

Naive Bayes

Ce ne așteaptă?

1. Teorema lui Bayes
2. Etapele algoritmului Naive Bayes
3. Extragerea caracteristicilor din text
4. Instrumente Scikit-Learn

1. Teorema lui Bayes

▪ Expresia matematică

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

unde:

- $P(A|B)$ - **probabilitatea evenimentului A când a avut loc evenimentul B**
- $P(B|A)$ - **probabilitatea evenimentului B când a avut loc evenimentul A**
- $P(A)$ - **probabilitatea evenimentului A**
- $P(B)$ - **probabilitatea evenimentului B**

- **Extinderea teoremei asupra setului de caracteristici**

$x = (x_1, \dots, x_n)$ - vectorul caracteristicilor

C_k - valoarea unei anumite clase de clasificare

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B)} \Rightarrow p(C_k|x) = \frac{p(C_k) * p(x|C_k)}{p(x)} \Rightarrow p(C_k|x) = \frac{p(C_k, x_1, \dots, x_n)}{p(x)}$$

unde:

$$p(C_k, x_1, \dots, x_n) = p(x_1|x_2, \dots, x_n, C_k) * \dots * p(x_{n-1}|x_n, C_k) * p(x_n|C_k) * p(C_k)$$

dacă se consideră o presupunere naivă (de aici și denumirea algoritmului) că caracteristicile x_i sunt independente atunci:

$$p(x_i|x_{i+1}, \dots, x_n, C_k)p(x_i|C_k) \Rightarrow p(C_k|x_1, \dots, x_n) \sim p(C_k) * \prod_{i=1}^n p(x_i|C_k)$$

unde: \sim semn al proporționalității

2. Etapele algoritmului Naive Bayes

■ Algoritmul clasificării recenziilor unui produs

1. Determinarea probabilităților fiecărei clase

Numărul total de recenzii pentru elaborarea algoritmului 35, dintre care 25 pozitive și 10 negative



$$p(poz) = \frac{25}{35} = 0,71$$



$$p(neg) = \frac{10}{35} = 0,29$$

2. Determinarea cuvintelor de bază în recenzii și numărul de apariție a acestora pentru fiecare clasă

Clasă\ cuvânt	produs	producător	bun	prost
pozitiv	10	7	8	1
negativ	8	10	0	5

3. Sumarea unei constante (const=1) la numărul apariției fiecărui cuvânt de bază pentru a evita prezența valorilor nule

Clasă\cuvânt	produs	producător	bun	prost
pozitiv	10+1=11	7+1=8	8+1=9	1+1=2
negativ	8+1=9	10+1=11	0+1=1	5+1=6

4. Se determina probabilitatea apariției fiecărui cuvânt per clasă

Pentru clasa pozitivă:

$$p(\text{produs}|\text{poz}) = \frac{11}{11 + 8 + 9 + 2} = \frac{11}{30} = 0,37 \quad p(\text{producător}|\text{poz}) = \frac{8}{30} = 0,26$$

$$p(\text{bun}|\text{poz}) = \frac{9}{30} = 0,3 \quad p(\text{prost}|\text{poz}) = \frac{2}{30} = 0,07$$

Pentru clasa negativă:

$$p(\text{produs}|\text{neg}) = \frac{9}{9 + 11 + 1 + 6} = \frac{9}{27} = 0,33 \quad p(\text{producător}|\text{neg}) = \frac{11}{27} = 0,41$$

$$p(\text{bun}|\text{neg}) = \frac{1}{27} = 0,04 \quad p(\text{prost}|\text{neg}) = \frac{6}{27} = 0,22$$

5. Aplicarea algoritmului asupra recenziilor noi

Fie recenzia formată din cuvintele: “producător bun”

$$p(\text{recenzie}_{poz}) \sim p(poz) * p(\text{producator}|poz) * p(bun|poz) = 0,71 * 0,26 * 0,3 = 0,055$$

$$p(\text{recenzie}_{neg}) \sim p(neg) * p(\text{producator}|neg) * p(bun|neg) = 0,29 * 0,41 * 0,04 = 0,005$$

Fie recenzia formată din cuvintele: “produs prost”

$$p(\text{recenzie}_{poz}) \sim p(poz) * p(\text{produs}|poz) * p(prost|poz) = 0,71 * 0,37 * 0,07 = 0,018$$

$$p(\text{recenzie}_{neg}) \sim p(neg) * p(\text{produs}|neg) * p(prost|neg) = 0,29 * 0,33 * 0,22 = 0,021$$

3. Extragerea caracteristicilor din text

- Algoritmii machine learning nu accepta datele textuale brute ci doar date numerice
- Extragerea caracteristicilor din text presupune procesarea și conversia datelor textuale în valori numerice
- Există două metode de bază de conversie a valorilor textuale în valori numerice:
 - Count Vectorization – formează un vector cu valori 0 și 1 în funcție de prezența sau lipsa cuvântului în text
 - TF-IDF (Term Frequency – Inverse Document Frequency) – formază un vector cu valori numerice proporționale importanței cuvântului

Count Vectorization

1. Se creează vocabularul format din cuvintele unice ale tuturor documentelor (textelor)

Tu ești bine
Eu mă simt bine
Eu sunt bine

Vocabular = ['Tu', 'ești', 'bine', 'Eu', 'mă', 'simt', 'sunt']

2. Se creează matricea termenilor documentelor (DTM - Document Term Matrix) ce include numărul de apariție a cuvintelor vocabularului în fiecare document

	Tu	ești	bine	Eu	mă	simt	sunt
Tu ești bine	1	1	1	0	0	0	0
Eu mă simt bine	0	0	1	1	1	1	0
Eu sunt bine	0	0	1	1	0	0	1

▪ Neajunsuri Count Vectorization

- Pentru vocabulare de dimensiuni foarte mari DTM va conține foarte multe valori de zero
 - Cuvinte comune (un, o, mai, cu, la, etc) ce nu conțin informație referitoare la clasa de predicție vor avea o pondere considerabilă
 - În funcție de domeniul textului vor exista cuvinte specifice (de exemplu, în sport – cuvântul alergare) ce de asemenea nu conțin informație referitoare la clasa de predicție vor avea o pondere considerabilă
-
- Cuvintele comune pot fi excluse din vocabular prin utilizarea proprietății `stop_words` ce va include lista acestor cuvinte

- **Term Frequency – Inverse Document Frequency (TM-IDF)**

1. Se determină frecvența termenilor (Term Frequency) $tf(t, d)$ – numărul de apariții ale cuvântului t în textul d

2. Se determină frecvența inversă a documentelor (Inverse Document Frequency) $idf(t, D)$ – logaritmul dintre raportul numărului totale de texte N la numărul D de texte unde apare cuvântul t

$$idf(t, D) = \log \frac{N}{D}$$

3. Se determină frecvența termenilor - frecvența inversă a documentelor (Term Frequency - Inverse Document Frequency) $tfidf(t, d, D)$ – ce reprezintă numărul de apariții ale cuvântului t în textul d ($tf(t, d)$) înmulțit cu un coeficient de ponderare care este funcție de numărul D de texte unde apare acest cuvânt t ($idf(t, D)$)

$$tfidf(t, d, D) = tf(t, d) * idf(t, D) = tf(t, d) * \log \frac{N}{D}$$

4. Instrumente Scikit-Learn

- **CountVectorizer** - crearea vocabularului și a matricei DTM - Document Term Matrix

- Se importă clasa convertorului **CountVectorizer**

```
from sklearn.feature_extraction.text import CountVectorizer
```

- Se creează convertorul ca o instanță a clasei cu specificare parametrului:

- **stop_words** – cuvintele ce se vor exclude din vocabular, (valori posibile = {'english'}, listă, implicit =None)

```
cv = CountVectorizer(stop_words=['o', 'este', 'altă'])
```

- Se aplică convertorul asupra textului și se formează DTM în format sparse

```
sparse_mat = cv.fit_transform(text)
```

- Se afișează amitricea în format dense

```
sparse_mat.todense()
```

- Se afișează amitricea în format dense

```
cv.vocabulary_
```

■ TfidfTransformer - converisia matricei DTM în matrice cu valori TF-IDF

- Se importă clasa convertorului TfidfTransformer

```
from sklearn.feature_extraction.text import TfidfTransformer
```
- Se creează convertorul ca o instanță a clasei

```
tfidf_transformer = TfidfTransformer()
```
- Se aplică convertorul asupra matricii sparse DTM formată cu CountVectorizer

```
tfidf_mat = tfidf_transformer.fit_transform(sparse_mat)
```
- Se afișează matricea cu TF-IDF în format dense

```
tfidf_mat.todense()
```

- **TfidfVectorizer** - crearea directă a matricei DTM cu valori TF-IDF
 - Se importă clasa convertorului TfidfVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
```
 - Se creează convertorul ca o instanță a clasei cu specificarea stop_words

```
tfidf = TfidfVectorizer(stop_words=['o', 'este', 'altă'])
```
 - Se aplică convertorul asupra textului

```
tfidf_mat = tfidf.fit_transform(text)
```
 - Se afișează matricea cu TF-IDF în format dense

```
tfidf_mat.todense()
```

■ MultinomialNB - elaborarea modelului Naive Bayes Multinomial

- Se importă clasa algoritmului Naive Bayes Multinomial
`from sklearn.naive_bayes import MultinomialNB`
- Se creează modelul ca o instanță a clasei
`model = MultinomialNB()`
- Se realizează triningul modelului de datele training transformate TF-IDF
`model.fit(X_train_tfidf, y_train)`
- Se realizează predicția de datele de test transformate TF-IDF
`y_pred = model.predict(X_test_tfidf)`