Algoritmos Naive Bayes

Introducción: ¿Qué son los modelos Naive Bayes?

En un sentido amplio, los modelos de Naive Bayes son una clase especial de algoritmos de clasificación de Aprendizaje Automatico, o Machine Learning.

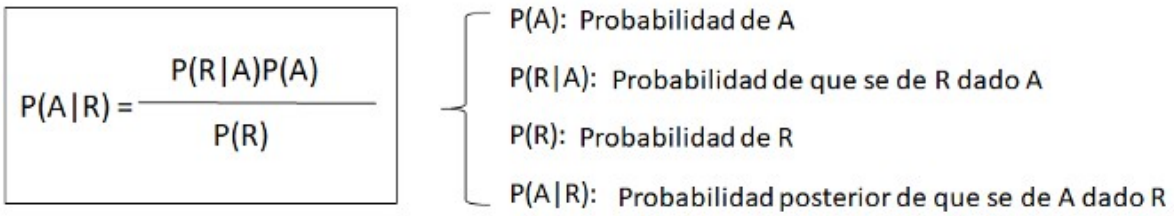
Se basan en una técnica de clasificación estadística llamada “teorema de Bayes”.

Estos modelos son llamados algoritmos “Naive”, o “Ingenuo” en español.

En ellos se asume que las variables predictoras son independientes entre sí. En otras palabras, que la presencia de una cierta característica en un conjunto de datos no está en absoluto relacionada con la presencia de cualquier otra característica.

Proporcionan una manera fácil de construir modelos con un comportamiento muy bueno debido a su simplicidad.

Lo consiguen proporcionando una forma de calcular la probabilidad ‘posterior’ de que ocurra un cierto evento A, dadas algunas probabilidades de eventos ‘anteriores’.



Ejemplo

Presentaremos los conceptos principales del algoritmo Naive Bayes estudiando un ejemplo.

Consideremos el caso de dos compañeros que trabajan en la misma oficina: Alicia y Bruno. Sabemos que:

Alicia viene a la oficina 3 días a la semana.

Bruno viene a la oficina 1 día a la semana.

Esta sería nuestra información “anterior”.

Estamos en la oficina y vemos pasar delante de nosotros a alguien muy rápido, tan rápido que no sabemos si es Alicia o Bruno.

Dada la información que tenemos hasta ahora y asumiendo que solo trabajan 4 días a la semana, las probabilidades de que la persona vista sea Alicia o Bruno, son:

Algoritmos Naive Bayes

P(Alicia) = 3/4 = 0.75

P(Bruno) = 1/4 = 0.25

Cuando vimos a la persona pasar, vimos que él o ella llevaba una chaqueta roja. También sabemos lo siguiente:

Alicia viste de rojo 2 veces a la semana.

Bruno viste de rojo 3 veces a la semana.

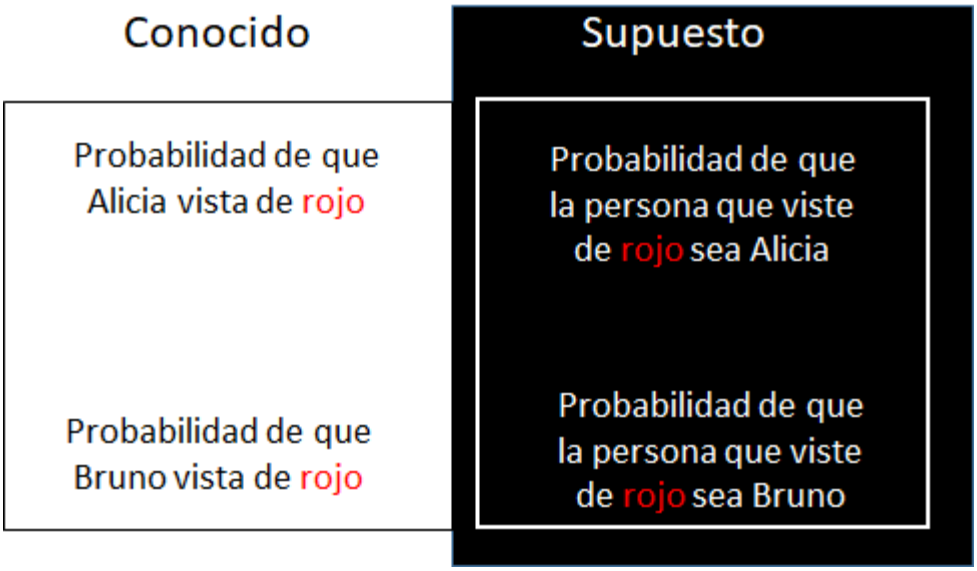
Así que, para cada semana de trabajo, que tiene cinco días, podemos inferir lo siguiente:

La probabilidad de que Alicia vista de rojo es → P(Rojo|Alicia) = 2/5 = 0.4

La probabilidad de que Bruno vista de rojo → P(Rojo|Bruno) = 3/5 = 0.6

Entonces, con esta información, ¿a quién vimos pasar? (en forma de probabilidad)

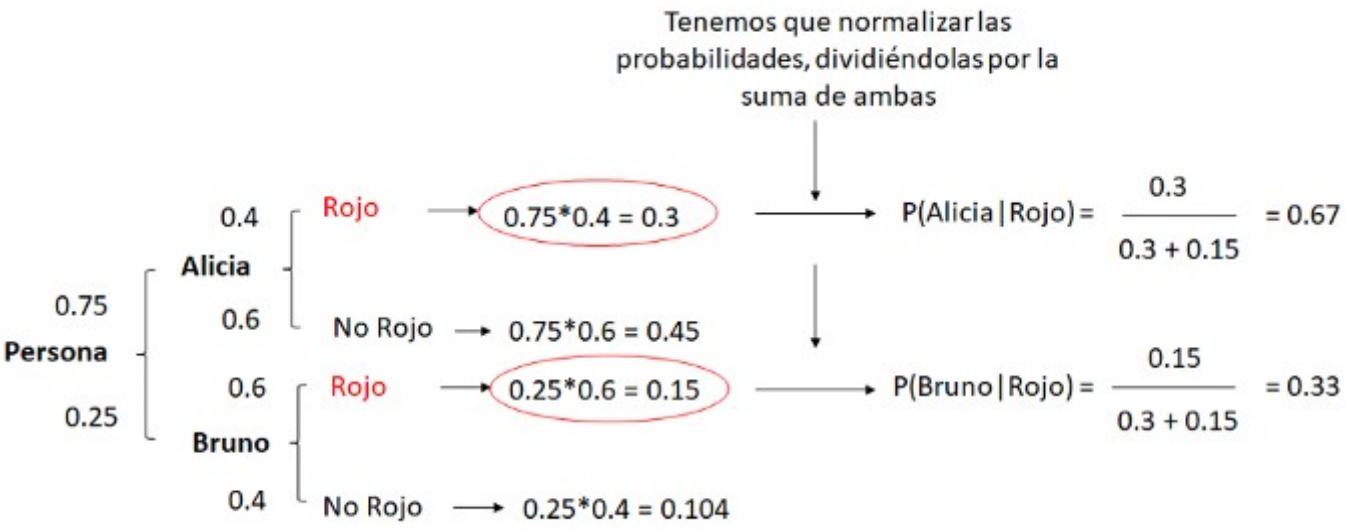
Esta nueva probabilidad será la información 'posterior'.



Inicialmente conocíamos las probabilidades P(Alicia) y P(Bruno), y después inferíamos las probabilidades de P(rojo|Alicia) y P(rojo|Bruno).

Algoritmos Naive Bayes

De forma que las probabilidades reales son:



Formalmente, el gráfico previo sería:

Algoritmos Naive Bayes

Algoritmo Naive Bayes Supervisado

A continuación se listan los pasos que hay que realizar para poder utilizar el algoritmo Naive Bayes en problemas de clasificación como el mostrado en el apartado anterior.

1. **Convertir** el conjunto de datos en una **tabla de frecuencias**.
2. Crear una **tabla de probabilidad** calculando las correspondientes a que ocurran los diversos eventos.
3. La ecuación **Naive Bayes se usa para calcular la probabilidad posterior de cada clase.**
4. La **clase con la probabilidad posterior más alta es el resultado de la predicción**.

Puntos fuertes y débiles de Naive Bayes

Los puntos fuertes principales son:

- Un manera **fácil y rápida de predecir clases, para problemas de clasificación binarios y multiclase.**
- En los casos en que sea apropiada una **presunción de independencia,** el algoritmo se comporta **mejor que otros modelos de clasificación**, incluso con menos datos de entrenamiento.
- El desacoplamiento de las distribuciones de características condicionales de clase significan que cada distribución puede ser estimada independientemente como si tuviera una sola dimensión. Esto ayuda con problemas derivados de la dimensionalidad y mejora el rendimiento.

Los puntos débiles principales son:

- Aunque son unos clasificadores bastante buenos, los algoritmos Naive Bayes son conocidos por ser **pobres estimadores.** Por ello, no se deben tomar muy en serio las probabilidades que se obtienen.
- La presunción de independencia Naive muy probablemente **no reflejará cómo son los datos en el mundo real.**
- Cuando el conjunto de datos de **prueba tiene una característica que no ha sido observada en el conjunto de entrenamiento**, el modelo le **asignará una probabilidad de cero** y será **inútil realizar predicciones**. Uno de los principales métodos para evitar esto, es la técnica de suavizado, siendo la estimación de Laplace una de las más populares.

Algoritmos Naive Bayes

Proyecto de Implementación: Detector de Spam

Actualmente, una de las aplicaciones principales de Machine Learning es la detección de spam. Casi todos los servicios de email más importantes proporcionan un detector de spam que clasifica el spam automáticamente y lo envía al buzón de “correo no deseado”.

En este proyecto, desarrollaremos un modelo Naive Bayes que clasifica los mensajes SMS como spam o no spam (‘ham’ en el proyecto). Se basará en datos de entrenamiento que le proporcionaremos.

Haciendo una investigación previa, encontramos que, normalmente, en los mensajes de spam se cumple lo siguiente:

- Contienen palabras como: ‘gratis’, ‘gana’, ‘ganador’, ‘dinero’ y ‘premio’.
- Tienden a contener palabras escritas con todas las letras mayúsculas y tienden al uso de muchos signos de exclamación.

Esto es un problema de clasificación binaria supervisada, ya que los mensajes son o ‘Spam’ o ‘No spam’ y alimentaremos un conjunto de datos etiquetado para entrenar el modelo.

Visión general

Realizaremos los siguientes pasos:

- Entender el conjunto de datos
- Procesar los datos
- Introducción al “Bag of Words” (BoW) y la implementación en la libreria Sci-kit Learn
- División del conjunto de datos (Dataset) en los grupos de entrenamiento y pruebas
- Aplicar “Bag of Words” (BoW) para procesar nuestro conjunto de datos
- Implementación de Naive Bayes con Sci-kit Learn
- Evaluación del modelo
- Conclusión

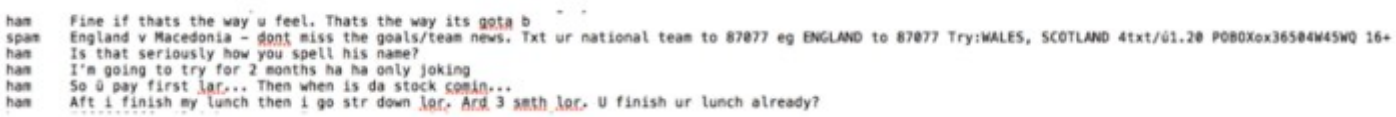
Algoritmos Naive Bayes

Entender el Conjunto de Datos

Utilizaremos un conjunto de datos del repositorio UCI Machine Learning.

Fichero: SMSSpamCollection

Un primer vistazo a los datos:



Las columnas no se han nombrado, pero como podemos imaginar al leerlas:

La primera columna determina la clase del mensaje, o ‘spam’ o ‘ham’ (no spam).

La segunda columna corresponde al contenido del mensaje

Primero importaremos el conjunto de datos y cambiaremos los nombre de las columnas. Haciendo una exploración previa, también vemos que el conjunto de datos está separado. El separador es ‘\t’.

```
# Importar la libreria Pandas
import pandas as pd# Dataset de
https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection


df = pd.read_table('./SMSSpamCollection',
                   sep='\t',
                   names=['label','sms_message'])# Visualización de las 5 primeras filas
df.head()
```

	label	sms_message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Algoritmos Naive Bayes

Preprocesamiento de Datos

Ahora, ya que el Sci-kit learn solo maneja valores numéricos como entradas, convertiremos las etiquetas en variables binarias, 0 representará 'ham' y 1 representará 'spam'.

Para representar la conversión:

```
# Conversion
df['label'] = df.label.map({'ham':0, 'spam':1})# Visualizar las dimensiones de los datos
df.shape
```

```
(5572, 2)
```

Introducción a la Implementación "Bag of Words" (BoW) y Sci-kit Learn

Nuestro conjunto de datos es una gran colección de datos en forma de texto (5572 filas). Como nuestro modelos solo aceptará datos numéricos como entrada, deberíamos procesar mensajes de texto. Aquí es donde "Bag of Words" entra en juego.

"Bag of Words" es un término usado para especificar los problemas que tiene una colección de datos de texto que necesita ser procesada. La idea es tomar un fragmento de texto y contar la frecuencia de las palabras en el mismo.

BoW trata cada palabra independientemente y el orden es irrelevante.

Podemos convertir un conjunto de documentos en una matriz, siendo cada documento una fila y cada palabra (token) una columna, y los valores correspondientes (fila, columna) son la frecuencia de ocurrencia de cada palabra (token) en el documento.

Como ejemplo, si tenemos los siguientes cuatro documentos:

```
['Hello, how are you!', 'Win money, win from home.', 'Call me now', 'Hello, Call you tomorrow?']
```

Convertiremos el texto a una matriz de frecuencia de distribución como la siguiente:

	are	call	from	hello	home	how	me	money	now	tomorrow	win	you
0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	0	1	0	1	0	0	1	0	0	2	0
2	0	1	0	0	0	0	1	0	1	0	0	0
3	0	1	0	1	0	0	0	0	0	1	0	1

Algoritmos Naive Bayes

Los documentos se numeran en filas, y cada palabra es un nombre de columna, siendo el valor correspondiente la frecuencia de la palabra en el documento.

Usaremos el método contador de vectorización de Sci-kit Learn, que funciona de la siguiente manera:

- Fragmenta y valora la cadena (separa la cadena en palabras individuales) y asigna un ID entero a cada fragmento (palabra).
- Cuenta la ocurrencia de cada uno de los fragmentos (palabras) valorados.
- Automáticamente convierte todas las palabras valoradas en minúsculas para no tratar de forma diferente palabras como “el” y “El”.
- También ignora los signos de puntuación para no tratar de forma distinta palabras seguidas de un signo de puntuación de aquellas que no lo poseen (por ejemplo “¡hola!” y “hola”).
- El tercer parámetro a tener en cuenta es el parámetro stop_words. Este parámetro se refiere a las palabra más comúnmente usadas en el lenguaje. Incluye palabras como “el”, “uno”, “y”, “soy”, etc. Estableciendo el valor de este parámetro por ejemplo en english, “CountVectorizer” automáticamente ignorará todas las palabras (de nuestro texto de entrada) que se encuentran en la lista de “stop words” de idioma inglés..

La implementación en Sci-kit Learn sería la siguiente:

```
# Definir los documentos
documents = ['Hello, how are you!',
             'Win money, win from home.',
             'Call me now.',
             'Hello, Call hello you tomorrow?']



# Importar el contador de vectorizacion e inicializarlo
from sklearn.feature_extraction.text import CountVectorizer
count_vector = CountVectorizer()



# Visualizar del objeto'count_vector' que es una instancia de 'CountVectorizer()'
print(count_vector)
```

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

Algoritmos Naive Bayes

Para ajustar el conjunto de datos del documento al objeto “CountVectorizer” creado, usaremos el método “fit()”, y conseguiremos la lista de palabras que han sido clasificadas como características usando el método “get_feature_names()”. Este método devuelve nuestros nombres de características para este conjunto de datos, que es el conjunto de palabras que componen nuestro vocabulario para “documentos”.

```
count_vector.fit(documents)
names = count_vector.get_feature_names()
names
```

```
['are',
 'call',
 'from',
 'hello',
 'home',
 'how',
 'me',
 'money',
 'now',
 'tomorrow',
 'win',
 'you']
```

A continuación, queremos crear una matriz cuyas filas serán una de cada cuatro documentos, y las columnas serán cada palabra. El valor correspondiente (fila, columna) será la frecuencia de ocurrencia de esa palabra (en la columna) en un documento particular (en la fila).

Podemos hacer esto usando el método “transform()” y pasando como argumento en el conjunto de datos del documento. El método “transform()” devuelve una matriz de enteros, que se puede convertir en tabla de datos usando “toarray()”.

```
doc_array = count_vector.transform(documents).toarray()
doc_array
```

```
array([[1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 2, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
       [0, 1, 0, 2, 0, 0, 0, 0, 0, 1, 0, 1]])
```

Algoritmos Naive Bayes

Para hacerlo fácil de entender, nuestro paso siguiente es convertir esta tabla en una estructura de datos y nombrar las columnas adecuadamente.

```
frequency_matrix = pd.DataFrame(data=doc_array, columns=names)
frequency_matrix
```

	are	call	from	hello	home	how	me	money	now	tomorrow	win	you
0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	0	1	0	1	0	0	1	0	0	2	0
2	0	1	0	0	0	0	1	0	1	0	0	0
3	0	1	0	2	0	0	0	0	0	1	0	1

Con esto, hemos implementado con éxito un problema de “BoW” o Bag of Words para un conjunto de datos de documentos que hemos creado.

Un problema potencial que puede surgir al usar este método es el hecho de que si nuestro conjunto de datos de texto es extremadamente grande, habrá ciertos valores que son más comunes que otros simplemente debido a la estructura del propio idioma. Así, por ejemplo, palabras como ‘es’, ‘el’, ‘a’, pronombres, construcciones gramaticales, etc. podrían sesgar nuestra matriz y afectar nuestro análisis.

Para mitigar esto, usaremos el parámetro stop_words de la clase CountVectorizer y estableceremos su valor en inglés.

Dividiendo el Conjunto de Datos en Conjuntos de Entrenamiento y Pruebas

Buscamos dividir nuestros datos para que tengan la siguiente forma:

X_train son nuestros datos de entrenamiento para la columna 'sms_message'

y_train son nuestros datos de entrenamiento para la columna 'label'

X_test son nuestros datos de prueba para la columna 'sms_message'

y_test son nuestros datos de prueba para la columna 'label'. Muestra el número de filas que tenemos en nuestros datos de entrenamiento y pruebas

Algoritmos Naive Bayes

```
# Dividir los datos en conjunto de entrenamiento y de test
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df['sms_message'], df['label'],
random_state=1)

print('Number of rows in the total set: {}'.format(df.shape[0]))

print('Number of rows in the training set: {}'.format(X_train.shape[0]))

print('Number of rows in the test set: {}'.format(X_test.shape[0]))
```

```
Number of rows in the total set: 5572
Number of rows in the training set: 417
Number of rows in the test set: 1393
```

Aplicar BoW para Procesar Nuestros Datos de Pruebas

Ahora que hemos dividido los datos, el próximo objetivo es convertir nuestros datos al formato de la matriz buscada. Para realizar esto, utilizaremos CountVectorizer() como hicimos antes. tenemos que considerar dos casos:

- Primero, tenemos que ajustar nuestros datos de entrenamiento (X_train) en CountVectorizer() y devolver la matriz.
- Sgundo, tenemos que transformar nustros datos de pruebas (X_test) para devolver la matriz.

Hay que tener en cuenta que X_train son los datos de entrenamiento de nuestro modelo para la columna 'sms_message' en nuestro conjunto de datos.

X_test son nuestros datos de prueba para la columna 'sms_message', y son los datos que utilizaremos (después de transformarlos en una matriz) para realizar predicciones. Compararemos luego esas predicciones con y_test en un paso posterior.

El código para este segmento está dividido en 2 partes. Primero aprendemos un diccionario de vocabulario para los datos de entrenamiento y luego transformamos los datos en una matriz de documentos; segundo, para los datos de prueba, solo transformamos los datos en una matriz de documentos.

```
# Instantiate the CountVectorizer method
count_vector = CountVectorizer()

# Fit the training data and then return the matrix
training_data = count_vector.fit_transform(X_train)

# Transform testing data and return the matrix. Note we are not fitting the testing data
into the CountVectorizer()
testing_data = count_vector.transform(X_test)
```

Algoritmos Naive Bayes

Implementación Naive Bayes con Sci-Kit Learn

Usaremos la implementación Naive Bayes “multinomial”. Este clasificador particular es adecuado para la clasificación de características discretas (como en nuestro caso, contador de palabras para la clasificación de texto), y toma como entrada el contador completo de palabras.

Por otro lado el Naive Bayes gausiano es más adecuado para datos continuos ya que asume que los datos de entrada tienen una distribución de curva de Gauss (normal).

Importaremos el clasificador “MultinomialNB” y ajustaremos los datos de entrenamiento en el clasificador usando fit().

```
from sklearn.naive_bayes import MultinomialNB
naive_bayes = MultinomialNB()
naive_bayes.fit(training_data, y_train)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Ahora que nuestro algoritmo ha sido entrenado usando el conjunto de datos de entrenamiento, podemos hacer algunas predicciones en los datos de prueba almacenados en ‘testing_data’ usando predict().

```
predictions = naive_bayes.predict(testing_data)
```

Una vez realizadas las predicciones el conjunto de pruebas, necesitamos comprobar la exactitud de las mismas.

Evaluación del modelo

Hay varios mecanismos para hacerlo, primero hagamos una breve recapitulación de los criterios y de la matriz de confusión.

- La matriz de confusión es donde se recogen el conjunto de posibilidades entre la clase correcta de un evento, y su predicción.

Prediction \ Class	True	False
True	True Positive	False Positive
False	False Negative	True Negative

Algoritmos Naive Bayes

- Exactitud: mide cómo de a menudo el clasificador realiza la predicción correcta. Es el ratio de número de predicciones correctas contra el número total de predicciones (el número de puntos de datos de prueba).

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

- Precisión: nos dice la proporción de mensajes que clasificamos como spam. Es el ratio entre positivos “verdaderos” (palabras clasificadas como spam que son realmente spam) y todos los positivos (palabras clasificadas como spam, lo sean realmente o no)

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Recall (sensibilidad): Nos dice la proporción de mensajes que realmente eran spam y que fueron clasificados por nosotros como spam. Es el ratio de positivos “verdaderos” (palabras clasificadas como spam, que son realmente spam) y todas las palabras que fueron realmente spam.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Para los problemas de clasificación que están sesgados en sus distribuciones de clasificación como en nuestro caso. Por ejemplo si tuviéramos 100 mensajes de texto y solo 2 fueron spam y los restantes 98 no lo fueron, la exactitud por si misma no es una buena métrica. Podríamos clasificar 90 mensajes como no spam (incluyendo los 2 que eran spam y los clasificamos como “no spam”, y por tanto falsos negativos) y 10 como spam (los 10 falsos positivos) y todavía conseguir una puntuación de exactitud razonablemente buena.

Para casos como este, la precisión y el recuerdo son bastante adecuados. Estas dos métricas pueden ser combinadas para conseguir la puntuación F1, que es el “peso” medio de las puntuaciones de precisión y recuerdo. Esta puntuación puede ir en el rango de 0 a 1, siendo 1 la mejor puntuación posible F1.

Usaremos las cuatro métricas para estar seguros de que nuestro modelo se comporta correctamente. Para todas estas métricas cuyo rango es de 0 a 1, tener una puntuación lo más cercana posible a 1 es un buen indicador de cómo de bien se está comportando el modelo.

Algoritmos Naive Bayes

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('Accuracy score: ', format(accuracy_score(y_test, predictions)))
print('Precision score: ', format(precision_score(y_test, predictions)))
print('Recall score: ', format(recall_score(y_test, predictions)))
print('F1 score: ', format(f1_score(y_test, predictions)))
```

```
Accuracy score:  0.9885139985642498
Precision score:  0.9720670391061452
Recall score:  0.9405405405405406
F1 score:  0.9560439560439562
```

Conclusión

- Una de las mayores ventajas que Naive Bayes tiene sobre otros algoritmos de clasificación es la **capacidad de manejo de un número extremadamente grande** de características. En nuestro caso, cada palabra es tratada como una característica y hay miles de palabras diferentes.
- También, **se comporta bien** incluso ante la presencia de c**aracterísticas irrelevantes** y no es relativamente afectado por ellos.
- La otra ventaja principal es su relativa simplicidad. Naive Bayes **funciona bien desde el principio** y ajustar sus parámetros es raramente necesario.
- **Raramente sobreajusta** los datos.
- Otra ventaja importante es que su modelo de **entrenamiento y procesos de predicción son muy rápidos** teniendo en cuenta la cantidad de datos que puede manejar.