_test, y_test)
15
      recall_test[count] = metrics.recall_score(y_test, bayes.predict(X_test))
      precision_test[count] = metrics.precision_score(y_test, bayes.predict(X_test))
16
17
      count = count + 1
```

¡Veamos los primeros 10 modelos de aprendizaje y sus métricas!

```
1 matrix = np.matrix(np.c_[list_alpha, score_train, score_test, recall_test, precision_test])
2 models = pd.DataFrame(data = matrix, columns =
               ['alpha', 'Train Accuracy', 'Test Accuracy', 'Test Recall', 'Test Precision'])
4 models.head(n=10)
```

	alpha	Train Accuracy	Test Accuracy	Test Recall	Test Precision	===
0	0.00001	0.998661	0.974443	0.920635	0.895753	ılı
1	0.11001	0.997857	0.976074	0.936508	0.893939	
2	0.22001	0.997857	0.977162	0.936508	0.900763	
3	0.33001	0.997589	0.977162	0.936508	0.900763	
4	0.44001	0.997053	0.977162	0.936508	0.900763	
5	0.55001	0.996250	0.976618	0.936508	0.897338	
6	0.66001	0.996518	0.976074	0.932540	0.896947	
7	0.77001	0.996518	0.976074	0.924603	0.903101	
8	0.88001	0.996250	0.976074	0.924603	0.903101	
9	0.99001	0.995982	0.976074	0.920635	0.906250	

Selecciono el modelo con mayor precisión de prueba.

```
1 best_index = models['Test Precision'].idxmax()
2 models.iloc[best_index, :]
                      15.730010
    alpha
    Train Accuracy
                       0.979641
```

Test Accuracy 0.969549 Test Recall 0.777778 Test Precision 1.000000 Name: 143, dtype: float64

Mi mejor modelo no produce ningún falso positivo, que es nuestro objetivo.

¡Veamos si hay más de un modelo con 100% de precisión!

1 models[models['Test Precision']==1].head(n=5)

	alpha	Train Accuracy	Test Accuracy	Test Recall	Test Precision	
143	15.73001	0.979641	0.969549	0.777778	1.0	ılı
144	15.84001	0.979641	0.969549	0.777778	1.0	
145	15.95001	0.979641	0.969549	0.777778	1.0	
146	16.06001	0.979373	0.969549	0.777778	1.0	
147	16.17001	0.979373	0.969549	0.777778	1.0	

Entre estos modelos con mayor precisión posible vamos a seleccionar cuál tiene mayor precisión de prueba.

```
1 best_index = models[models['Test Precision']==1]['Test Accuracy'].idxmax()
2 bayes = naive_bayes.MultinomialNB(alpha=list_alpha[best_index])
3 bayes.fit(X_train, y_train)
4 models.iloc[best_index, :]
    alpha
                     15.730010
   Train Accuracy 0.969545
Test Accuracy 0.777778
0.77778
    Test Precision 1.000000
    Name: 143, dtype: float64
```

▼ Matriz de Confusion (Confusion matrix) con el clasificador naive bayes

```
1 m_confusion_test = metrics.confusion_matrix(y_test, bayes.predict(X_test))
2 pd.DataFrame(data = m_confusion_test, columns = ['Predicted 0', 'Predicted 1'],
              index = ['Actual 0', 'Actual 1'])
3
              Predicted 0 Predicted 1
    Actual 0
                     1587
                                     0
                                          11.
    Actual 1
                       56
                                   196
```

Hemos clasificado erróneamente 56 correos electrónicos no deseados como spam y 0 spam como spam (que es lo que queríamos)

▼ Ejercicios

Entrenamos diferentes modelos bayes cambiando el parámetro de regularización α .

Evaluamos la exactitud, recuperación y precisión del modelo con el conjunto de prueba.

```
1 #valor= <----modifique el valor, por ejemplo: 100000---->
 2 \text{ valor} = 1
 3 list_alpha = np.arange(1/valor, 20, 0.11)
 4 score_train = np.zeros(len(list_alpha))
 5 score test = np.zeros(len(list alpha))
 6 recall_test = np.zeros(len(list_alpha))
 7 precision_test= np.zeros(len(list_alpha))
 8 \text{ count} = 0
9 for alpha in list_alpha:
10 #Definimos el modelo
bayes = naive_bayes.MultinomialNB(alpha=alpha)
12
    #Entrenamos el modelo
13
    bayes.fit(X_train, y_train)
14
     #Obtenemos los scores
15
     score_train[count] = bayes.score(X_train, y_train)
16
     score_test[count] = bayes.score(X_test, y_test)
     recall_test[count] = metrics.recall_score(y_test, bayes.predict(X_test))
17
18
      precision_test[count] = metrics.precision_score(y_test, bayes.predict(X_test))
19
      count = count + 1
```

¡Veamos los primeros 10 modelos de aprendizaje y sus métricas, y los ultimos 10 modelos de aprendizaje y sus métricas!

```
1 # Aqui tu codigo
2 matrix = np.matrix(np.c_[list_alpha, score_train, score_test, recall_test, precision_test])
3 models = pd.DataFrame(data = matrix, columns =
              ['alpha', 'Train Accuracy', 'Test Accuracy', 'Test Recall', 'Test Precision'])
5 models.head(n=10)
```

	alpha	Train Accuracy	Test Accuracy	Test Recall	Test Precision	
0	1.00	0.995982	0.976074	0.920635	0.906250	ıl.
1	1.11	0.995446	0.977705	0.920635	0.916996	
2	1.22	0.995446	0.978249	0.920635	0.920635	
3	1.33	0.995178	0.978793	0.920635	0.924303	
4	1.44	0.995178	0.980968	0.920635	0.939271	
5	1.55	0.994910	0.980968	0.920635	0.939271	
6	1.66	0.994910	0.981512	0.920635	0.943089	
7	1.77	0.994374	0.981512	0.920635	0.943089	
8	1.88	0.994107	0.981512	0.920635	0.943089	
9	1.99	0.994107	0.981512	0.916667	0.946721	

1 models.tail(n=10)

	alpha	Train Accuracy	Test Accuracy	Test Recall	Test Precision	
163	18.93	0.977498	0.966830	0.757937	1.0	ılı
16/	10 0/	Λ Q77/QQ	U 08883U	n 757037	1 0	

▼ Evaluando el modelo con mayor presicion

- 1. Selecciona el modelo con mayor precisión de prueba
- 2. Muestra sus metricas: Train Accuracy, Test Accuracy, Test Recall, Test Precision
- 3. Elabora su matriz de confusion

```
0.975891
    169 19.59
                                     0.966286
                                                  0.753968
                                                                       1 0
1 # Aqui tu codigo
2 best_index = models['Test Precision'].idxmax()
3 models.iloc[best_index, :]
                     15.740000
   alpha
   Train Accuracy 0.979641
   Test Accuracy
                      0.969549
   Test Recall
                      0.777778
                     1.000000
   Test Precision
   Name: 134, dtype: float64
1 m_confusion_test = metrics.confusion_matrix(y_test, bayes.predict(X_test))
2 pd.DataFrame(data = m_confusion_test, columns = ['Predicted 0', 'Predicted 1'],
             index = ['Actual 0', 'Actual 1'])
             Predicted 0 Predicted 1
    Actual 0
                    1587
                                         Actual 1
                      62
                                  190
```

Evaluando el modelo con menor presicion

- 1. Selecciona el modelo con menor precisión de prueba
- 2. Muestra sus metricas: Train Accuracy, Test Accuracy, Test Recall, Test Precision
- 3. Elabora su matriz de confusion

```
1 # Aqui tu codigo
2 worst_index = models['Test Precision'].idxmin()
3 models.iloc[worst_index, :]
    alpha
                     1.000000
    Train Accuracy 0.995982
    Test Accuracy
                     0.976074
   Test Recall
                     0.920635
   Test Precision
                     0.906250
    Name: 0, dtype: float64
1 m_confusion_test = metrics.confusion_matrix(y_test, bayes.predict(X_test))
2 pd.DataFrame(data = m_confusion_test, columns = ['Predicted 0', 'Predicted 1'],
             index = ['Actual 0', 'Actual 1'])
             Predicted 0 Predicted 1
    Actual 0
                    1587
                                    0
                                         ıl.
    Actual 1
                      62
                                  190
```

✓ 0 s se ejecutó 12:09

• ×