# Machine Learning III: Introducción a Métodos de Clasificación Supervisada

CONCEPTOS

## PARTE 1: Un enfoque práctico para Machine Learning

1. Acerca del software.
2. ¿Qué es Machine Learning?
3. Modelando el problema de Machine Learning
4. El problema de clasificación supervisada. Un ejemplo programático básico guiado

   4.1 Representación del problema en sklearn

   4.2 Aprendiendo y prediciendo

   4.3 Más sobre el 'feature space'

   4.4 Entrenamiento y prueba

   4.5 Selección de modelos (I)

## PARTE 2: Conceptos de aprendizaje y teoría

1. ¿Qué es aprender?
   - Aprendizaje PAC
2. Dentro del modelo de aprendizaje
   - El algoritmo de aprendizaje Machine Learning humano
   - Clase de modelo y espacio de hipótesis
   - Función objetivo
   - Algoritmo de búsqueda/optimización/aprendizaje
3. Curvas de aprendizaje y sobreajuste (Overfitting)
   - Curvas de aprendizaje
   - Sobreajuste
4. Curas para el sobreajuste
   - Selección de modelos (II)
   - Regularización
   - Conjunto
5. ¿Qué hacer cuando...?

## PARTE 3: Primeros modelos

1. Modelos generativos y discriminativos
   - Modelos bayesianos (Naive Bayes) y algunas aplicaciones.
2. Máquinas de Vectores de Soporte (Support Vector Machines).
   - Modelando Máquinas de Vectores de Soporte.

# PART 3: Primeros modelos

# 1. Modelos generativos y discriminativos.

En el campo del aprendizaje automático, a menudo encontramos dos enfoques fundamentales para entender y modelar datos: **Modelos Generativos** y **Modelos Discriminativos**. Estos modelos ofrecen perspectivas distintas sobre cómo abordar problemas en el aprendizaje automático, cada uno con sus propias ventajas y aplicaciones. Esta sección tiene como objetivo aclarar estos conceptos, proporcionando una comprensión más clara para los estudiantes, particularmente aquellos para quienes el inglés no es su primer idioma.

- **Modelos Generativos** están diseñados para capturar cómo se generan los datos, permitiéndonos simular y entender la distribución subyacente de los datos. El objetivo principal de los modelos generativos es estimar la distribución de probabilidad conjunta $P(x, y)$, donde $x$ representa las características de entrada y $y$ representa las etiquetas de salida. Al dominar esta distribución, los modelos generativos nos permiten realizar tareas como la clasificación calculando la probabilidad posterior $P(y \vee x)$, que indica la probabilidad de una etiqueta dada la entrada de datos. Este cálculo se facilita mediante la Regla de Bayes, que vincula la probabilidad posterior con la probabilidad de verosimilitud $P(x \vee y)$ y la probabilidad previa $P(y)$. Una aplicación clásica de los modelos generativos es en escenarios donde no solo se requiere clasificación, sino también la generación de nuevos puntos de datos similares. Por ejemplo, los clasificadores Naive Bayes, los Modelos Ocultos de Markov y los Modelos de Mezcla Gaussiana son todos modelos generativos que pueden predecir etiquetas y generar datos que imitan el conjunto de datos original.

- **Modelos Discriminativos**, por otro lado, se centran directamente en aprender el límite de decisión o función que distingue entre diferentes clases en el conjunto de datos. A diferencia de los modelos generativos, los modelos discriminativos no se preocupan por modelar la distribución de los datos. Su objetivo es predecir la etiqueta de salida $y$ para una entrada dada $x$ de la manera más precisa posible, aproximando la probabilidad condicional $P(y \vee x)$. Esto hace que los modelos discriminativos sean particularmente adecuados para tareas donde el objetivo principal es la predicción, como la clasificación y la regresión. Ejemplos de modelos discriminativos incluyen la Regresión Logística, las Máquinas de Vectores de Soporte (SVM) y las Redes Neuronales. Estos modelos son celebrados por su

capacidad para producir predicciones precisas, especialmente en escenarios complejos donde entender el proceso de generación de los datos es menos crítico.

- Un subconjunto importante de ambos modelos generativos y discriminativos son **Modelos Lineales**. Estos modelos asumen que los límites de decisión pueden expresarse como funciones lineales (o a veces cuadráticas) de las características de entrada. Un modelo lineal puede ser descrito por la ecuación $h(x)=a^T x+b$, donde $a$ representa un vector de pesos, $x$ es el vector de características de entrada, y $b$ es un término de sesgo. A pesar de su simplicidad, los modelos lineales forman la piedra angular de muchos algoritmos de aprendizaje automático porque son fáciles de interpretar y pueden ser la base para modelos más complejos. A través de la exploración de ejemplos lineales simples de ambos modelos generativos y discriminativos, los estudiantes pueden obtener una visión de cómo estos modelos se aplican a problemas de clasificación del mundo real, ofreciendo una comprensión fundamental de los límites de decisión lineales.

Esta visión general completa tiene como objetivo aclarar las distinciones y aplicaciones de modelos generativos y discriminativos en el aprendizaje automático, con un enfoque especial en hacer accesibles los conceptos para los estudiantes para quienes el inglés es un segundo idioma. Al proporcionar ejemplos y explicar los principios fundamentales detrás de estos modelos, el objetivo es fomentar una comprensión más profunda de los enfoques y su significado en el campo del aprendizaje automático.

VIDEO: Explicación corta

# 1.1. Modelos bayesianos (Naive Bayes) y algunas aplicaciones

Imagina que tenemos una gran cesta de frutas y queremos clasificarlas en tipos como manzanas, plátanos y naranjas. Esta tarea es similar a lo que hace el clasificador Naive Bayes en el mundo del aprendizaje automático, pero en lugar de clasificar frutas, clasifica información.

## ¿Qué es Naive Bayes?

Naive Bayes es como una máquina de clasificación inteligente que nos ayuda a organizar cosas basándose en lo que sabe sobre ellas. Para nuestro ejemplo de frutas, piensa en Naive Bayes como un amigo que te ayuda a clasificar frutas diciéndote cuál es probable que sea una manzana, un plátano o una naranja, basándose en sus características como color, forma y sabor.

- **Teorema de Bayes**: Esta es la regla que usa nuestra máquina de clasificación. Es una fórmula matemática que ayuda a predecir el tipo de fruta basándose en sus características. Por ejemplo, si le dices que una fruta es larga y amarilla, el Teorema de Bayes ayuda a adivinar que probablemente sea un plátano.

- **Suposición Ingenua**: La parte "ingenua" proviene de que la máquina trata cada característica (como el color o la forma) como si fuera completamente independiente de las demás. Aunque esto no es cómo funcionan las cosas en la vida real (por ejemplo, las naranjas son tanto naranjas como redondas), esta suposición

hace que nuestra máquina sea muy rápida y sorprendentemente buena para adivinar.

## ¿Cómo clasifica Naive Bayes las frutas?

Desglosémoslo en pasos simples:

1. **Aprendizaje**: Primero, le muestras a la máquina muchas frutas diferentes y le dices qué son. Esto es como enseñarle el juego mostrándole ejemplos.
2. **Adivinanza**: Después de aprender, cuando le das a la máquina una nueva fruta que no ha visto antes, utiliza lo que aprendió para adivinar el tipo de fruta. Mira las características de la fruta (color, forma, sabor) y calcula a qué tipo es más probable que pertenezca.

### Viendo Naive Bayes en acción

Imagina que dibujas círculos alrededor de manzanas, plátanos y naranjas basándote en sus características. Naive Bayes hace algo similar en su mente. Cuando llega una nueva fruta, ve dónde encaja mejor basándose en los círculos dibujados a partir de lo que aprendió.

## ¿Por qué es tan especial Naive Bayes?

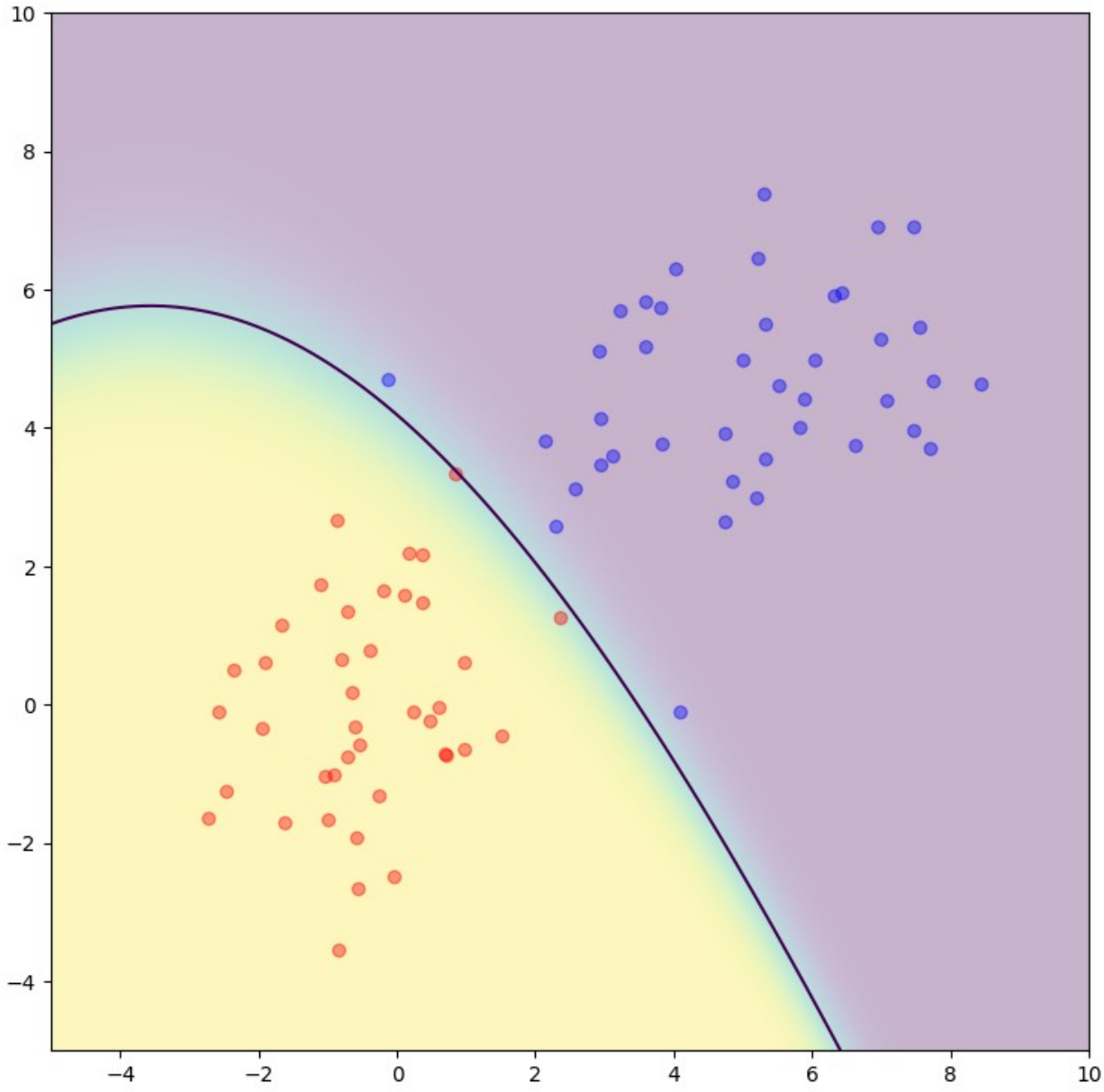
A pesar de que hace suposiciones simples, Naive Bayes puede clasificar rápidamente mucha información (o frutas) y hacer conjeturas precisas. Es como tener un clasificador de frutas super rápido que mejora cuanto más aprende.

A continuación, veremos cómo podemos usar este increíble clasificador no solo para frutas, sino para clasificar todo tipo de cosas, haciendo nuestras vidas más fáciles y organizadas. Lo tomaremos paso a paso, asegurándonos de que sea divertido y fácil de entender.

VIDEO: Naive Bayes explicado

```
# Restablecer el entorno para asegurar un estado limpio
%reset -f

# Habilitar la generación de gráficos en el Jupyter Notebook
%matplotlib inline

# Importar las bibliotecas necesarias
import numpy as np
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt

# Crear datos sintéticos para dos grupos
# Los datos para cada grupo se generan alrededor de un punto central
diferente para simular distinción
X = np.concatenate([1.25*np.random.randn(40,2),
5+1.5*np.random.randn(40,2)])
y = np.concatenate([np.ones((40,1)), -np.ones((40,1))])
```

```
# Inicializar el clasificador Naive Bayes Gaussiano
nb = GaussianNB()
# Entrenar el clasificador con los datos sintéticos
nb.fit(X, y.ravel())

# Crear una cuadrícula de puntos para visualizar el límite de decisión
delta = 0.025
xx = np.arange(-5.0, 10.0, delta)
yy = np.arange(-5.0, 10.0, delta)
XX, YY = np.meshgrid(xx, yy)

# Predecir probabilidades en la cuadrícula para visualizar el límite
de decisión
Z = nb.predict_proba(np.c_[XX.ravel(), YY.ravel()])
Z = Z[:, 1].reshape(XX.shape)

# Graficar
plt.figure()
# Separar los datos en dos grupos para graficar
idxplus = y == 1
idxminus = y == -1
idxplus = idxplus.flatten()
idxminus = idxminus.flatten()
# Graficar el primer grupo en rojo
plt.scatter(X[idxplus, 0], X[idxplus, 1], color='r', alpha=0.4)
# Graficar el segundo grupo en azul
plt.scatter(X[idxminus, 0], X[idxminus, 1], color='b', alpha=0.4)
# Superponer la densidad de probabilidad para visualizar el límite de
decisión
plt.imshow(Z, interpolation='bilinear', origin='lower', extent=(-5,
10, -5, 10), alpha=0.3, vmin=0, vmax=1)
# Dibujar la línea de contorno donde la probabilidad es 0.5 (límite de
decisión)
plt.contour(XX, YY, Z, [0.5])
# Ajustar el tamaño de la figura para mejor visibilidad
fig = plt.gcf()
fig.set_size_inches(9, 9)
```




Si el límite no es lineal, ¿por qué se considera un modelo lineal? No es un modelo lineal, aunque es un modelo afín con respecto a los pesos. En el caso particular de la clasificación de texto, utilizaremos ciertas funciones de densidad de probabilidad que harán que el modelo sea lineal.

El clasificador Gaussian Naive Bayes, un pilar del aprendizaje automático, destaca particularmente por su simplicidad y efectividad en tareas de clasificación. A través del ejemplo en Python proporcionado, hemos visualizado no solo el proceso de clasificación, sino también cómo Naive Bayes toma decisiones basadas en probabilidades.

**Aspectos clave**

**Simplicidad en acción:** El clasificador Naive Bayes, a pesar de su suposición subyacente de independencia de características, demuestra un rendimiento robusto al clasificar datos en

grupos distintos. Este ejemplo, utilizando datos sintéticos, muestra cómo incluso con suposiciones básicas, se pueden obtener percepciones significativas.

**Visualización de límites de decisión:** Al graficar los puntos de datos y el límite de decisión, obtenemos una comprensión visual de cómo Naive Bayes clasifica los datos. La línea de contorno en la que la probabilidad es 0.5 actúa como umbral: los puntos de datos que caen a un lado de esta línea pertenecen a una categoría, y los que están al otro lado pertenecen a otra.

**Enfoque probabilístico:** El gradiente de fondo en la visualización representa la probabilidad de pertenecer a una de las categorías. Este gradiente ilustra la naturaleza probabilística de Naive Bayes, ofreciendo más que solo clasificaciones; proporciona percepciones sobre la certeza de sus predicciones.

# 1.2. Representación básica de documentos

Imagina que estás ordenando una pila de diferentes historias y quieres organizarlas en grupos como "cuentos de aventuras" o "cuentos de hadas". Esto es lo que llamamos **clasificación de texto**, y utilizamos un método llamado Naive Bayes para ayudarnos a averiguar a qué grupo pertenece cada historia.

## ¿Cómo entendemos las historias?

Para ayudar a Naive Bayes a clasificar historias, primero necesitamos traducir las historias a un formato que pueda entender. Hacemos esto utilizando un enfoque de **bolsa de palabras**. Piensa en esto como tomar cada historia y descomponerla en una lista de palabras que nos interesan.

- **Ejemplo de Bolsa de Palabras**: Supongamos que estamos interesados en historias sobre "piratas" y "magos". Decidimos enfocarnos en ciertas palabras como "barco", "tesoro", "varita" y "hechizo". Luego vemos cuántas veces aparecen estas palabras en cada historia.

| Historia | Barco | Tesoro | Varita | Hechizo |
|---|---|---|---|---|
| 1 (Piratas) | 3 | 2 | 0 | 0 |
| 2 (Piratas) | 2 | 1 | 0 | 1 |
| 3 (Magos) | 0 | 0 | 2 | 3 |
| 4 (Magos) | 0 | 1 | 3 | 2 |

- En esta tabla, la Historia 1 tiene 3 palabras "barco", 2 palabras "tesoro" y ninguna de las palabras "varita" o "hechizo", lo que ayuda a Naive Bayes a adivinar que es una historia de piratas.

## Cómo Naive Bayes hace su conjetura

Naive Bayes observa el recuento de palabras para cada historia. Si una historia tiene más palabras de "pirata" como "barco" y "tesoro", adivina que es sobre piratas. Si tiene más palabras de "mago" como "varita" y "hechizo", adivina que es sobre magos.

## Puntos clave para recordar

- **Conteo vs. Presencia**: A veces, solo saber si una palabra aparece en la historia o no (sí/no) es suficiente, en lugar de contar cuántas veces aparece.

- **El orden de las palabras no importa**: Con la bolsa de palabras, el orden de las palabras se ignora. "El tesoro está en el barco" y "El barco está en el tesoro" se verían como lo mismo. Aunque significan cosas diferentes, para nuestro propósito de clasificar historias, este enfoque simple aún puede ser muy efectivo.

Al transformar historias en bolsas de palabras, Naive Bayes nos ayuda a categorizarlas en temas como "piratas" o "magos". A pesar de su simplicidad, este método es una herramienta poderosa para ayudar a las computadoras a entender y organizar historias, al igual que nosotros.

## Entendiendo el Clasificador Naive Bayes en la Clasificación de Documentos

Naive Bayes es un enfoque sencillo pero poderoso utilizado para predecir la categoría de documentos. Es como jugar un juego de adivinanzas donde, basado en las palabras utilizadas en un documento, intentamos predecir su tema, como si se tratara de "economía" o "tecnología".

## Cómo Funciona Naive Bayes

En esencia, Naive Bayes selecciona la categoría que es más probable que sea correcta basada en las palabras presentes en el documento. Esto se hace utilizando una fórmula matemática conocida como Teorema de Bayes:

- Para descubrir cuál categoría es la más probable, usamos esta fórmula:

$$\hat{y} = arg\,max_{y}\, p(y \vee x).$$

  Aquí, $\hat{y}$ es nuestra mejor conjetura para la categoría del documento, $y$ representa una categoría posible, y $x$ es la descripción del documento (como las palabras que contiene).

- El Teorema de Bayes nos ayuda en este juego de adivinanzas al relacionar diferentes probabilidades:

$$p(y \vee x) = \frac{p(x \vee y)\,p(y)}{p(x)}.$$

  En términos simples, esto nos dice cuán probable es una categoría dada la descripción del documento. Calculamos esto mirando cuán comunes son las palabras en cada categoría ($p(x \vee y)$), cuán común es cada categoría ($p(y)$) y cuán comunes son estas palabras en todos los documentos ($p(x)$).

## Simplificando el Proceso

Cuando estamos clasificando documentos, esencialmente estamos comparando qué categoría es más probable basada en las palabras utilizadas. Por ejemplo, si un documento contiene palabras más comunes en "economía" que en "tecnología", será clasificado bajo "economía".

- Curiosamente, no necesitamos preocuparnos por $p(x)$, la probabilidad de ver un conjunto particular de palabras, porque no cambia nuestra decisión. Esto simplifica nuestra fórmula a:

$$P(y \vee x) \propto P(y)P(x \vee y)$$

Esto significa que estamos principalmente interesados en cuán probable es una categoría ($P(y)$) y cuán probable es ver estas palabras si el documento está en esa categoría ($P(x \vee y)$).

## La Suposición Especial de Naive Bayes

Lo que hace "naive" a Naive Bayes es su suposición de que cada palabra en el documento afecta la categoría independientemente de otras palabras. Esta suposición nos permite simplemente multiplicar las probabilidades de palabras individuales relacionadas con una categoría para encontrar la probabilidad total:

$$p(x_1, x_2, \ldots, x_N \vee y) = p(x_1 \vee y)p(x_2 \vee y)\ldots p(x_N \vee y) = \prod_{i=1}^{N} p(x_i \vee y)$$

Por ejemplo, la probabilidad de que un documento trate sobre "tecnología" dado que contiene ciertas palabras se puede calcular multiplicando las probabilidades de cada una de esas palabras pertenecientes a la categoría "tecnología".

## Aplicación Práctica

Al aplicar Naive Bayes a la clasificación de documentos, es posible que no siempre conozcamos la probabilidad previa de cada categoría ($p(y)$). En tales casos, podríamos tratar todas las categorías como igualmente probables o usar lo que se conoce como un prior no informativo. Esto nos lleva a centrarnos en la probabilidad de palabras dentro de las categorías, simplificando nuestra tarea a encontrar la categoría más probable basada en las palabras del documento.

En resumen, Naive Bayes nos ayuda a clasificar documentos comparando las probabilidades de palabras dentro de las categorías, haciéndolo una herramienta sencilla pero efectiva para entender y organizar la información.

# 1.3. Estimación de las probabilidades condicionadas

El último paso que queda es la estimación de las probabilidades condicionadas individuales. Existen dos variantes clásicas el **Naive Bayes multinomial** y el **Naive Bayes Bernoulli**. La diferencia entre ambas radica en el objetivo de lo que modelan. **En el NB Multinomial calculamos la probabilidad de generar el documento observado.** En este sentido, multiplicamos la probabilidad condicional de cada palabra del documento por todas las palabras presentes en el documento. Una visión alternativa es el *modelo Bernoulli*. **Obsérvese que en el Bernouilli Naive Bayes la probabilidad final depende de las palabras que aparecen en el documento pero también de las palabras que no aparecen** mientras que en el multinomial NB sólo depende de las palabras que aparecen. Por el contrario, el Naive Bayes multinomial tiene en cuenta la multiplicidad de las palabras del documento, mientras que Bernoulli no. Consideremos en este ejemplo el *modelo Bernoulli* que es coherente con nuestra representación donde un cero

indica que una palabra no está presente en el documento y un uno representa que está presente. Para estimar esta probabilidad podemos utilizar una aproximación frecuentista a la probabilidad, es decir, estimaremos la probabilidad como la frecuencia de aparición de cada término en cada categoría. Este cálculo divide el número de documentos en los que aparece la palabra entre el número total de documentos.

En nuestro ejemplo anterior, $p(x_3=1(\text{aparece la palabra 'precio'})\vee y=\text{'tech'})=1/2$ y $p(x_3=1(\text{aparece la palabra 'precio'})\vee y=\text{'eco'})=2/2$. Se calcula dividiendo el número de documentos en los que aparece la palabra precio en una categoría determinada entre el número de documentos de esa categoría.

### 1.3.1. El efecto de probabilidad cero

En el ejemplo anterior, la probabilidad $p(x_5=1\vee y=texto'eco')=0$. Esto implica que si aparece la palabra "móvil" el documento no puede pertenecer a la clase 'economía'. No es razonable penalizar completamente toda una clase por la aparición o no de una sola palabra. En su lugar, es habitual asignar a esos casos un valor de probabilidad muy bajo. Un método bien conocido para corregir este efecto es la llamada **corrección de Laplace**. Se calcula del siguiente modo

$$p(x_i=1\vee y=c_k)=\frac{\#\text{de documentos de la clase}\,c_k\,\text{donde aparece la palabra}\,x_i\,\text{aparece}+1}{\#\text{de documentos de la clase}\,c_k+M}$$

donde $M$ es la cantidad de palabras de la descripción.

### 1.3.2. Efecto de desbordamiento

A medida que aumenta el número de palabras de la descripción, aumenta la probabilidad de que muchas de esas palabras no estén presentes en el documento. El producto de muchos valores muy pequeños puede provocar efectos de desbordamiento en coma flotante. Por este motivo, es habitual utilizar la probabilidad logarítmica en su lugar. Esta transformación no modifica el límite de decisión. En nuestro caso simplificado

$$\log p(x\vee y)=\sum_{i=1}^{N}\log p(x_i\vee y)$$

## 1.4. Aplicación de Naive Bayes a la clasificación de textos

En este ejemplo, nuestro objetivo es clasificar automáticamente las noticias según su título en veintiocho temas estándar. En este problema trataremos todas las noticias de portada del New York Times desde 1996 hasta 2006, codificadas según las Agendas Políticas (http://www.policyagendas.org). Esta colección de datos ha sido recopilada por Amber E. Boydstun.

En concreto, nos interesa clasificar las noticias de The New York Times en los siguientes macrotemas según su título:

1 Macroeconomics 2 Civil Rights, Minority Issues, and Civil Liberties 3 Health 4 Agriculture 5 Labor, Employment, and Immigration 6 Education 7 Environment 8 Energy 10

Transportation 12 Law, Crime, and Family Issues 13 Social Welfare 14 Community Development and Housing Issues 15 Banking, Finance, and Domestic Commerce 16 Defense 17 Space, Science, Technology and Communications 18 Foreign Trade 19 International Affairs and Foreign Aid 20 Government Operations 21 Public Lands and Water Management 24 State and Local Government Administration 26 Weather and Natural Disasters 27 Fires 28 Arts and Entertainment 29 Sports and Recreation 30 Death Notices 31 Churches and Religion 99 Other, Miscellaneous, and Human Interest

```
# Restablecer el entorno para asegurar un estado limpio
%reset -f

# Cargar datos
import pandas as pd
data = pd.read_csv('Boydstun_NYT_FrontPage_Dataset_1996-2006_0.csv')
data.head()
```

```
   Article_ID      Date Article_Sequence  \
0           1  1/1/1996                a
1           2  1/1/1996                b
2           3  1/1/1996                c
3           4  1/1/1996                d
4           5  1/1/1996                e

                                               Title  \
0  Nation's Smaller Jails Struggle To Cope With S...
1            Dancing (and Kissing) In the New Year
2   Forbes's Silver Bullet for the Nation's Malaise
3  Up at Last, Bridge to Bosnia Is Swaying Gatewa...
4  2 SIDES IN SENATE DISAGREE ON PLAN TO END FURL...

                                             Summary  Topic_6digit  \
0         Jails overwhelmed with hardened criminals        120500
1                            new years activities        280000
2                Steve Forbes running for President        201201
3  U.S. military constructs bridge to help their ...        160200
4  Democrats and Republicans can't agree on plan ...        201206

   Topic_4digit  Topic_2digit  War on Terror  Katrina
Israel/Palestine  \
0          1205            12              0        0
0
1          2800            28              0        0
0
2          2012            20              0        0
0
3          1602            16              0        0
0
4          2012            20              0        0
0
```

```
   Immigration  Presidential Elections  Clinton Impeachment  Enron
Darfur  \
0            0                       0                    0      0
0
1            0                       0                    0      0
0
2            0                       1                    0      0
0
3            0                       0                    0      0
0
4            0                       0                    0      0
0

   Race/Ethnicity  Schiavo
0               0        0
1               0        0
2               0        0
3               0        0
4               0        0
```

Dividamos el conjunto de datos en dos conjuntos:

- Entrenaremos el clasificador con noticias hasta 2004.
- Probaremos el clasificador con noticias de 2005 y 2006.

```
import numpy as np
import pandas as pd

# Dividir el conjunto de datos basado en la fecha: los datos de
entrenamiento serán de antes del 1/1/2004, y los datos de prueba de
2004 a 2006
split = pd.to_datetime(pd.Series(data['Date'])) <
pd.to_datetime('2004-01-01')

# Extraer la columna 'Title' como nuestros datos crudos
raw_data = data['Title']

# Dividir los títulos en conjuntos de entrenamiento y prueba basados
en la fecha
raw_train = raw_data[split]
raw_test = raw_data[np.logical_not(split)]

# Extraer las etiquetas de tema para nuestro conjunto de datos
y = data['Topic_2digit']

# Dividir las etiquetas en conjuntos de entrenamiento y prueba
correspondientes a nuestras divisiones de títulos
y_train = y[split]
y_test = y[np.logical_not(split)]
```

```python
# Imprimir los tamaños de nuestros conjuntos de entrenamiento y prueba para asegurar que la división se realizó correctamente
print('Verificar los tamaños de las divisiones, entrenamiento, prueba y la cantidad total de datos:')
print(raw_train.shape, raw_test.shape, raw_data.shape)

# Mostrar las etiquetas únicas en nuestro conjunto de datos para entender las categorías de clasificación
print('Mostrar las etiquetas:')
print(np.unique(y))
```

```
Verificar los tamaños de las divisiones, entrenamiento, prueba y la cantidad total de datos:
(23584,) (7450,) (31034,)
Mostrar las etiquetas:
[ 1  2  3  4  5  6  7  8 10 12 13 14 15 16 17 18 19 20 21 24 26 27 28 29
 30 31 99]
```

```python
# Importar la herramienta necesaria para el procesamiento de texto
from sklearn.feature_extraction.text import CountVectorizer

# Inicializar el CountVectorizer con parámetros específicos
vectorizer = CountVectorizer(
    min_df=2,  # Una palabra debe aparecer en al menos dos documentos para ser considerada
    stop_words='english',  # Eliminar palabras comunes en inglés (por ejemplo, 'and', 'the', 'of') que no aportan mucho al significado
    strip_accents='unicode'  # Eliminar acentos de los caracteres para mayor consistencia
)

# Demostrar el proceso de preprocesamiento y tokenización con un ejemplo
test_string = raw_train[0]  # Tomar el primer título del conjunto de entrenamiento como ejemplo
print("Ejemplo: " + test_string + "\n")
# Mostrar el resultado del preprocesamiento (por ejemplo, pasar a minúsculas, eliminar puntuación)
print("Preprocesado: " + vectorizer.build_preprocessor()(test_string) + "\n")
# Mostrar la lista de palabras (tokens) después de dividir el texto preprocesado
print("Tokenizado: " + str(vectorizer.build_tokenizer()(test_string)) + "\n")
# Aplicar el analizador completo (preprocesamiento, tokenización y filtrado de palabras vacías)
print("Cadena de datos analizada: " + str(vectorizer.build_analyzer()(test_string)) + "\n")
```

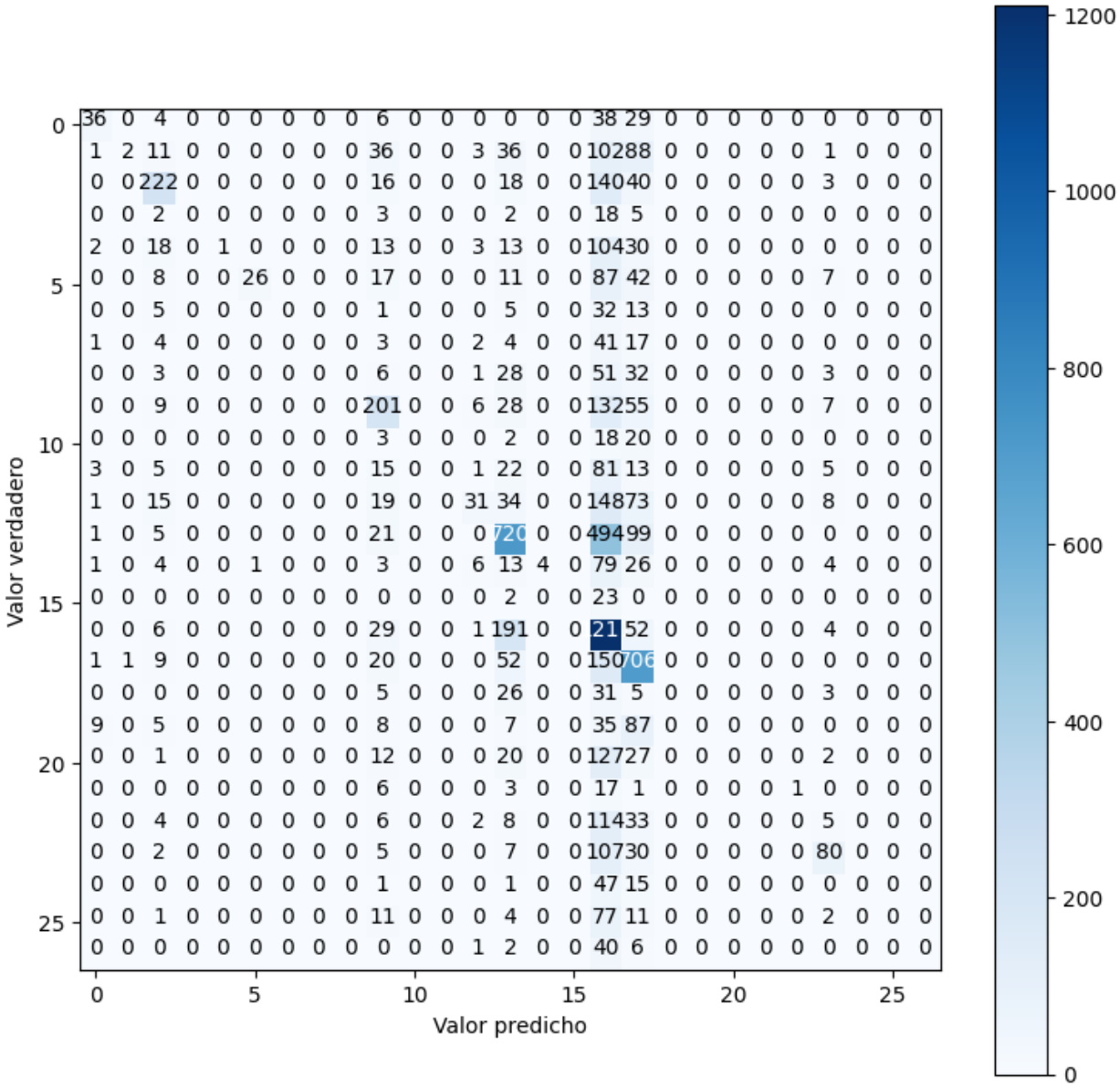
```
# Procesar los conjuntos de datos completos para convertir el texto crudo en una matriz de conteo de tokens
X_train = vectorizer.fit_transform(raw_train)  # Aprender el vocabulario y transformar el conjunto de entrenamiento
X_test = vectorizer.transform(raw_test)  # Transformar el conjunto de prueba basado en el vocabulario aprendido

# Imprimir el número total de tokens (palabras únicas) encontrados en el conjunto de datos
print("Número de tokens: " +
str(len(vectorizer.get_feature_names_out())) + "\n")
# Mostrar una porción de los tokens para inspección
print("Extracto de tokens:")
print(vectorizer.get_feature_names_out()[1000:1100])
```

```
Ejemplo: Nation's Smaller Jails Struggle To Cope With Surge in Inmates


Preprocesado: nation's smaller jails struggle to cope with surge in inmates

Tokenizado: ['Nation', 'Smaller', 'Jails', 'Struggle', 'To', 'Cope', 'With', 'Surge', 'in', 'Inmates']

Cadena de datos analizada: ['nation', 'smaller', 'jails', 'struggle', 'cope', 'surge', 'inmates']

Número de tokens: 8950

Extracto de tokens:
['boeing' 'boiling' 'boils' 'bold' 'bolster' 'bolsters' 'bolt' 'bolts'
 'bomb' 'bombay' 'bombed' 'bomber' 'bombers' 'bombing' 'bombings'
'bombs'
 'bonanza' 'bond' 'bondage' 'bonds' 'bone' 'bones' 'bonn' 'bono'
'bonus'
 'bonuses' 'book' 'books' 'booksellers' 'bookstore' 'boom' 'boomers'
 'booming' 'booms' 'boost' 'boot' 'bora' 'border' 'borders' 'born'
 'borough' 'boroughs' 'borrow' 'borrowing' 'bosnia' 'bosnian'
'bosnians'
 'boss' 'bosses' 'boston' 'botched' 'bottle' 'bought' 'bounce' 'bound'
 'bounty' 'bout' 'bow' 'bowing' 'bowl' 'bows' 'box' 'boxes' 'boxing'
'boy'
 'boycott' 'boys' 'brace' 'braced' 'braces' 'bracket' 'bradley'
'brain'
 'brains' 'branches' 'brand' 'brash' 'brave' 'bravery' 'braves'
'brawl'
 'brawley' 'brazil' 'brazilian' 'breach' 'breaches' 'bread' 'break'
 'breakdown' 'breaking' 'breaks' 'breakthrough' 'breakup' 'breast'
 'breather' 'breed' 'brewers' 'brewing' 'bribery' 'bricks']
```

```
# Habilitar la generación de gráficos directamente dentro del cuaderno
%matplotlib inline

# Importar el clasificador Bernoulli Naive Bayes de scikit-learn
from sklearn.naive_bayes import BernoulliNB

# Inicializar el clasificador
nb = BernoulliNB()

# Entrenar el clasificador con los datos de entrenamiento
nb.fit(X_train, y_train)

# Predecir las etiquetas del conjunto de prueba
y_hat = nb.predict(X_test)

# Importar herramientas necesarias para la evaluación
from sklearn import metrics
import matplotlib.pyplot as plt
import numpy as np

# Definir una función para graficar la matriz de confusión
def plot_confusion_matrix(y_pred, y):
    cm = metrics.confusion_matrix(y, y_pred)
    plt.imshow(cm, interpolation='nearest', cmap='Blues')
    plt.colorbar()
    plt.ylabel('Valor verdadero')
    plt.xlabel('Valor predicho')

    # Añadir los valores dentro de las celdas
    thresh = cm.max() / 2
    for i, j in np.ndindex(cm.shape):
        plt.text(j, i, cm[i, j], horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    # Ajustar el tamaño de la figura
    fig = plt.gcf()
    fig.set_size_inches(9, 9)

# Imprimir la precisión de la clasificación: la proporción de
instancias correctamente predichas
print("Precisión de la clasificación:", metrics.accuracy_score(y_test,
y_hat))

# Llamar a la función para graficar la matriz de confusión para las
predicciones de prueba
plot_confusion_matrix(y_hat, y_test)

# Imprimir un informe de clasificación detallado que muestra
precisión, recuperación, f1-score y soporte para cada clase
```

```python
print("Informe de Clasificación:")
print(metrics.classification_report(y_test, y_hat, zero_division=1))
```

```
Precisión de la clasificación: 0.4348993288590604
Informe de Clasificación:
              precision    recall  f1-score   support

           1       0.64      0.32      0.43       113
           2       0.67      0.01      0.01       280
           3       0.65      0.51      0.57       439
           4       1.00      0.00      0.00        30
           5       1.00      0.01      0.01       184
           6       0.96      0.13      0.23       198
           7       1.00      0.00      0.00        56
           8       1.00      0.00      0.00        72
          10       1.00      0.00      0.00       124
          12       0.43      0.46      0.44       438
          13       1.00      0.00      0.00        43
          14       1.00      0.00      0.00       145
          15       0.54      0.09      0.16       329
          16       0.57      0.54      0.55      1340
          17       1.00      0.03      0.06       141
          18       1.00      0.00      0.00        25
          19       0.34      0.81      0.48      1494
          20       0.45      0.75      0.57       939
          21       1.00      0.00      0.00        70
          24       1.00      0.00      0.00       151
          26       1.00      0.00      0.00       189
          27       1.00      0.00      0.00        28
          28       0.00      0.00      0.00       172
          29       0.60      0.35      0.44       231
          30       1.00      0.00      0.00        64
          31       1.00      0.00      0.00       106
          99       1.00      0.00      0.00        49

    accuracy                           0.43      7450
   macro avg       0.81      0.15      0.15      7450
weighted avg       0.59      0.43      0.36      7450
```




```
# Guardar los datos para uso futuro.
import pickle

# Abrir un archivo en modo binario de escritura
ofname = open('NYT_data.pkl', 'wb')

# Guardar los datos en el archivo usando pickle
s = pickle.dump([X_train, y_train, X_test, y_test], ofname)

# Cerrar el archivo después de guardar los datos
ofname.close()
```

```
# ¿Cuáles son las N características más predictivas por clase?
N = 5
# Obtener el vocabulario del vectorizador
voc = vectorizer.get_feature_names_out()  # Usar
`get_feature_names_out` en lugar de `get_feature_names`

# Iterar sobre cada etiqueta única en el conjunto de datos
for i, label in enumerate(np.unique(y)):
    # Obtener los índices de las características más predictivas para
la clase actual
    topN = np.argsort(nb.feature_log_prob_[i])[-N:]
    # Imprimir la etiqueta de la clase y los términos más predictivos
    print('Código: ' + str(label) + ' Términos: ' + str([voc[j] for j
in topN]))
```

```
Código: 1 Términos: ['cut', 'bush', 'economy', 'budget', 'tax']
Código: 2 Términos: ['race', 'gay', 'new', 'court', 'abortion']
Código: 3 Términos: ['care', 'medicare', 'drug', 'health', 'new']
Código: 4 Términos: ['farm', 'safety', 'new', 'farmers', 'food']
Código: 5 Términos: ['workers', 'strike', 'union', 'immigrants',
'new']
Código: 6 Términos: ['students', 'city', 'new', 'school', 'schools']
Código: 7 Términos: ['rules', 'warming', 'air', 'new', 'pollution']
Código: 8 Términos: ['blackout', 'california', 'power', 'energy',
'oil']
Código: 10 Términos: ['new', 'security', '800', 'flight', 'crash']
Código: 12 Términos: ['drug', 'case', 'death', 'new', 'police']
Código: 13 Términos: ['plan', 'security', 'new', 'social', 'welfare']
Código: 14 Términos: ['city', 'homeless', 'york', 'rent', 'new']
Código: 15 Términos: ['new', 'billion', 'deal', 'enron', 'microsoft']
Código: 16 Términos: ['bush', 'challenged', 'war', 'iraq', 'nation']
Código: 17 Términos: ['space', 'nasa', 'loss', 'new', 'shuttle']
Código: 18 Términos: ['business', 'bush', 'clinton', 'china', 'trade']
Código: 19 Términos: ['mideast', 'war', 'israel', 'new', 'china']
Código: 20 Términos: ['2000', 'clinton', 'bush', 'president',
'campaign']
Código: 21 Términos: ['park', 'plan', 'zero', 'ground', 'new']
Código: 24 Términos: ['mayor', 'giuliani', 'city', 'budget', 'new']
Código: 26 Términos: ['blizzard', 'new', 'overview', 'hurricane',
'storm']
Código: 27 Términos: ['blaze', 'ferry', 'killed', 'crash', 'fires']
Código: 28 Términos: ['arts', 'tv', 'broadway', 'art', 'new']
Código: 29 Términos: ['world', 'playoffs', 'series', 'yankees',
'baseball']
Código: 30 Términos: ['87', '79', 'crash', 'dead', 'dies']
Código: 31 Términos: ['faith', 'bishop', 'new', 'church', 'pope']
Código: 99 Términos: ['editors', 'today', 'readers', 'note',
'special']
```

Comprobemos qué ocurriría si enriqueciéramos el conjunto de datos con el resumen del artículo.

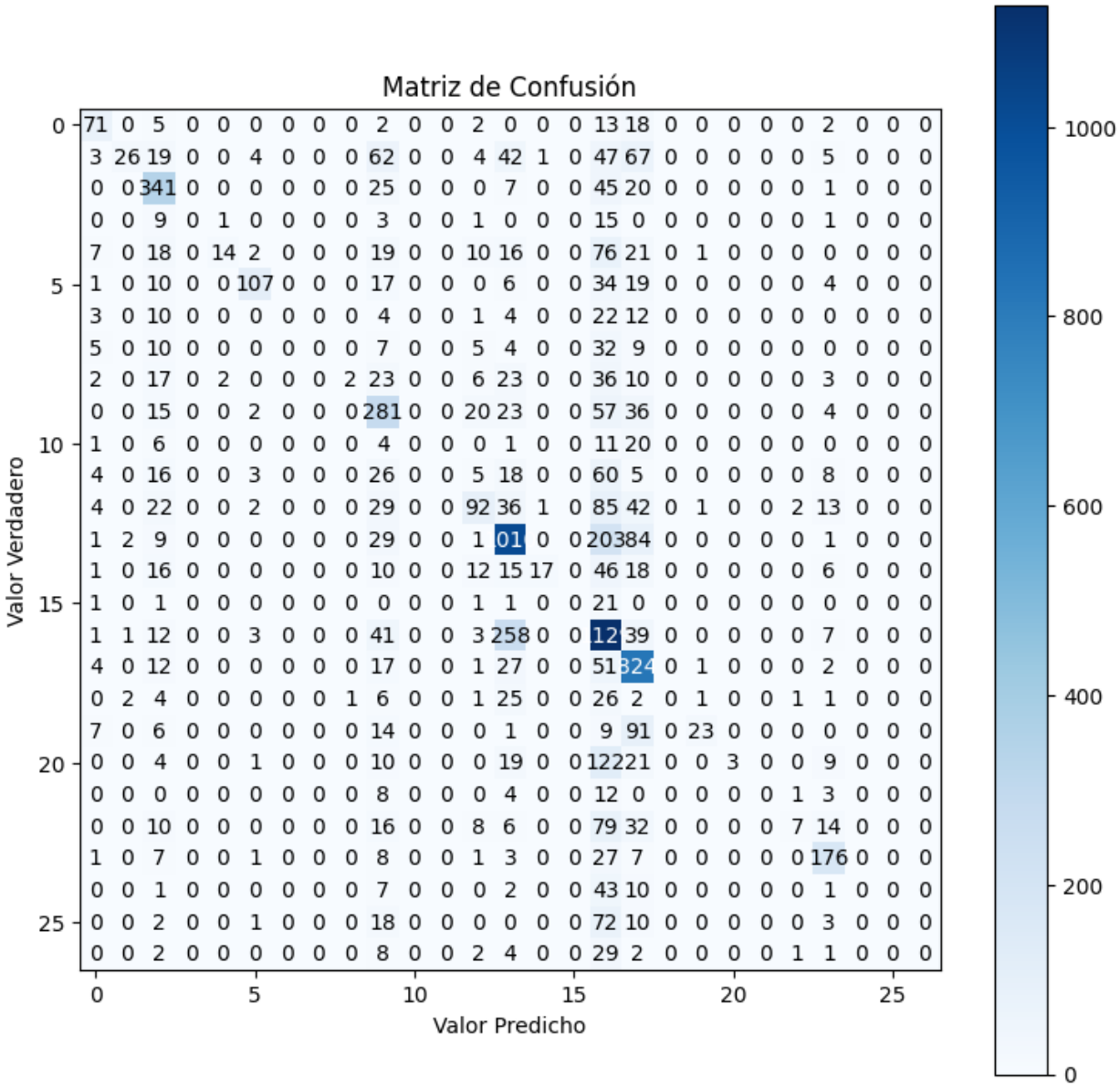
```
# Combinar las columnas 'Title' y 'Summary' en una sola serie de datos
raw_data = data['Title'] + " " + data['Summary']

# Dividir los datos combinados en conjuntos de entrenamiento y prueba
raw_train = raw_data[split]
raw_test = raw_data[np.logical_not(split)]

# Extraer las etiquetas de tema para el conjunto de datos
y = data['Topic_2digit']

# Dividir las etiquetas en conjuntos de entrenamiento y prueba
correspondientes a nuestros datos combinados
y_train = y[split]
y_test = y[np.logical_not(split)]

# Importar la herramienta necesaria para el procesamiento de texto
from sklearn.feature_extraction.text import CountVectorizer

# Inicializar el CountVectorizer con parámetros específicos
vectorizer = CountVectorizer(
    min_df=2,  # Una palabra debe aparecer en al menos dos documentos
para ser considerada
    stop_words='english',  # Eliminar palabras comunes en inglés (por
ejemplo, 'and', 'the', 'of') que no aportan mucho al significado
    strip_accents='unicode'  # Eliminar acentos de los caracteres para
mayor consistencia
)

# Ejemplo de texto
test_string = raw_train[0]
print("Ejemplo: " + test_string + "\n")
# Mostrar el resultado del preprocesamiento (por ejemplo, pasar a
minúsculas, eliminar puntuación)
print("Preprocesado: " + vectorizer.build_preprocessor()(test_string)
+ "\n")
# Mostrar la lista de palabras (tokens) después de dividir el texto
preprocesado
print("Tokenizado: " + str(vectorizer.build_tokenizer()(test_string))
+ "\n")
# Aplicar el analizador completo (preprocesamiento, tokenización y
filtrado de palabras vacías)
print("Cadena de datos analizada: " + str(vectorizer.build_analyzer()
(test_string)) + "\n")

# Ajustar el vectorizador y convertir los datos en matrices de conteo
de tokens
X_train = vectorizer.fit_transform(raw_train)  # Aprender el
```

```
vocabulario y transformar el conjunto de entrenamiento
X_test = vectorizer.transform(raw_test)  # Transformar el conjunto de
prueba basado en el vocabulario aprendido

# Imprimir el número total de tokens (palabras únicas) encontrados en
el conjunto de datos
print("\nNúmero de tokens: " +
str(len(vectorizer.get_feature_names_out())) + "\n")
# Mostrar una porción de los tokens para inspección
print("Extracto de tokens:")
print(vectorizer.get_feature_names_out()[1000:1100])
```

```
Ejemplo: Nation's Smaller Jails Struggle To Cope With Surge in Inmates
Jails overwhelmed with hardened criminals

Preprocesado: nation's smaller jails struggle to cope with surge in
inmates  jails overwhelmed with hardened criminals

Tokenizado: ['Nation', 'Smaller', 'Jails', 'Struggle', 'To', 'Cope',
'With', 'Surge', 'in', 'Inmates', 'Jails', 'overwhelmed', 'with',
'hardened', 'criminals']

Cadena de datos analizada: ['nation', 'smaller', 'jails', 'struggle',
'cope', 'surge', 'inmates', 'jails', 'overwhelmed', 'hardened',
'criminals']


Número de tokens: 11219

Extracto de tokens:
['bankruptcy' 'banks' 'banned' 'banner' 'banning' 'bans' 'baptist'
 'baptists' 'bar' 'barak' 'barbie' 'bare' 'barely' 'bares' 'bargain'
 'bargaining' 'bargains' 'barnes' 'barney' 'baron' 'barons' 'barrage'
 'barred' 'barrel' 'barren' 'barrier' 'barriers' 'barring' 'bars'
'barter'
 'base' 'baseball' 'based' 'basement' 'bases' 'bashing' 'basic'
'basics'
 'basis' 'baskeball' 'basket' 'basketball' 'basks' 'basra' 'bastion'
'bat'
 'bath' 'batter' 'battered' 'battering' 'batters' 'battery' 'battle'
 'battlefield' 'battleground' 'battles' 'battling' 'bay' 'bayer'
'bayou'
 'bazaar' 'bbc' 'beach' 'beaches' 'beachgoers' 'bear' 'bearing'
'bearings'
 'bears' 'beat' 'beaten' 'beating' 'beatings' 'beats' 'beautiful'
'beauty'
 'beckons' 'bed' 'bedevil' 'bedeviled' 'beds' 'beef' 'beetle' 'beg'
 'began' 'begin' 'beginning' 'begins' 'begun' 'behavior' 'behemoth'
 'beijing' 'beirut' 'belarus' 'belated' 'belfast' 'belgrade' 'beliefs'
 'belies' 'believe']
```

```python
from sklearn.naive_bayes import BernoulliNB
from sklearn import metrics
import matplotlib.pyplot as plt
import numpy as np

# Inicializar el clasificador Bernoulli Naive Bayes
nb = BernoulliNB()

# Ajustar el clasificador con el conjunto de entrenamiento
nb.fit(X_train, y_train)

# Realizar predicciones sobre el conjunto de prueba
y_hat = nb.predict(X_test)

# Función para graficar la matriz de confusión
def plot_confusion_matrix(y_true, y_pred):
    cm = metrics.confusion_matrix(y_true, y_pred)
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Matriz de Confusión')
    plt.colorbar()
    plt.ylabel('Valor Verdadero')
    plt.xlabel('Valor Predicho')

    # Añadir los valores dentro de las celdas
    thresh = cm.max() / 2
    for i, j in np.ndindex(cm.shape):
        plt.text(j, i, format(cm[i, j], 'd'), ha='center',
va='center',
                 color='white' if cm[i, j] > thresh else 'black')

    # Ajustar el tamaño de la figura
    fig = plt.gcf()
    fig.set_size_inches(9, 9)
    plt.show()

# Imprimir la exactitud de clasificación
print("Exactitud de clasificación:", metrics.accuracy_score(y_test,
y_hat))

# Graficar la matriz de confusión
plot_confusion_matrix(y_test, y_hat)

# Imprimir un reporte de clasificación detallado
print("Reporte de Clasificación:")
print(metrics.classification_report(y_test, y_hat, zero_division=0))
```

```
Exactitud de clasificación: 0.5534228187919463
```




Reporte de Clasificación:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.61 | 0.63 | 0.62 | 113 |
| 2 | 0.84 | 0.09 | 0.17 | 280 |
| 3 | 0.58 | 0.78 | 0.67 | 439 |
| 4 | 0.00 | 0.00 | 0.00 | 30 |
| 5 | 0.82 | 0.08 | 0.14 | 184 |
| 6 | 0.85 | 0.54 | 0.66 | 198 |
| 7 | 0.00 | 0.00 | 0.00 | 56 |
| 8 | 0.00 | 0.00 | 0.00 | 72 |
| 10 | 0.67 | 0.02 | 0.03 | 124 |
| 12 | 0.40 | 0.64 | 0.50 | 438 |

```
           13       0.00      0.00      0.00        43
           14       0.00      0.00      0.00       145
           15       0.52      0.28      0.36       329
           16       0.65      0.75      0.70      1340
           17       0.89      0.12      0.21       141
           18       0.00      0.00      0.00        25
           19       0.47      0.76      0.58      1494
           20       0.58      0.88      0.70       939
           21       0.00      0.00      0.00        70
           24       0.85      0.15      0.26       151
           26       1.00      0.02      0.03       189
           27       0.00      0.00      0.00        28
           28       0.58      0.04      0.08       172
           29       0.66      0.76      0.71       231
           30       0.00      0.00      0.00        64
           31       0.00      0.00      0.00       106
           99       0.00      0.00      0.00        49

    accuracy                           0.55      7450
   macro avg       0.41      0.24      0.24      7450
weighted avg       0.55      0.55      0.49      7450
```

```
import pickle

# Guardar los datos en un archivo para su uso futuro
with open('NYT_context_data.pkl', 'wb') as ofname:
    pickle.dump([X_train, y_train, X_test, y_test], ofname)

import numpy as np

# Número de características más predictivas a mostrar
N = 5

# Obtener los nombres de las características
voc = vectorizer.get_feature_names_out()

# Para cada clase, encontrar los términos más predictivos
for i, label in enumerate(np.unique(y)):
    # Obtener los índices de las características ordenadas por su log-
probabilidad
    topN = np.argsort(nb.feature_log_prob_[i])[-N:]
    # Imprimir la etiqueta de la clase y los términos más predictivos
    print('Código: ' + str(label) + ' Términos: ' + str([voc[i] for i
in topN]))
```

```
Código: 1 Términos: ['cut', 'economy', 'market', 'budget', 'tax']
Código: 2 Términos: ['gay', 'new', 'supreme', 'court', 'abortion']
Código: 3 Términos: ['medicare', 'care', 'drug', 'new', 'health']
Código: 4 Términos: ['meat', 'farm', 'new', 'farmers', 'food']
Código: 5 Términos: ['new', 'strike', 'workers', 'union',
```

```
'immigrants']
Código: 6 Términos: ['students', 'nyc', 'new', 'schools', 'school']
Código: 7 Términos: ['water', 'pollution', 'warming', 'global', 'new']
Código: 8 Términos: ['california', 'prices', 'oil', 'energy', 'power']
Código: 10 Términos: ['investigation', 'new', 'twa', 'flight',
'crash']
Código: 12 Términos: ['case', 'abuse', 'death', 'new', 'police']
Código: 13 Términos: ['security', 'social', 'clinton', 'new',
'welfare']
Código: 14 Términos: ['rent', 'housing', 'homeless', 'nyc', 'new']
Código: 15 Términos: ['scandal', 'new', 'antitrust', 'enron',
'microsoft']
Código: 16 Términos: ['bush', 'challenged', 'war', 'nation', 'iraq']
Código: 17 Términos: ['breakup', 'space', 'columbia', 'shuttle',
'new']
Código: 18 Términos: ['deal', 'chinese', 'clinton', 'china', 'trade']
Código: 19 Términos: ['israeli', 'peace', 'new', 'china', 'israel']
Código: 20 Términos: ['2000', 'bush', 'president', 'campaign',
'clinton']
Código: 21 Términos: ['zero', 'indian', 'ground', 'memorial', 'new']
Código: 24 Términos: ['governor', 'budget', 'nyc', 'mayor', 'new']
Código: 26 Términos: ['earthquake', 'hurricane', 'new', 'storm',
'weather']
Código: 27 Términos: ['accident', 'killed', 'crash', 'new', 'fires']
Código: 28 Términos: ['york', 'museum', 'day', 'art', 'new']
Código: 29 Términos: ['series', 'world', 'yankees', 'olympics',
'baseball']
Código: 30 Términos: ['john', 'crash', 'death', 'dead', 'dies']
Código: 31 Términos: ['catholic', 'religious', 'new', 'church',
'pope']
Código: 99 Términos: ['special', 'man', 'new', 'editors', 'note']
```

Obsérvese que añadir el pequeño resumen mejora la tasa de reconocimiento en $10\%$.

Como nota al margen, Naive Bayes con estos modelos crea un límite de decisión lineal. Por esta razón, a veces NB se llama un clasificador lineal.

# 2. Máquinas de Vectores de Soporte (Support Vector Machines).

Las Máquinas de Vectores de Soporte (SVM) son un método poderoso utilizado en aprendizaje automático para tareas de clasificación. Pertenecen a una categoría de algoritmos conocida como aprendizaje discriminativo, donde el objetivo es encontrar un límite de decisión que separe las diferentes clases en los datos.

## ¿Qué hace especial a SVM?

A diferencia de otros modelos lineales como el perceptrón o la regresión logística, SVM ofrece un enfoque más robusto para la clasificación. Aquí está el porqué:

- **Modelado explícito de límites**: SVM se centra en encontrar el mejor límite posible (o hiperplano en dimensiones superiores) que separe las clases. Este límite se elige no solo para separar las clases, sino para hacerlo de manera que maximice el margen entre los puntos más cercanos de las clases al límite. Estos puntos más cercanos son conocidos como vectores de soporte, lo que da nombre al algoritmo.
- **Versatilidad**: Mientras que el modelo clásico para SVM es lineal, puede extenderse para manejar la clasificación no lineal utilizando algo llamado el truco del kernel. Esto permite que SVM clasifique datos que no son separables linealmente transformándolos en un espacio de mayor dimensión donde sí existe un separador lineal.

## La Intuición Detrás de SVM

Imagina que estás tratando de dibujar una línea que separe manzanas de naranjas en una mesa. SVM tiene como objetivo dibujar esta línea no solo en cualquier lugar, sino de tal manera que la distancia más pequeña desde la línea hasta la manzana o naranja más cercana sea maximizada. Esto asegura que el límite de decisión esté lo más lejos posible de los puntos más cercanos de cada clase, proporcionando un buffer que ayuda a que la clasificación sea más robusta frente a nuevos puntos de datos.

- **Maximizar el Margen**: La idea clave es encontrar la "calle" más ancha posible (margen) entre las clases, con los "bordes" de esta calle tocando justo los puntos más cercanos de cada clase. Estos puntos en el borde son los vectores de soporte.
- **Manejo de Datos Más Complejos**: Para datos que no pueden ser separados por una línea recta, SVM utiliza el truco del kernel para proyectar los datos en un espacio de mayor dimensión donde es posible un separador lineal. Esto es como levantar las manzanas y naranjas de la mesa al aire para encontrar un plano que los separe.

## Conclusión

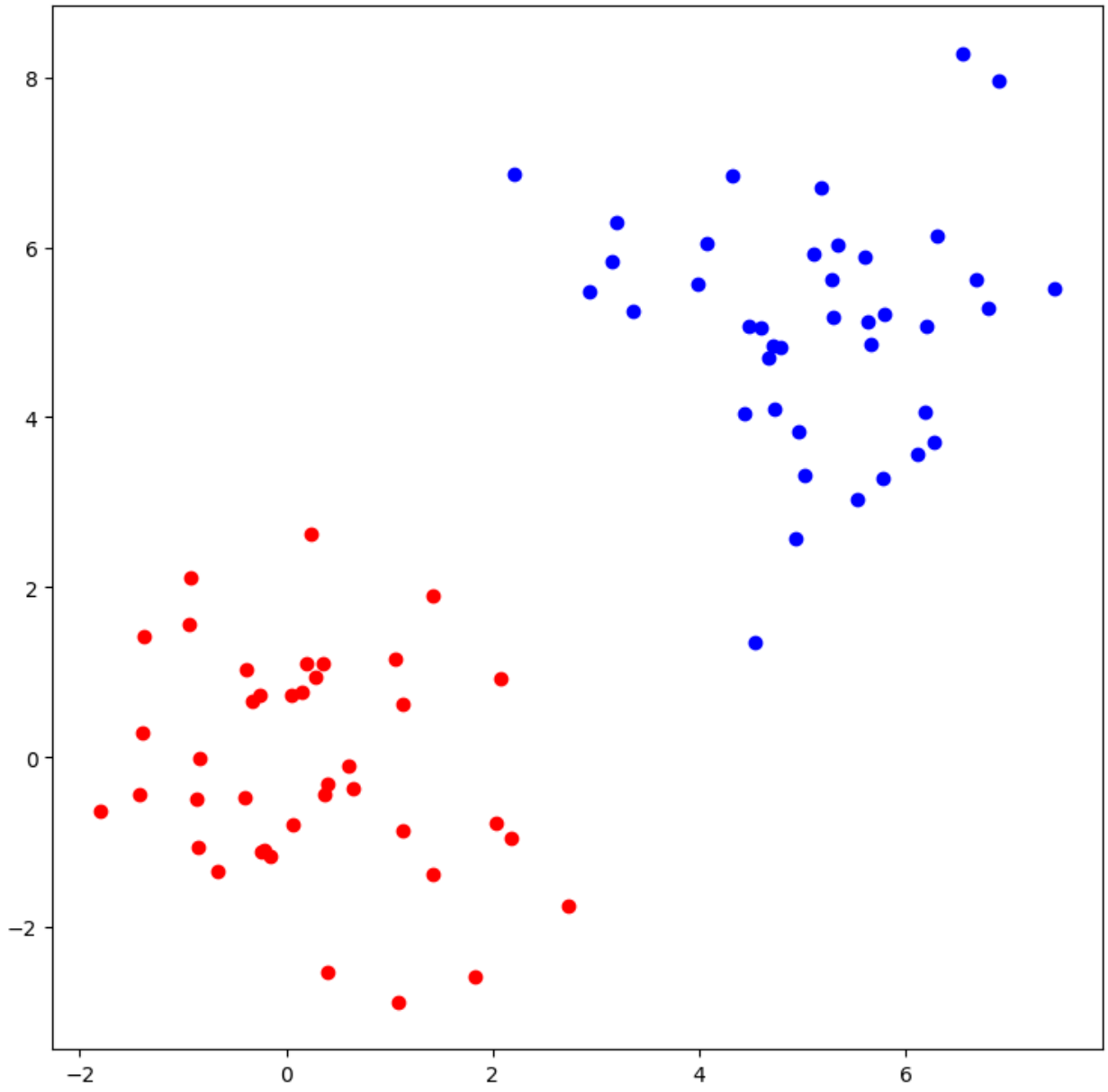
SVM es una herramienta fundamental en el aprendizaje automático, conocida por su capacidad para crear límites claros y bien definidos entre clases. Su capacidad para manejar datos tanto lineales como no lineales la convierte en una opción versátil para una amplia gama de problemas de clasificación. Entender los principios detrás de SVM permite obtener una visión más profunda de cómo las máquinas pueden aprender a distinguir entre diferentes categorías de datos, lo que la convierte en una parte crucial de cualquier conjunto de herramientas de aprendizaje automático.

VIDEO: Enlace SVM

```
%matplotlib inline
%reset -f
```

```
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, FloatSlider

class HLA():
    def __init__(self):
        np.random.seed(1)
        # Generar datos sintéticos para dos clases
        self.X = np.concatenate([1.25*np.random.randn(40,2),
5+1.5*np.random.randn(40,2)])
        self.y = np.concatenate([np.ones((40,1)), -np.ones((40,1))])

        # Visualizar los datos
        plt.figure(figsize=(9, 9))
        plt.scatter(self.X[0:40,0], self.X[0:40,1], color='r',
label='Class 1')
        plt.scatter(self.X[40:,0], self.X[40:,1], color='b',
label='Class 2')

        # Crear una malla para la visualización
        delta = 0.025
        xx = np.arange(-5.0, 10.0, delta)
        yy = np.arange(-5.0, 10.0, delta)
        XX, YY = np.meshgrid(xx, yy)
        Xf = XX.flatten()
        Yf = YY.flatten()
        self.sz = XX.shape
        self.data = np.concatenate([Xf[:, np.newaxis], Yf[:,
np.newaxis]], axis=1)

    def run(self, w0, w1, offset):
        # Ajustar los pesos y el sesgo en el modelo lineal
        w = np.array([w0, w1]).reshape(2, 1)

        # Calcular el valor de la función de decisión
        Z = self.data.dot(w) + offset
        Z = Z.reshape(self.sz)

        # Visualizar los datos y la frontera de decisión
        plt.figure(figsize=(9, 9))
        plt.scatter(self.X[0:40,0], self.X[0:40,1], color='r',
label='Class 1')
        plt.scatter(self.X[40:,0], self.X[40:,1], color='b',
label='Class 2')
        plt.imshow(Z, interpolation='bilinear', origin='lower',
extent=(-5, 10, -5, 10), alpha=0.3, vmin=-30, vmax=30)
        XX = self.data[:,0].reshape(self.sz)
        YY = self.data[:,1].reshape(self.sz)
        plt.contour(XX, YY, Z, [0], colors='k')  # Línea de la
```

```
frontera de decisión

        # Configurar y mostrar la visualización
        plt.title('Decision Boundary Visualization')
        plt.xlabel('Feature 1')
        plt.ylabel('Feature 2')
        plt.legend()
        plt.show()

def decorator(w0, w1, offset):
    widget_hla.run(w0, w1, offset)

# Crear una instancia de la clase HLA
widget_hla = HLA()

# Configurar widgets interactivos para ajustar los parámetros del
modelo
interact(decorator,
         w0=FloatSlider(value=0, min=-10.0, max=10.0, step=0.1,
description='w0'),
         w1=FloatSlider(value=0, min=-10.0, max=10.0, step=0.1,
description='w1'),
         offset=FloatSlider(value=0, min=-20.0, max=40.0, step=0.1,
description='offset'))
```




{"model_id":"4486bd19497a49149ce7fbdf2e85385e","version_major":2,"version_minor":0}

```
<function __main__.decorator(w0, w1, offset)>
```

Observaciones:

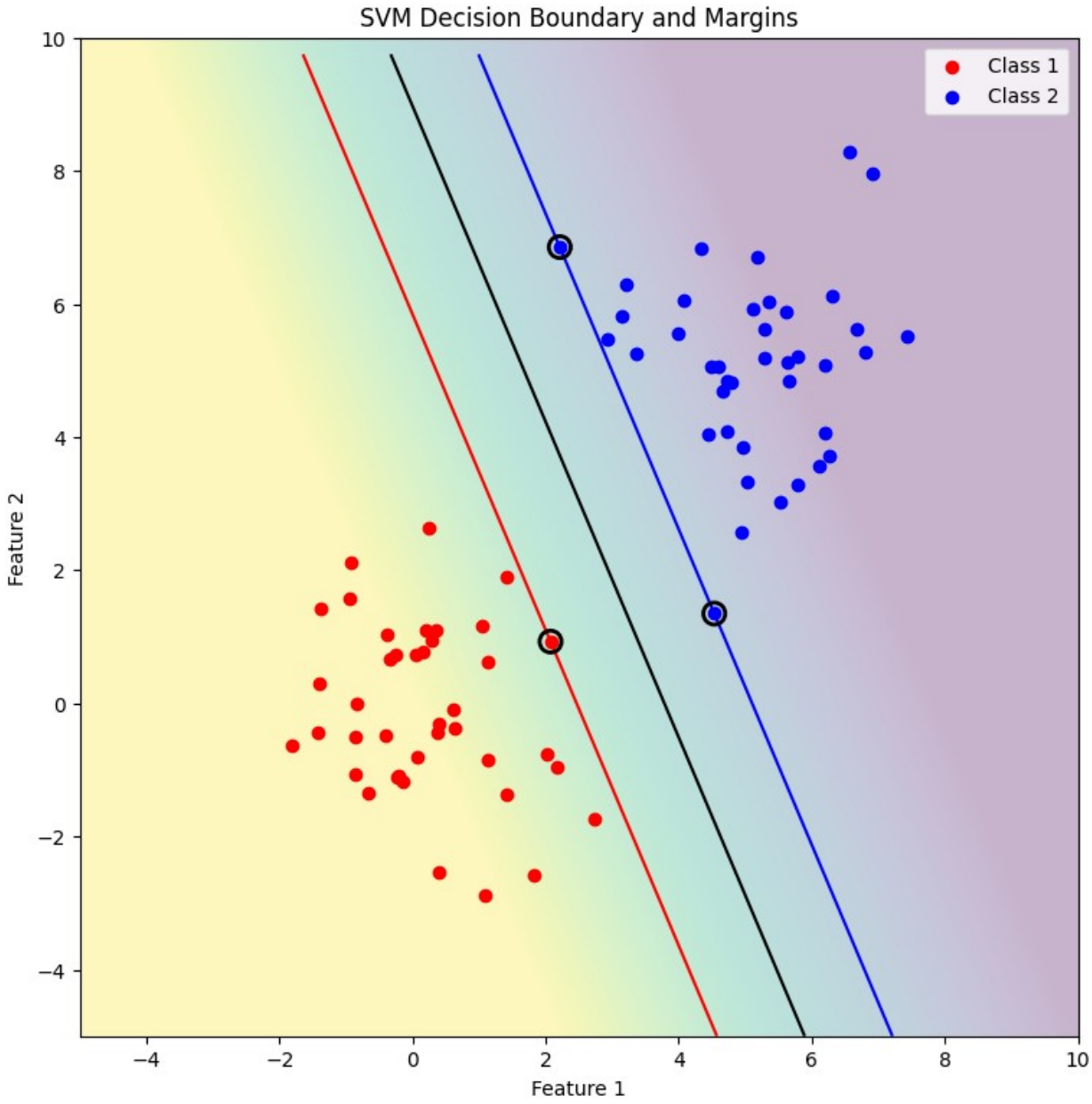
- Modela implícitamente la noción de ruido. Se espera que el límite con margen máximo sea robusto a pequeñas perturbaciones en los datos.
- Un clasificador con margen máximo tiene una solución única en el caso separable.

Comprobemos el resultado de ajustar un clasificador SVM utilizando sklearn:

```
%matplotlib inline
%reset -f

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm  # Import SVM model from scikit-learn

class svm_example():
    def __init__(self):
        '''Data creation: Generates synthetic data for
classification'''
        np.random.seed(1)  # Set seed for reproducibility
        # Generate synthetic data: two clusters with normal
distribution
        self.X = np.concatenate([1.25*np.random.randn(40,2),
5+1.5*np.random.randn(40,2)])
        # Generate labels: first 40 are 1, next 40 are -1
        self.y = np.concatenate([np.ones((40,1)), -np.ones((40,1))])

    def run(self):
        '''Fit a linear SVM: Train an SVM classifier with a linear
kernel'''
        self.clf = svm.SVC(kernel='linear')  # Initialize the SVM with
a linear kernel
        self.clf.fit(self.X, self.y.ravel())  # Fit the SVM model with
the data

    def display(self):
        '''Display the decision boundary, margins, and support
vectors'''
        # Create a mesh grid for plotting decision boundary
        delta = 0.25
        xx = np.arange(-5.0, 10.0, delta)
        yy = np.arange(-5.0, 10.0, delta)
        XX, YY = np.meshgrid(xx, yy)

        # Prepare the grid points for prediction
        Xf = XX.flatten()
        Yf = YY.flatten()
        sz = XX.shape
        data = np.concatenate([Xf[:, np.newaxis], Yf[:, np.newaxis]],
axis=1)

        # Predict the decision function value for each grid point
        Z = self.clf.decision_function(data)
        Z.shape = sz

        # Plot the data points: red for one class, blue for the other
        plt.figure(figsize=(9, 9))
        plt.scatter(self.X[0:40, 0], self.X[0:40, 1], color='r',
```

```
label='Class 1')
        plt.scatter(self.X[40:, 0], self.X[40:, 1], color='b',
label='Class 2')

        # Display the decision boundary and margins
        plt.imshow(Z, interpolation='bilinear', origin='lower',
extent=(-5, 10, -5, 10), alpha=0.3, vmin=-3, vmax=3)
        plt.contour(XX, YY, Z, [-1, 0, 1], colors=['b', 'k', 'r'])  #
Draw the margins and decision boundary

        # Enhance the plot
        plt.title('SVM Decision Boundary and Margins')
        plt.xlabel('Feature 1')
        plt.ylabel('Feature 2')
        plt.legend()

        # Print the number of support vectors for each class
        print('Number of support vectors: ' +
str(np.sum(self.clf.n_support_)))

        # Highlight the support vectors on the plot
        plt.scatter(self.clf.support_vectors_[:, 0],
                    self.clf.support_vectors_[:, 1],
                    s=120,
                    facecolors='none',
                    edgecolors='k',
                    linewidths=2,
                    zorder=10,
                    label='Support Vectors')

        # Print the coefficients of the decision function (w0, w1) and
the offset
        print('(w0, w1) = ' + str(self.clf.coef_[0]))
        print('Offset = ' + str(self.clf.intercept_[0]))

        # Return grid for further use if necessary
        return XX, YY, Z

# Create an instance of the svm_example class
c = svm_example()
c.run()  # Train the SVM model
XX, YY, Z = c.display()  # Display the results
```

```
Number of support vectors: 3
(w0, w1) = [-0.75827758 -0.31976526]
Offset = 2.869764740799983
```




Observa que hay un subconjunto crítico de puntos de datos. Se denominan **vectores de apoyo**. Si alguno de esos puntos desaparece, el límite cambia. El límite de decisión depende de los vectores de soporte, por lo que tenemos que almacenarlos en nuestro modelo.

Comprueba la intuición en 3D:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from ipywidgets import interact
```

```
# Generar datos sintéticos
np.random.seed(1)
X = np.concatenate([1.25*np.random.randn(40,2),
5+1.5*np.random.randn(40,2)])
y = np.concatenate([np.ones((40,1)), -np.ones((40,1))])

# Definir la función de decisión (esto debe ser calculado con el
modelo entrenado)
# Aquí usamos una función ficticia Z para ilustrar
delta = 0.25
xx = np.arange(-5.0, 10.0, delta)
yy = np.arange(-5.0, 10.0, delta)
XX, YY = np.meshgrid(xx, yy)
Z = np.sin(XX) + np.cos(YY)  # Ejemplo de función de decisión

def control3D(elevation, azimuth):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    fig.set_size_inches(12,12)

    # Dibujar la superficie de la función de decisión
    surf = ax.plot_surface(XX, YY, Z, cmap=cm.coolwarm, alpha=0.3,
linewidth=0)

    # Dibujar los puntos de datos
    ax.scatter(X[0:40, 0], X[0:40, 1], 1, color='r', label='Class 1')
    ax.scatter(X[40:, 0], X[40:, 1], -1, color='b', label='Class 2')

    # Añadir contornos si es necesario
    # ax.contour(XX, YY, Z, [-1, 0, 1], colors=['b', 'k', 'r'])  # No
funciona directamente en 3D

    # Ajustar vista
    ax.view_init(elev=elevation, azim=azimuth)

    # Añadir leyenda
    ax.legend()

# Interactividad con widgets
interact(control3D, elevation=(0, 90), azimuth=(0, 360))
```

{"model_id":"262178e8e2524386969d1e9d83f90b23","version_major":2,"version_minor":0}

```
<function __main__.control3D(elevation, azimuth)>
```

# 2.1 Modelando Máquinas de Vectores de Soporte.

## 2.1.1 Geometría del hiperplano

Un hiperplano en ${\bf R}^d$ se define como una combinación afín de las variables:
$\pi \equiv a^T x + b = 0$.

Características:

- Un hiperplano divide el espacio en dos semiespacios. La evaluación de la ecuación del hiperplano en cualquier elemento de uno de los semiespacios es un valor positivo. Es un valor negativo para todos los elementos en el otro semiespacio.
- La distancia de un punto $x \in{\bf R}^d$ al hiperplano $\pi$ es

$$d(x, \pi) = \frac{a^T x + b}{\| a \|_2}$$

## 3.1.2 Modelando el hiperplano separador

Dado un problema de clasificación binaria con datos de entrenamiento $D = \{(x_i, y_i)\}, i = 1 \ldots N, y_i \in \{+1, -1\}$. Consideremos $S \subseteq D$ el subconjunto de todos los puntos de datos que pertenecen a la clase +1, $S = \{x_i \vee y_i = +1\}$, y $R = \{x_i \vee y_i = -1\}$ su complemento.

Entonces, el problema de encontrar un hiperplano separador consiste en cumplir con las siguientes restricciones

$$a^T s_i + b > 0 \text{ y } a^T r_i + b < 0 \; \forall \, s_i \in S, r_i \in R.$$

Note las desigualdades estrictas en la formulación. Informalmente, podemos considerar la restricción mínima satisfecha. Y observar que el resto debe satisfacerse con un valor mayor. Así, podemos establecer arbitrariamente ese valor a 1 y reescribir el problema como

$$a^T s_i + b \geq 1 \text{ y } a^T r_i + b \leq -1.$$

Este es un *problema de factibilidad* y usualmente se escribe de la siguiente manera en notación estándar de optimización

$$\begin{array}{ll} \text{minimizar} & 1 \\ \text{sujeto a} & a^T r_i + b \leq -1, \forall \, r_i \in R \\ \text{¿} & \text{¿} \end{array}$$

o de manera compacta

$$\begin{array}{ll} \text{minimizar} & 1 \\ \text{sujeto a} & y_i \left( a^T x_i + b \right) \geq 1, \forall \, x_i \in D \end{array}$$

La solución de este problema no es única, por ejemplo, recuerde todos los parámetros del 'Algoritmo de Aprendizaje Humano'.

## 2.1.3 El hiperplano de margen máximo

Seleccionar el hiperplano de margen máximo requiere agregar una nueva restricción a nuestro problema. Recuerde de la geometría del hiperplano que la distancia de cualquier punto a un hiperplano está dada por $d(x,\pi)=\frac{a^T x+b}{\| a \|_2}$.

Recuerde que queremos que los datos positivos estén más allá del valor 1 y los datos negativos por debajo de -1. Entonces, ¿cuál es el valor de distancia que queremos maximizar?

El punto positivo más cercano al límite está en $1/\| a \|_2$ y el punto de dato negativo más cercano al límite también está en $1/\| a \|_2$. Por lo tanto, los puntos de datos de diferentes clases están al menos a $2/\| a \|_2$ de distancia.

Recuerde que nuestro objetivo es encontrar el hiperplano separador con el máximo margen, es decir, con la máxima distancia entre elementos de diferentes clases. Por lo tanto, podemos completar la formulación anterior con nuestro último requisito de la siguiente manera

$$\begin{aligned} &\text{maximizar} && 2/\| a \|_2 \\ &\text{sujeto a} && y_i\left(a^T x_i+b\right)\geq 1, \forall x_i en D \end{aligned}$$

o equivalentemente,

$$\begin{aligned} &\text{minimizar} && \| a \|_2/2 \\ &\text{sujeto a} && y_i\left(a^T x_i+b\right)\geq 1, \forall x_i en D \end{aligned}$$

Esta formulación tiene una solución siempre que el problema sea linealmente separable.

## 2.1.4 Tratando con el caso no separable

Para tratar con las clasificaciones erróneas, vamos a introducir un nuevo conjunto de variables $\xi_i$, que representa la cantidad de violación en la restricción $i$-ésima. Si la restricción ya está satisfecha, entonces $\xi_i=0$, y $\xi_i>0$ de lo contrario. Dado que $\xi_i$ está relacionado con los errores, nos gustaría mantener esta cantidad lo más cercana a cero posible. Esto nos lleva a introducir un elemento en el objetivo que compensa con el margen máximo.

El nuevo modelo se convierte en

$$\begin{aligned} &\text{minimizar} && \| a \|_2/2+C\sum_{i=1}^{N}\xi_i \\ &\text{sujeto a} && y_i\left(a^T x_i+b\right)\geq 1-\xi_i, i=1\ldots N \\ &\text{¿} && \text{¿} \end{aligned}$$

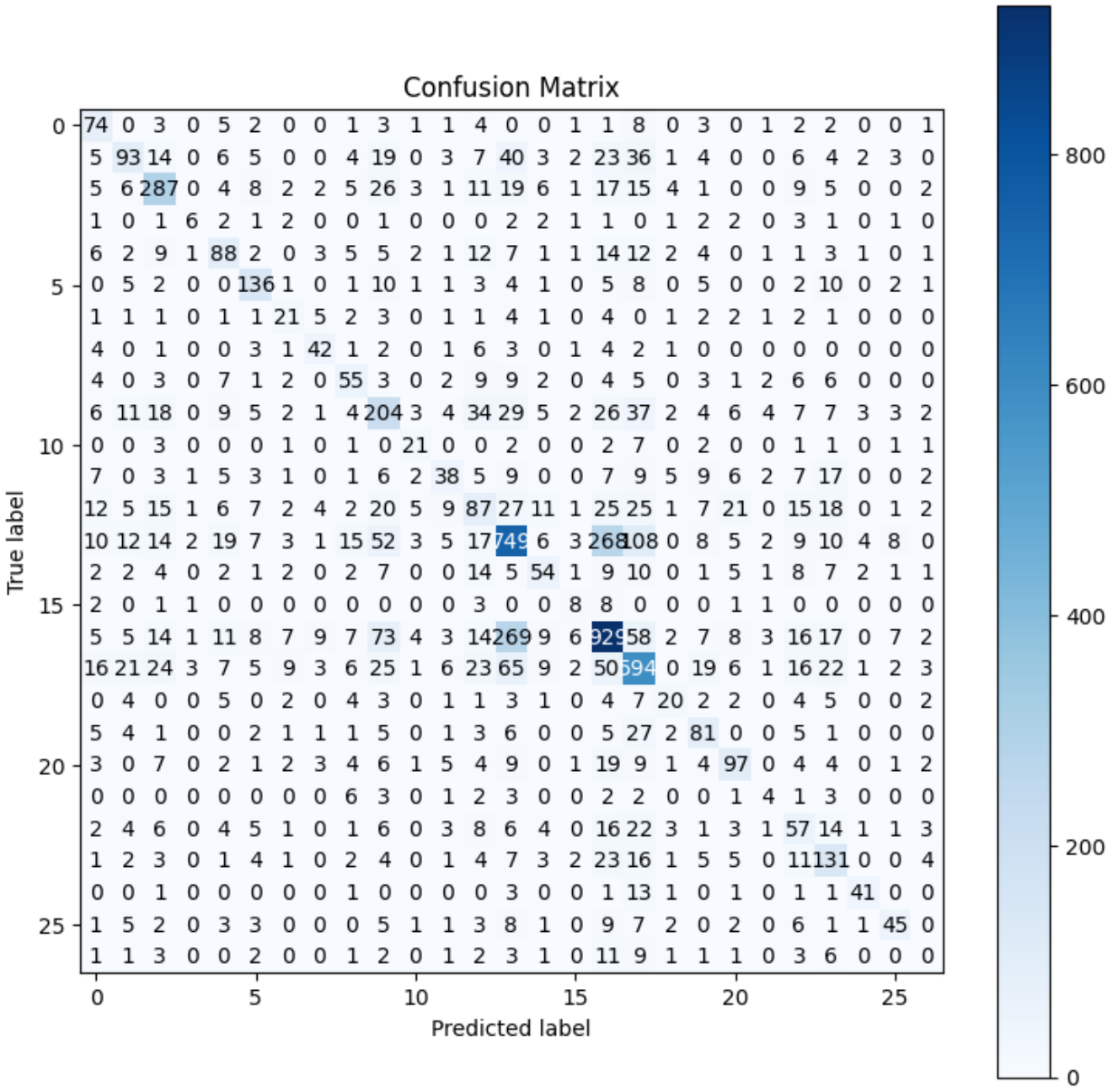
donde $C$ es el parámetro de compensación que equilibra aproximadamente el margen y la tasa de clasificación errónea. Esta formulación también se llama **SVM de margen suave**.

## 2.1.5 El problema del New York Times otra vez

Vamos a aplicar nuestro conocimiento al predictor de temas de titulares del New York Times.

```
import pickle
from sklearn import svm, metrics
import matplotlib.pyplot as plt
import numpy as np

# Cargar datos desde el archivo pickle
with open('NYT_data.pkl', 'rb') as fname:
    data = pickle.load(fname)
X_train = data[0]
y_train = data[1]
X_test = data[2]
y_test = data[3]
print('Loading ok.')

# Inicializar y entrenar el clasificador SVM lineal
clf = svm.LinearSVC()
clf.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_hat = clf.predict(X_test)

# Función para plotear la matriz de confusión
def plot_confusion_matrix(y_pred, y):
    conf_matrix = metrics.confusion_matrix(y, y_pred)
    plt.imshow(conf_matrix, interpolation='nearest',
cmap=plt.cm.Blues)
    plt.colorbar()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title('Confusion Matrix')

    # Añadir los valores dentro de las celdas
    thresh = conf_matrix.max() / 2
    for i, j in np.ndindex(conf_matrix.shape):
        plt.text(j, i, format(conf_matrix[i, j], 'd'), ha='center',
va='center',
                 color='white' if conf_matrix[i, j] > thresh else
'black')

    fig = plt.gcf()
    fig.set_size_inches(9, 9)
    plt.show()

# Evaluar el rendimiento del modelo
print("Classification accuracy:", metrics.accuracy_score(y_test,
y_hat))
plot_confusion_matrix(y_hat, y_test)
print("Classification Report:")
print(metrics.classification_report(y_test, y_hat))
```

```
Loading ok.
Classification accuracy: 0.5318120805369128
```



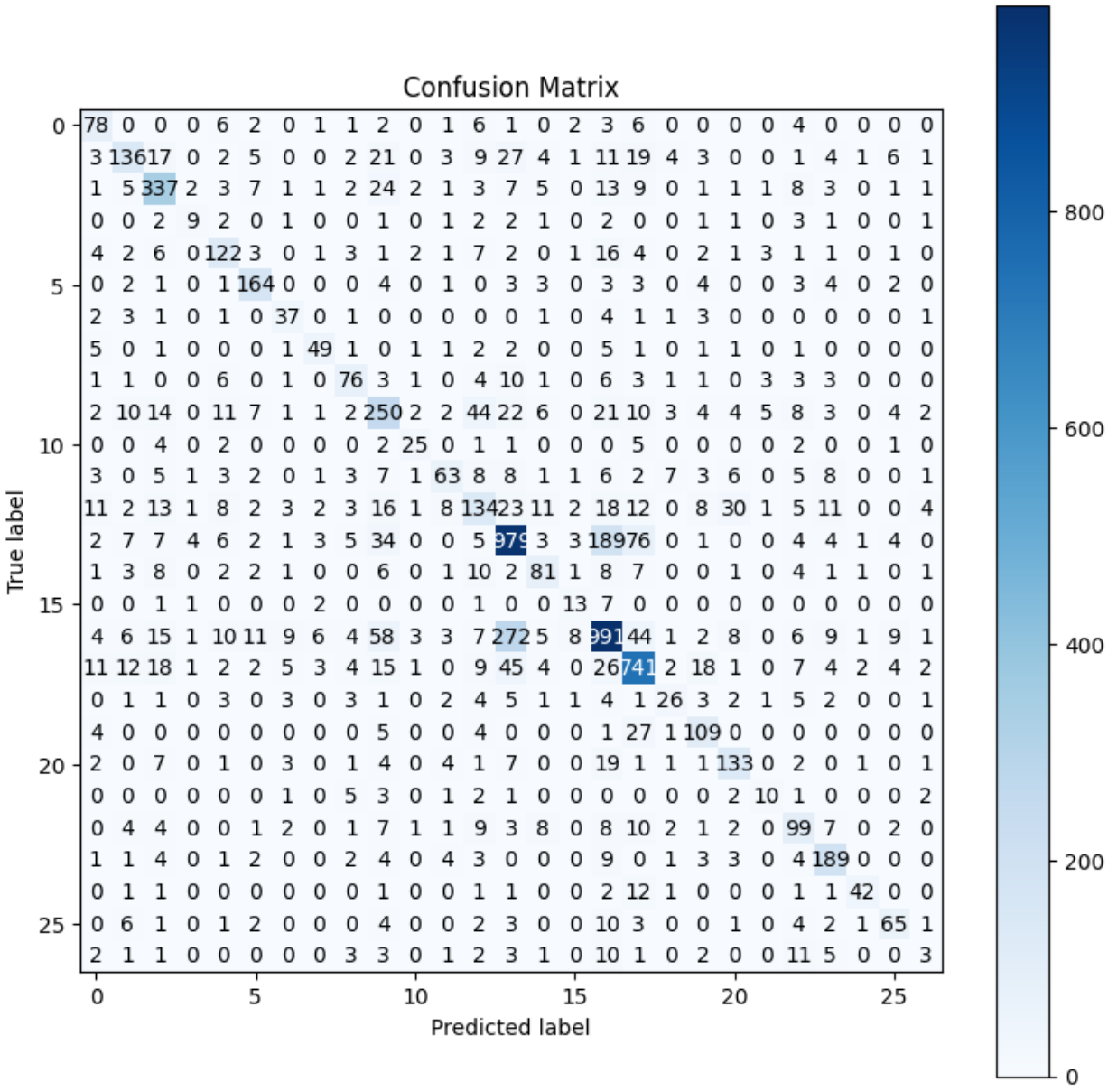

```
Classification Report:
              precision    recall  f1-score   support

           1       0.43      0.65      0.52       113
           2       0.51      0.33      0.40       280
           3       0.65      0.65      0.65       439
           4       0.38      0.20      0.26        30
           5       0.47      0.48      0.47       184
           6       0.64      0.69      0.66       198
```

```
           7       0.33      0.38      0.35        56
           8       0.57      0.58      0.58        72
          10       0.42      0.44      0.43       124
          12       0.41      0.47      0.44       438
          13       0.44      0.49      0.46        43
          14       0.42      0.26      0.32       145
          15       0.31      0.26      0.29       329
          16       0.58      0.56      0.57      1340
          17       0.45      0.38      0.41       141
          18       0.24      0.32      0.28        25
          19       0.62      0.62      0.62      1494
          20       0.57      0.63      0.60       939
          21       0.39      0.29      0.33        70
          24       0.46      0.54      0.50       151
          26       0.55      0.51      0.53       189
          27       0.17      0.14      0.15        28
          28       0.28      0.33      0.30       172
          29       0.44      0.57      0.50       231
          30       0.73      0.64      0.68        64
          31       0.59      0.42      0.49       106
          99       0.00      0.00      0.00        49

    accuracy                           0.53      7450
   macro avg       0.45      0.44      0.44      7450
weighted avg       0.53      0.53      0.53      7450
```

```python
# Recover NTY data
import pickle
from sklearn import svm, metrics
import matplotlib.pyplot as plt
import numpy as np

# Cargar datos desde el archivo pickle
with open('NYT_context_data.pkl', 'rb') as fname:
    data = pickle.load(fname)
X_train = data[0]
y_train = data[1]
X_test = data[2]
y_test = data[3]
print('Loading ok.')

# Inicializar y entrenar el clasificador SVM lineal
clf = svm.LinearSVC()
clf.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_hat = clf.predict(X_test)

# Función para plotear la matriz de confusión
```

```
def plot_confusion_matrix(y_true, y_pred):
    conf_matrix = metrics.confusion_matrix(y_true, y_pred)
    plt.imshow(conf_matrix, interpolation='nearest',
cmap=plt.cm.Blues)
    plt.colorbar()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title('Confusion Matrix')

    # Añadir los valores dentro de las celdas
    thresh = conf_matrix.max() / 2
    for i, j in np.ndindex(conf_matrix.shape):
        plt.text(j, i, format(conf_matrix[i, j], 'd'), ha='center',
va='center',
                 color='white' if conf_matrix[i, j] > thresh else
'black')

    fig = plt.gcf()
    fig.set_size_inches(9, 9)
    plt.show()

# Evaluar el rendimiento del modelo
print("Classification accuracy:", metrics.accuracy_score(y_test,
y_hat))
plot_confusion_matrix(y_test, y_hat)
print("Classification Report:")
print(metrics.classification_report(y_test, y_hat))
```

```
Loading ok.
Classification accuracy: 0.6659060402684563
```




```
Classification Report:
              precision    recall  f1-score   support

           1       0.57      0.69      0.62       113
           2       0.67      0.49      0.56       280
           3       0.72      0.77      0.74       439
           4       0.45      0.30      0.36        30
           5       0.63      0.66      0.65       184
           6       0.77      0.83      0.80       198
           7       0.53      0.66      0.59        56
           8       0.70      0.68      0.69        72
          10       0.62      0.61      0.62       124
          12       0.53      0.57      0.55       438
```

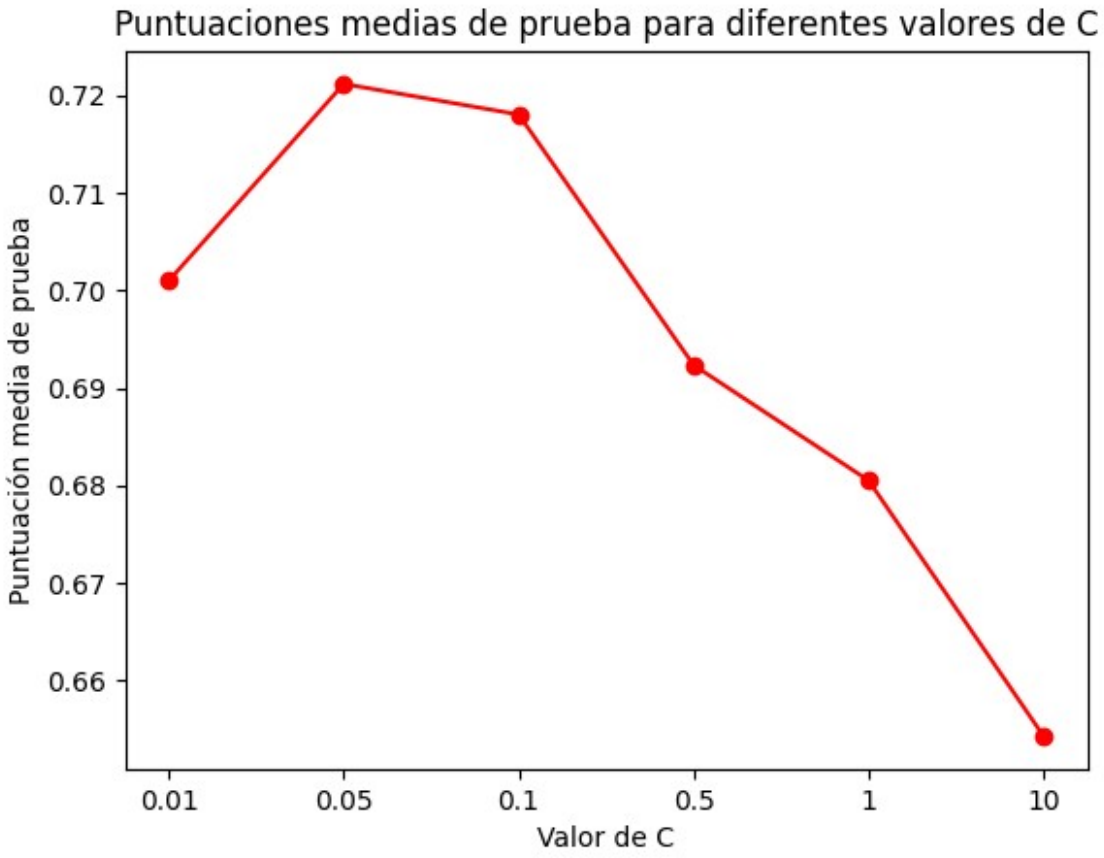
```
           13       0.62      0.58      0.60        43
           14       0.64      0.43      0.52       145
           15       0.48      0.41      0.44       329
           16       0.69      0.73      0.71      1340
           17       0.60      0.57      0.58       141
           18       0.39      0.52      0.45        25
           19       0.71      0.66      0.69      1494
           20       0.74      0.79      0.77       939
           21       0.51      0.37      0.43        70
           24       0.64      0.72      0.68       151
           26       0.68      0.70      0.69       189
           27       0.42      0.36      0.38        28
           28       0.52      0.58      0.54       172
           29       0.72      0.82      0.77       231
           30       0.84      0.66      0.74        64
           31       0.66      0.61      0.63       106
           99       0.13      0.06      0.08        49

    accuracy                           0.67      7450
   macro avg       0.60      0.59      0.59      7450
weighted avg       0.66      0.67      0.66      7450
```

Utilizando los parámetros por defecto podemos mejorar la tasa de reconocimiento en $10\,\%$. Sin embargo, no podemos comprobar las palabras más importantes. ¿Podemos encontrar una solución mejor?

```python
# Importando las bibliotecas necesarias para la selección de modelos
from sklearn import model_selection
from sklearn import svm

# Definiendo la cuadrícula de parámetros:
# 'C' es un parámetro de regularización para LinearSVC. Controla el
equilibrio entre lograr un bajo error de entrenamiento y un
# bajo error de prueba (generalización).
# Un valor más pequeño de 'C' conduce a una frontera de decisión más
suave (menos ajuste a los datos de entrenamiento),
# mientras que un valor más grande de 'C' fomenta que el modelo
clasifique correctamente todos los ejemplos de entrenamiento al
# darle al modelo más flexibilidad.
# Aquí, estamos definiendo un rango de valores de 'C' para probar con
GridSearchCV para encontrar el mejor.
parameters = {'C': [0.01, 0.05, 0.1, 0.5, 1, 10]}

# Inicializando el modelo LinearSVC
svc = svm.LinearSVC()

# Configurando GridSearchCV:
# 'svc' es el modelo SVM con un kernel lineal que se va a entrenar.
```

```
# 'parameters' contiene la cuadrícula de parámetros (valores de 'C' en este caso) que queremos probar.
# GridSearchCV trabajará sistemáticamente a través de las combinaciones de parámetros (diferentes valores de 'C'),
# entrenará el modelo para cada combinación y evaluará su rendimiento.
clf = model_selection.GridSearchCV(svc, parameters)

# Ajustando GridSearchCV:
# Esto entrenará el modelo LinearSVC varias veces con los diferentes valores de 'C' especificados en 'parameters'.
# Para cada valor de 'C', utiliza validación cruzada para evaluar el rendimiento del modelo.
# La validación cruzada es una técnica para evaluar cómo los resultados de un análisis estadístico se generalizarán a
# un conjunto de datos independiente.
# Lo hace dividiendo el conjunto de datos de entrenamiento original en un conjunto de entrenamiento para entrenar el modelo,
# y un conjunto de validación para evaluarlo. Este proceso se repite para cada valor de 'C'.
clf.fit(X_train, y_train)

# Después de que .fit() se complete, clf (nuestro objeto GridSearchCV) contendrá mucha información:
# - El mejor valor de 'C' encontrado.
# - El modelo ajustado con el mejor valor de 'C'.
# - Las puntuaciones o métricas de rendimiento para cada valor de 'C' probado.

GridSearchCV(estimator=LinearSVC(),
             param_grid={'C': [0.01, 0.05, 0.1, 0.5, 1, 10]})

# Imprimiendo la mejor parametrización encontrada
print('La mejor parametrización es ' + str(clf.best_params_))

# Imprimiendo la puntuación alcanzada con la mejor parametrización
print('La puntuación alcanzada es ' + str(clf.best_score_))

print('Revisando el resto de las puntuaciones \n')

import matplotlib.pyplot as plt

# Imprimiendo las puntuaciones medias de prueba para cada valor de 'C' probado
print(clf.cv_results_['mean_test_score'])

# Graficando las puntuaciones medias de prueba
plt.plot(clf.cv_results_['mean_test_score'], 'r', marker='o')

# Configurando el eje x para mostrar los valores de 'C'
ax = plt.gca()
```

```
ax.set_xticks(range(len(clf.cv_results_['mean_test_score'])))
ax.set_xticklabels([0.01, 0.05, 0.1, 0.5, 1, 10])

# Añadiendo etiquetas y título a la gráfica
plt.xlabel('Valor de C')
plt.ylabel('Puntuación media de prueba')
plt.title('Puntuaciones medias de prueba para diferentes valores de
C')

# Mostrando la gráfica
plt.show()

La mejor parametrización es {'C': 0.05}
La puntuación alcanzada es 0.7211660295004282
Revisando el resto de las puntuaciones

[0.70098283 0.72116603 0.71802838 0.69233308 0.68050286 0.65429854]
```



# Entendiendo la Mejor Parametrización y el Puntaje Alcanzado en GridSearchCV

Al realizar una búsqueda en cuadrícula en aprendizaje automático usando `GridSearchCV`, el objetivo es encontrar los mejores parámetros para un modelo que resulten en el puntaje de rendimiento más alto. Aquí explicamos qué significan estos términos:

## Mejor Parametrización

- **Mejor Parametrización**: Esto se refiere al conjunto de parámetros que dieron los mejores resultados después de que se completó la búsqueda en cuadrícula. En el contexto de una SVM con un parámetro `C`, indica el valor de `C` que condujo al mejor rendimiento del modelo. Por ejemplo, si `GridSearchCV` devuelve `{'C': 1}`, significa que usar `C=1` para la SVM resultó en las predicciones más precisas durante el proceso de validación cruzada.

## Puntaje Alcanzado

- **Puntaje Alcanzado**: Después de encontrar la mejor parametrización, `GridSearchCV` también proporciona el mejor puntaje que se logró con este parámetro. Este puntaje es un número que refleja qué tan bien está funcionando el modelo con los mejores parámetros. El significado específico del puntaje depende del método de puntuación utilizado (por ejemplo, precisión, exactitud, recall, puntaje F1). Por ejemplo, si el puntaje es `0.60212765`, y estamos usando la precisión como nuestra métrica de puntuación, indica que el modelo con el mejor parámetro (`C=1`) pudo predecir correctamente las etiquetas de clase para aproximadamente el 60.21% del conjunto de datos validado cruzadamente.

## Poniéndolo Todo Junto

El proceso de usar `GridSearchCV` no solo ayuda a ajustar el modelo para encontrar los mejores ajustes, sino que también nos da una estimación de qué tan bien es probable que el modelo se desempeñe en datos no vistos. Es importante mirar tanto los mejores parámetros como el puntaje alcanzado para entender la efectividad potencial de tu modelo.

```
# Realizar predicciones con el modelo entrenado
y_hat = clf.predict(X_test)

from sklearn import metrics
import matplotlib.pyplot as plt

# Función para trazar la matriz de confusión
def plot_confusion_matrix(y_pred, y):
    # Generar la matriz de confusión
    cm = metrics.confusion_matrix(y, y_pred)
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.colorbar()
    plt.ylabel('Valor verdadero')
    plt.xlabel('Valor predicho')
```

```
    # Añadir etiquetas a los ejes
    classes = np.unique(y)
    plt.xticks(ticks=np.arange(len(classes)), labels=classes)
    plt.yticks(ticks=np.arange(len(classes)), labels=classes)

    # Añadir los valores a la matriz
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, cm[i, j], ha="center", va="center",
color="black")

    # Ajustar el tamaño de la figura
    fig = plt.gcf()
    fig.set_size_inches(9, 9)

# Imprimir la precisión de clasificación
print("Precisión de clasificación:", metrics.accuracy_score(y_test,
y_hat))

# Trazar la matriz de confusión
plot_confusion_matrix(y_hat, y_test)

# Imprimir el informe de clasificación
print("Informe de clasificación:")
print(metrics.classification_report(y_test, y_hat))
```

```
Precisión de clasificación: 0.7014765100671141
Informe de clasificación:
              precision    recall  f1-score   support

           1       0.63      0.74      0.68       113
           2       0.72      0.49      0.58       280
           3       0.76      0.84      0.80       439
           4       0.71      0.33      0.45        30
           5       0.73      0.71      0.72       184
           6       0.76      0.85      0.80       198
           7       0.58      0.73      0.65        56
           8       0.74      0.76      0.75        72
          10       0.65      0.65      0.65       124
          12       0.54      0.62      0.58       438
          13       0.76      0.60      0.68        43
          14       0.73      0.44      0.55       145
          15       0.57      0.43      0.49       329
          16       0.70      0.79      0.74      1340
          17       0.64      0.62      0.63       141
          18       0.53      0.40      0.45        25
          19       0.74      0.69      0.72      1494
          20       0.76      0.83      0.80       939
          21       0.48      0.34      0.40        70
```

| | | | | |
|---|---|---|---|---|
| 24 | 0.67 | 0.75 | 0.71 | 151 |
| 26 | 0.64 | 0.68 | 0.66 | 189 |
| 27 | 0.57 | 0.29 | 0.38 | 28 |
| 28 | 0.57 | 0.57 | 0.57 | 172 |
| 29 | 0.75 | 0.85 | 0.80 | 231 |
| 30 | 0.91 | 0.66 | 0.76 | 64 |
| 31 | 0.81 | 0.62 | 0.71 | 106 |
| 99 | 0.11 | 0.02 | 0.03 | 49 |
| accuracy | | | 0.70 | 7450 |
| macro avg | 0.66 | 0.60 | 0.62 | 7450 |
| weighted avg | 0.70 | 0.70 | 0.70 | 7450 |