Teoría detrás del teorema de Bayes

Si estudiaste Informática, Matemáticas o cualquier otro campo relacionado con la estadística, es muy probable que en algún momento te hayas topado con la siguiente fórmula:

P(H|E) = (P(E|H) \* P(H)) / P(E)

dónde

* P(H|E) es la probabilidad de hipótesis H dado el evento E, una probabilidad posterior.
* P(E|H) es la probabilidad de evento E dado que la hipótesis H es verdad.
* P(H) es la probabilidad de hipótesis H siendo cierto (independientemente de cualquier evento relacionado), o probabilidad previa de H.
* P(E) es la probabilidad de que ocurra el evento (independientemente de la hipótesis).

Este es el teorema de Bayes. A primera vista, puede ser difícil encontrarle sentido, pero es muy intuitivo si lo exploramos a través de un ejemplo:

Digamos que nos interesa saber si un correo electrónico que contiene la palabra sexo (evento) es spam (hipótesis).

Si volvemos a la descripción del teorema, este problema se puede formular como:

P(class=SPAM|contains="sex") = (P(contains="sex"|class=SPAM) \* P(class=SPAM)) / P(contains="sex")

que en un lenguaje sencillo es: La probabilidad de que un correo electrónico que contenga la palabra sexo sea spam es igual a la proporción de correos electrónicos SPAM que contienen la palabra sexo multiplicada por la proporción de correos electrónicos que son spam y dividida por la proporción de correos electrónicos correos electrónicos que contienen la palabra sexo.

P(class=SPAM|contains="sex") es la probabilidad de que un correo electrónico sea SPAM dado que este correo electrónico contiene la palabra sexo. Esto es lo que nos interesa predecir.

P(contains="sex"|class=SPAM) es la probabilidad de que un correo electrónico contenga la palabra sexo dado que este correo electrónico ha sido reconocido como SPAM. Estos son nuestros datos de entrenamiento, que representan la correlación entre un correo electrónico que se considera SPAM y el correo electrónico que contiene la palabra sexo.

P(class=SPAM) es la probabilidad de que un correo electrónico sea SPAM (sin conocimiento previo de las palabras que contiene). Esta es simplemente la proporción de correos electrónicos que son SPAM en todo nuestro conjunto de capacitación. Multiplicamos por este valor porque nos interesa saber qué tan significativa es la información relativa a los correos electrónicos no deseados. Si este valor es bajo, la importancia de cualquier evento relacionado con los correos electrónicos no deseados también será baja.

P(contains="sex") es la probabilidad de que un correo electrónico contenga la palabra sexo. Esta es simplemente la proporción de correos electrónicos que contienen la palabra sexo en todo nuestro conjunto de entrenamiento.

Dividimos por este valor porque cuanto más exclusiva es la palabra sexo, más importante es el contexto en el que aparece. Así, si este número es bajo (la palabra aparece muy raramente), puede ser un gran indicador de que en los casos en que sí aparece, es una característica relevante para analizar.

En resumen, el Teorema de Bayes nos permite realizar una deducción razonada de los eventos que ocurren en el mundo real en base al conocimiento previo de las observaciones que puedan implicarlo. Para aplicar este teorema a cualquier problema, necesitamos calcular los dos tipos de probabilidades que aparecen en la fórmula.

**Probabilidades de clase**

En el teorema, P(A) representa las probabilidades de cada evento. En el Clasificador Naive Bayes, podemos interpretar estas Probabilidades de Clase simplemente como la frecuencia de cada instancia del evento dividida por el número total de instancias. Por ejemplo, en el ejemplo anterior de detección de spam, P(class=SPAM) representa el número de correos electrónicos clasificados como spam dividido por la suma de todas las instancias (esto es spam + not spam)

P(class=SPAM) = count(class=SPAM) / (count(class=notSPAM) + count(class=SPAM))

Probabilidades condicionales

En el teorema, P(A|B) representa las probabilidades condicionales de un evento A dado otro evento B. En el clasificador Naive Bayes, estos codifican la probabilidad posterior de A ocurriendo cuando B es verdad.

Para el ejemplo del spam, P(class=SPAM|contains="sex") representa el número de casos en los que un correo electrónico se considera spam y contiene la palabra sexo, dividido por el número total de mensajes de correo electrónico que contienen la palabra sexo:

P(class=SPAM|contains="sex") = count(class=SPAM & contains=sex) / count(contains=sex)

**Aplicaciones**

La aplicación del Clasificador Naive Bayes se ha mostrado exitosa en diferentes escenarios. Un caso de uso clásico es la clasificación de documentos: determinar si un documento dado corresponde a determinadas categorías. Sin embargo, esta técnica tiene sus ventajas y limitaciones.

**Ventajas**

Naive Bayes es un algoritmo simple y fácil de implementar. Debido a esto, podría superar a modelos más complejos cuando la cantidad de datos es limitada.

Naive Bayes funciona bien con datos numéricos y categóricos. También se puede utilizar para realizar regresiones mediante Gaussian Naive Bayes.

**Limitaciones**

Dada la construcción del teorema, no funciona bien cuando falta cierta combinación de valores en los datos de entrenamiento. En otras palabras, si no tiene apariciones de una etiqueta de clase y un determinado valor de atributo juntos (por ejemplo, clase = «spam», contiene = «$$$») entonces la estimación de probabilidad basada en la frecuencia será cero. Dada la suposición de independencia condicional de Naive-Bayes, cuando se multiplican todas las probabilidades obtendrá cero.

Naive Bayes funciona bien siempre que las categorías se mantengan simples. Por ejemplo, funciona bien para problemas que involucran palabras clave como características (por ejemplo, detección de spam), pero no funciona cuando la relación entre palabras es importante (por ejemplo, análisis de sentimientos).

**Demo en Scikit-Learn**

¡Es hora de la demostración! Usaremos Python 3 junto con Scikit-Learn para construir un detector de SPAM muy simple para mensajes SMS (para aquellos de ustedes que son jóvenes, esto es lo que usamos para la mensajería en la Edad Media).

Puede buscar y descargar el conjunto de datos desde este enlace.

Necesitaremos tres bibliotecas que facilitarán mucho nuestra codificación: scikit-learn, pandas y nltk.

**Cargando los datos**

SMS Spam Collection v.1 es un conjunto de mensajes SMS etiquetados que se han recopilado para la investigación de SMS Spam. Contiene un conjunto de mensajes SMS en inglés de 5.574 mensajes, etiquetados según sea ham (legítimo) o spam. La distribución es de un total de 4.827 mensajes SMS legítimos (86,6%) y un total de 747 (13,4%) mensajes de spam.

Si abrimos el conjunto de datos, veremos que tiene el formato [label] [tab] [message], que se parece a esto:

ham Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...

ham Ok lar... Joking wif u oni...

spam Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's

ham U dun say so early hor... U c already then say...

Para cargar los datos, podemos usar el Dataframe de Pandas read\_table método.

Esto nos permite definir un separador (en este caso, una pestaña) y cambiar el nombre de las columnas en consecuencia:

import pandas as pd

df = pd.read\_table('SMSSpamCollection',

sep='t',

header=None,

names=['label', 'message'])

**Preprocesamiento**

Nos centraremos en eliminar las variaciones inútiles para nuestra tarea.

Primero, convertir las etiquetas de cadenas a valores binarios para nuestro clasificador:

df['label'] = df.label.map({'ham': 0, 'spam': 1})

En segundo lugar, convierta todos los caracteres del mensaje a minúsculas:

df['message'] = df.message.map(lambda x: x.lower())

En tercer lugar, elimine cualquier puntuación:

df['message'] = df.message.str.replace('[^ws]', '')

Cuarto, convierta los mensajes en una sola palabra usando nltk.

Primero, tenemos que importar y descargar el tokenizador desde la consola:

import nltk

nltk.download()

Aparecerá una ventana de instalación. Vaya a la pestaña «Modelos» y seleccione «punkt» en la columna «Identificador». Luego haga clic en «Descargar» e instalará los archivos necesarios. ¡Entonces debería funcionar! Ahora podemos aplicar la tokenización:

df['message'] = df['message'].apply(nltk.word\_tokenize)

Quinto, realizaremos algunos derivación de palabras. La idea de derivar es normalizar nuestro texto para que todas las variaciones de palabras tengan el mismo significado, independientemente del tiempo verbal. Uno de los algoritmos de derivación más populares es el Porter Stemmer:

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

df['message'] = df['message'].apply(lambda x: [stemmer.stem(y) for y in x])

Finalmente, transformaremos los datos en ocurrencias, que serán las características que alimentaremos a nuestro modelo:

from sklearn.feature\_extraction.text import CountVectorizer

# This converts the list of words into space-separated strings

df['message'] = df['message'].apply(lambda x: ' '.join(x))

count\_vect = CountVectorizer()

counts = count\_vect.fit\_transform(df['message'])

Podríamos dejarlo como el simple recuento de palabras por mensaje, pero es mejor usar Término Frecuencia Frecuencia inversa del documento, más conocido como tf-idf:

from sklearn.feature\_extraction.text import TfidfTransformer

transformer = TfidfTransformer().fit(counts)

counts = transformer.transform(counts)

Entrenando el modelo

====================

Comenzaremos dividiendo nuestros datos en conjuntos de prueba y entrenamiento:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(counts, df['label'], test\_size=0.1, random\_state=69)

Luego, todo lo que tenemos que hacer es inicializar el Clasificador Naive Bayes y ajustar los datos.

Para problemas de clasificación de texto, el clasificador multinomial Naive Bayes es adecuado:

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB().fit(X\_train, y\_train)

Evaluación del modelo

=====================

import numpy as np

predicted = model.predict(X\_test)

print(np.mean(predicted == y\_test))

El sencillo clasificador Naive Bayes tiene una precisión del 98,2%

Pero no es suficiente con solo proporcionar la precisión,

ya que nuestro conjunto de datos está desequilibrado en lo que respecta a las etiquetas

(86,6% legítimo en contraste con 13,4% de spam).

Puede suceder que nuestro clasificador sobrepase la clase legítima mientras ignora la clase de spam.

Para resolver esta incertidumbre, echemos un vistazo a la matriz de confusión:

from sklearn.metrics import confusion\_matrix

print(confusion\_matrix(y\_test, predicted))

[[478 4]

[ 6 70]]

La cantidad de errores está bastante equilibrada entre legítimos y spam,

con 4 mensajes legítimos clasificados como spam y 6 mensajes spam clasificados como legítimos.

En general, estos son muy buenos resultados para nuestro clasificador simple.

Conclusión

==========

En este artículo, hemos visto un curso intensivo sobre teoría y práctica del Clasificador Naive Bayes.

Hemos creado un clasificador Bayes ingenuo multimodal simple que logra una precisión del 98,2%

en la detección de spam para mensajes SMS.

import pandas as pd

import numpy as np

import nltk

from nltk.stem import PorterStemmer

df = pd.read\_table('SMSSpamCollection',

sep='t',

header=None,

names=['label', 'message'])

df

df['label'] = df.label.map({'ham': 0, 'spam': 1})

#df['message'] = df.message.map(lambda x: x.lower())

df['message'] = df.applymap(lambda s: s.lower() if type(s) == str else s)

df['message'] = df.message.str.replace('[^ws]', '')

df['message'] = df['message'].apply(nltk.word\_tokenize)

stemmer = PorterStemmer()

df['message'] = df['message'].apply(lambda x: [stemmer.stem(y) for y in x])

df['message'] = df['message'].apply(lambda x: ' '.join(x))

count\_vect = CountVectorizer()

counts = count\_vect.fit\_transform(df['message'])

from sklearn.feature\_extraction.text import TfidfTransformer

transformer = TfidfTransformer().fit(counts)

counts = transformer.transform(counts)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(counts, df['label'], test\_size=0.1, random\_state=69)

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB().fit(X\_train, y\_train)

predicted = model.predict(X\_test)

print(np.mean(predicted == y\_test))

from sklearn.metrics import confusion\_matrix

print(confusion\_matrix(y\_test, predicted))