Статистические методы в искусственном интеллекте. Предсказание. Наивный Байесовский алгоритм и реализация Байесовского выражения

- Представьте себе следующую ситуацию: вы работаете над задачей классификации, уже создали набор гипотез и сформировали признаки. Через час заказчики хотят увидеть первый вариант модели.
- Перед вами обучающий набор данных, содержащий несколько сотен тысяч элементов и большое количество признаков. Что вы будете делать? На вашем месте я бы воспользовался **наивным байесовским алгоритмом** (naive Bayes algorithm, НБА), который превосходит по скорости многие другие алгоритмы классификации. В его основе лежит теорема Байеса.

Содержание

- Что такое наивный байесовский алгоритм?
- Как он работает?
- Положительные и отрицательные.
- 4 приложения наивного байесовского алгоритма.
- Как создать базовую модель на его основе с помощью Python?
- Советы по оптимизации модели.

Что такое наивный байесовский алгоритм?

- Наивный байесовский алгоритм – это алгоритм классификации, основанный на теореме Байеса с допущением о независимости признаков. Другими словами, НБА предполагает, что наличие какого-либо признака в классе не связано с наличием какого-либо другого признака. Например, фрукт может считаться яблоком, если он красный, круглый и его диаметр составляет порядка 8 сантиметров. Даже если эти признаки зависят друг от друга или от других признаков, в любом случае они вносят независимый вклад в вероятность того, что этот фрукт является яблоком. В связи с таким допущением алгоритм называется «наивным».
- Модели на основе НБА достаточно просты и крайне полезны при работе с очень большими наборами данных. При своей простоте НБА способен превзойти даже некоторые сложные алгоритмы классификации.

- Теорема Байеса позволяет рассчитать апостериорную вероятность *P(c|x)* на основе *P(c)*, *P(x)* и *P(x|c)*.

Likelihood

Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

Posterior Probability

Predictor Prior Probability

$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

- *P(c|x)* – апостериорная вероятность данного класса *c* (т.е. данного значения целевой переменной) при данном значении признака *x*.
- *P(c)* – априорная вероятность данного класса.
- *P(x|c)* – правдоподобие, т.е. вероятность данного значения признака при данном классе.
- *P(x)* – априорная вероятность данного значения признака.

Как работает наивный байесовский алгоритм?

- Давайте рассмотрим пример. Ниже представлен обучающий набор данных, содержащий один признак «Погодные условия» (weather) и целевую переменную «Игра» (play), которая обозначает возможность проведения матча. На основе погодных условий мы должны определить, состоится ли матч. Чтобы сделать это, необходимо выполнить следующие шаги.
- Шаг 1. Преобразуем набор данных в частотную таблицу (frequency table).
- Шаг 2. Создадим таблицу правдоподобия (likelihood table), рассчитав соответствующие вероятности. Например, вероятность облачной погоды (overcast) составляет 0,29, а вероятность того, что матч состоится (yes) – 0,64.

Шаг 3. С помощью теоремы Байеса рассчитаем апостериорную вероятность для каждого класса при данных погодных условиях. Класс с наибольшей апостериорной вероятностью будет результатом прогноза.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

- *Sunny – Солнечная погода*
  Rainy – Дождливая погода
  Overcast – Облачная погода

Задача. Состоится ли матч при солнечной погоде (sunny)?

- Мы можем решить эту задачу с помощью описанного выше подхода.
- *P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P(Sunny)*
- Здесь мы имеем следующие значения:
- *P(Sunny | Yes) = 3 / 9 = 0,33*
- *P(Sunny) = 5 / 14 = 0,36*
- *P(Yes) = 9 / 14 = 0,64*
- Теперь рассчитаем *P(Yes | Sunny)*:
- *P(Yes | Sunny) = 0,33 * 0,64 / 0,36 = 0,60*
- Значит, при солнечной погоде более вероятно, что матч состоится.
- Аналогичным образом с помощью НБА можно прогнозировать несколько различных классов на основе множества признаков. Этот алгоритм в основном используется в области классификации текстов и при решении задач многоклассовой классификации.

Положительные и отрицательные стороны наивного байесовского алгоритма

Положительные стороны:

- Классификация, в том числе многоклассовая, выполняется легко и быстро.
- Когда допущение о независимости выполняется, НБА превосходит другие алгоритмы, такие как логистическая регрессия (logistic regression), и при этом требует меньший объем обучающих данных.
- НБА лучше работает с категорийными признаками, чем с непрерывными. Для непрерывных признаков предполагается нормальное распределение, что является достаточно сильным допущением.

Отрицательные стороны:

- Если в тестовом наборе данных присутствует некоторое значение категорийного признака, которое не встречалось в обучающем наборе данных, тогда модель присвоит нулевую вероятность этому значению и не сможет сделать прогноз. Это явление известно под названием «нулевая частота» (zero frequency). Данную проблему можно решить с помощью сглаживания. Одним из самых простых методов является сглаживание по Лапласу (Laplace smoothing).
- Хотя НБА является хорошим классификатором, значения спрогнозированных вероятностей не всегда являются достаточно точными. Поэтому не следует слишком полагаться на результаты, возвращенные методом *predict_proba*.
- Еще одним ограничением НБА является допущение о независимости признаков. В реальности наборы полностью независимых признаков встречаются крайне редко.

4 приложения наивного байесовского алгоритма

- **Классификация в режиме реального времени.** НБА очень быстро обучается, поэтому его можно использовать для обработки данных в режиме реального времени.
- **Многоклассовая классификация.** НБА обеспечивает возможность многоклассовой классификации. Это позволяет прогнозировать вероятности для множества значений целевой переменной.
- **Классификация текстов, фильтрация спама, анализ тональности текста.** При решении задач, связанных с классификацией текстов, НБА превосходит многие другие алгоритмы. Благодаря этому, данный алгоритм находит широкое применение в области фильтрации спама (идентификация спама в электронных письмах) и анализа тональности текста (анализ социальных медиа, идентификация позитивных и негативных мнений клиентов).
- **Рекомендательные системы.** Наивный байесовский классификатор в сочетании с коллаборативной фильтрацией (collaborative filtering) позволяет реализовать рекомендательную систему. В рамках такой системы с помощью методов машинного обучения и интеллектуального анализа данных новая для пользователя информация отфильтровывается на основании спрогнозированного мнения этого пользователя о ней.

Как создать базовую модель на основе наивного байесовского алгоритма с помощью Python?

- В этом нам поможет библиотека scikit-learn. Данная библиотека содержит три типа моделей на основе наивного байесовского алгоритма:
- **Gaussian** (нормальное распределение). Модель данного типа используется в случае непрерывных признаков и предполагает, что значения признаков имеют нормальное распределение.
- **Multinomial** (мультиномиальное распределение). Используется в случае дискретных признаков. Например, в задаче классификации текстов признаки могут показывать, сколько раз каждое слово встречается в данном тексте.
- **Bernoulli** (распределение Бернулли). Используется в случае двоичных дискретных признаков (могут принимать только два значения: 0 и 1). Например, в задаче классификации текстов с применением подхода «мешок слов» (bag of words) бинарный признак определяет присутствие (1) или отсутствие (0) данного слова в тексте.
- В зависимости от набора данных вы можете выбрать подходящую модель из описанных выше. Далее представлен пример кода для модели Gaussian.

Пример кода на Python

```
#Import Library of Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB
import numpy as np

#assigning predictor and target variables
x= np.array([[-3,7],[1,5], [1,2], [-2,0], [2,3], [-4,0], [-1,1], [1,1], [-2,2], [2,7], [-4,1], [-2,7]])
Y = np.array([3, 3, 3, 3, 4, 3, 3, 4, 3, 4, 4, 4])
```

```
#Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(x, Y)

#Predict Output
predicted= model.predict([[1,2],[3,4]])
print predicted

Output: ([3,4])
```

Пример:

- Байесовский классификатор относится к разряду машинного обучения. Суть такова: система, перед которой стоит задача определить, является ли следующее письмо спамом, заранее обучена каким-то количеством писем точно известных где «спам», а где «не спам». Уже стало понятно, что это обучение с учителем, где в роли учителя выступаем мы. Байесовский классификатор представляет документ (в нашем случае письмо) в виде набора слов, которые якобы не зависят друг от друга (вот от сюда и вытекает та самая наивность).

- Необходимо рассчитать оценку для каждого класса (спам/не спам) и выбрать ту, которая получилась максимальной. Для этого используем следующую формулу:

$$arg\max[P(Q_k)\prod_{i=1}^{n}P(x_i|Q_k)]$$

$$P(Q_k) = \frac{\text{число документов класса } Q_k}{\text{общее количество документов}}$$

$P(x_i|Q_k) = \dfrac{\alpha + N_{ik}}{\alpha M + N_k}$ — вхождение слова x_i в документ класса Q_k (со сглаживанием)*

N_k — количество слов входящих в документ класса Q_k

M — количество слов из обучающей выборки

N_{ik} — количество вхождений слова x_i в документ класса Q_k

α — параметр для сглаживания

- Когда объем текста очень большой, приходится работать с очень маленькими числами. Для того чтобы этого избежать, можно преобразовать формулу по свойству логарифма**:

$$\log ab = \log a + \log b$$

- Подставляем и получаем:

$$arg\max[\log P(Q_k) + \sum_{i=1}^{n} \log P(x_i|Q_k)]$$

*Во время выполнения подсчетов вам может встретиться слово, которого не было на этапе обучения системы. Это может привести к тому, что оценка будет равна нулю и документ нельзя будет отнести ни в одну из категорий (спам/не спам). Как бы вы не хотели, вы не обучите свою систему всем возможным словам. Для этого необходимо применить сглаживание, а точнее – сделать небольшие поправки во все вероятности вхождения слов в документ. Выбирается параметр $0<\alpha\leq 1$ (если $\alpha=1$, то это сглаживание Лапласа)

**Логарифм – монотонно возрастающая функция. Как видно из первой формулы – мы ищем максимум. Логарифм от функции достигнет максимума в той же точке (по оси абсцисс), что и сама функция. Это упрощает вычисление, ибо меняется только численное значение.

От теории к практике

Пусть наша система обучалась на следующих письмам, заранее известных где «спам», а где «не спам» (обучающая выборка):

Спам:

«Путевки по низкой цене»

«Акция! Купи шоколадку и получи телефон в подарок»

Не спам:

«Завтра состоится собрание»

«Купи килограмм яблок и шоколадку»

Задание: определить, к какой категории отнести следующее письмо:

«В магазине гора яблок. Купи семь килограмм и шоколадку»

Решение:

Составляем таблицу. Убираем все «стоп-слова», рассчитываем вероятности, параметр для сглаживания принимаем за единицу.

	Слово	Кол-во вхождений в «Спам»	Кол-во вхождений в «Не спам»	$P(x_i \mid \text{Спам})$	$P(x_i \mid \text{Не спам})$
Слова из обучающей выборки	Путевки	1	0		
	Низкой	1	0		
	Цене	1	0		
	Акция	1	0		
	Купи	1	1	$\frac{1+1}{14+9}$	$\frac{1+1}{14+7}$
	Шоколадку	1	1	$\frac{1+1}{14+9}$	$\frac{1+1}{14+7}$
	Получи	1	0		
	Телефон	1	0		
	Подарок	1	0		
	Завтра	0	1		
	Состоится	0	1		
	Собрание	0	1		
	Килограмм	0	1	$\frac{1+0}{14+9}$	$\frac{1+1}{14+7}$
	Яблок	0	1	$\frac{1+0}{14+9}$	$\frac{1+1}{14+7}$
	Магазине	0	0	$\frac{1+0}{14+9}$	$\frac{1+0}{14+7}$
	Гора	0	0	$\frac{1+0}{14+9}$	$\frac{1+0}{14+7}$
	Семь	0	0	$\frac{1+0}{14+9}$	$\frac{1+0}{14+7}$

- Оценка для категории «Спам»:

$$\frac{2}{4} \cdot \frac{2}{23} \cdot \frac{2}{23} \cdot \frac{1}{23} \cdot \frac{1}{23} \cdot \frac{1}{23} \cdot \frac{1}{23} \cdot \frac{1}{23} \approx 0,000000000587 (\text{или } 5{,}87\text{E-}10)$$

- Оценка для категории «Не спам»:

$$\frac{2}{4} \cdot \frac{2}{21} \cdot \frac{2}{21} \cdot \frac{2}{21} \cdot \frac{2}{21} \cdot \frac{1}{21} \cdot \frac{1}{21} \cdot \frac{1}{21} \approx 0,00000000444 (\text{или } 4{,}44\text{E-}9)$$

- **Ответ:** оценка «Не спам» больше оценки «Спам». Значит проверочное письмо — не спам!

- То же самое рассчитаем и с помощью функции, преобразованной по свойству логарифма:
  Оценка для категории «Спам»:

$$\log\frac{2}{4}+\log\frac{2}{23}+\log\frac{2}{23}+\log\frac{1}{23}+\log\frac{1}{23}+\log\frac{1}{23}+\log\frac{1}{23}+\log\frac{1}{23}\approx -21,25$$

- Оценка для категории «Не спам»:

$$\log\frac{2}{4}+\log\frac{2}{21}+\log\frac{2}{21}+\log\frac{2}{21}+\log\frac{2}{21}+\log\frac{1}{21}+\log\frac{1}{21}+\log\frac{1}{21}\approx -19,23$$

- **Ответ:** аналогично предыдущему ответу. Проверочное письмо – не спам!

Реализация на языке программирования R

```
library("tm")          #Библиотека для stopwords
library("stringr")     #Библиотека для работы со строками

#Обучаюшая выборка со спам письмами:
spam <- c(
  'Путевки по низкой цене',
  'Акция! Купи шоколадку и получи телефон в подарок'
)

#Обучающая выборка с не спам письмами:
not_spam <- c(
  'Завтра состоится собрание',
  'Купи килограмм яблок и шоколадку'
)

#Письмо требующее проверки
test_letter <- "В магазине гора яблок. Купи семь килограмм и шоколадку"
```

```r
#----------------Для спама--------------------
#Убираем все знаки препинания
spam <- str_replace_all(spam, "[[:punct:]]", "")

#Делаем все маленьким регистром
spam <- tolower(spam)

#Разбиваем слова по пробелу
spam_words <- unlist(strsplit(spam, " "))

#Убираем слова, которые совпадают со словами из stopwords
spam_words  <- spam_words[! spam_words %in% stopwords("ru")]

#Создаем таблицу с уникальными словами и их количеством
unique_words <- table(spam_words)

#Создаем data frame
main_table <- data.frame(u_words=unique_words)

#Переименовываем столбцы
names(main_table) <- c("Слова","Спам")
```

```r
#----------------Для не спама------------------
not_spam <- str_replace_all(not_spam, "[[:punct:]]", "")
not_spam <- tolower(not_spam)
not_spam_words <- unlist(strsplit(not_spam, " "))
not_spam_words  <- not_spam_words[! not_spam_words %in% stopwords("ru")]

#----------------Для проверки------------------
test_letter <- str_replace_all(test_letter, "[[:punct:]]", "")
test_letter <- tolower(test_letter)
test_letter <- unlist(strsplit(test_letter, " "))
test_letter <- test_letter[! test_letter %in% stopwords("ru")]
#------------------------------------------
```

```r
#Создаем новый столбик для подсчета не спам писем
main_table$Не_спам <- 0

for(i in 1:length(not_spam_words)){
#Создаем логическую переменную
  need_word <- TRUE
  for(j in 1:(nrow(main_table))){
#Если "не спам" слово существует, то к счетчику уникальных слов +1
    if(not_spam_words[i]==main_table[j,1])
    {
      main_table$Не_спам[j] <- main_table$Не_спам[j]+1
      need_word <- FALSE
    }
  }
#Если слово не встречалось еще, то добавляем его в конец data frame и создаем счетчики
  if(need_word==TRUE)
  {
    main_table <- rbind(main_table,data.frame(Слова=not_spam_words[i],Спам=0,Не_спам=1))
  }
}
```

```r
#-------------
#Создаем столбик содержащий вероятности того, что выбранное слово - спам
main_table$Вероятность_спам <- NA

#Создаем столбик содержащий вероятности того, что выбранное слово - не спам
main_table$Вероятность_не_спам <- NA

#-------------
#Создаем функцию подсчета вероятности вхождения слова Xi в документ класса Qk
formula_1 <- function(N_ik,M,N_k)
{
  (1+N_ik)/(M+N_k)
}
#-------------

#Считаем количество слов из обучающей выборки
quantity <- nrow(main_table)

for(i in 1:length(test_letter))
{
```

```
#Используем ту же логическую переменную, чтобы не создавать новую
need_word <- TRUE
for(j in 1:nrow(main_table))
{
#Если слово из проверочного письма уже существует в нашей выборке то считаем вероятность каждой категории
if(test_letter[i]==main_table$Слова[j])
{
main_table$Вероятность_спам[j] <- formula_1(main_table$Спам[j],quantity,sum(main_table$Спам)) main_table$Вероятность_не_спам[j] <- formula_1(main_table$Не_спам [j],quantity,sum(main_table$Не_спам))
need_word <- FALSE } }
#Если слова нет, то добавляем его в конец data frame, и считаем вероятность спама/не спама
if(need_word==TRUE)
{
main_table <- rbind(main_table,data.frame(Слова=test_letter[i],Спам=0,Не_спам=0, Вероятность_спам=NA,Вероятность_не_спам=NA))
main_table$Вероятность_спам[nrow(main_table)] <- formula_1(main_table$Спам [nrow(main_table)],quantity,sum(main_table$Спам)) main_table$Вероятность_не_спам[nrow(main_table)] <- formula_1(main_table$Не_спам[nrow(main_table)],quantity,sum(main_table$Не_спам)) } }
```

```r
#Переменная для подсчета оценки класса "Спам"
probability_spam <- 1

#Переменная для подсчета оценки класса "Не спам"
probability_not_spam <- 1

for(i in 1:nrow(main_table))
{
  if(!is.na(main_table$Вероятность_спам[i]))
  {
#Шаг 1.1 Определяем оценку того, что письмо - спам
    probability_spam <- probability_spam * main_table$Вероятность_спам[i]
  }
  if(!is.na(main_table$Вероятность_не_спам[i]))
  {
#Шаг 1.2 Определяем оценку того, что письмо - не спам
    probability_not_spam <- probability_not_spam * main_table$Вероятность_не_спам[i]
  }
}

#Шаг 2.1 Определяем оценку того, что письмо - спам
probability_spam <- (length(spam)/(length(spam)+length(not_spam)))*probability_spam
```

```
#Шаг 2.2 Определяем оценку того, что письмо - не спам
probability_not_spam <- (length(not_spam)/(length(spam)+length(not_spam)))*probability_not_spam

#Чья оценка больше - тот и победил
ifelse(probability_spam>probability_not_spam,"Это сообщение - спам!","Это сообщение - не спам!")
```