Clasificator Naiv Bayes Multinomial

251: Grigorașcu Andrei Antonio, Bîrsan Gheorghe-Daniel, Șerbănescu George Florin

November 12, 2024

Introducere

Acest proiect implementează un clasificator de text Naiv Bayes multinomial în Python, folosind Google Collab. Clasificatorul Naiv Bayes se bazează pe teorema lui Bayes și este utilizat frecvent pentru sarcini de clasificare a textului. În acest proiect, antrenăm modelul folosind un set de date cu text etichetat și îl folosim pentru a clasifica date noi.

Teorema lui Bayes

Algoritmul Naiv Bayes se bazează pe teorema lui Bayes, care descrie probabilitatea unei clase C dată prezența unor caracteristici F. Formula teoremei lui Bayes este:

$$P(C \mid F) = \frac{P(F \mid C) \cdot P(C)}{P(F)} \tag{1}$$

În cazul nostru, F reprezintă cuvintele dintr-un text, iar C este eticheta clasei (sau a categoriei).

Presupoziția Naiv Bayes

Naiv Bayes presupune că fiecare cuvânt din text este condiționat independent de fiecare alt cuvânt, dată clasa C. Aceasta simplifică calculele și ne permite să calculăm probabilitatea fiecărui cuvânt separat. Astfel, putem scrie:

$$P(C \mid F) \propto P(C) \prod_{i=1}^{n} P(f_i \mid C) \tag{2}$$

unde f_i reprezintă fiecare cuvânt din text, iar n este numărul total de cuvinte din text.

Implementare

1. **Tokenizare:** Împărțim fiecare text în cuvinte, eliminăm semnele de punctuație și transformăm totul în litere mici, apoi eliminăm conjucțiile, prepozițiile și alte cuvinte de legătură.

2. Calcularea probabilităților clasei: Calculăm probabilitățile $P(C)$ pentru fiecare clasă.

3. Calcularea probabilităților cuvintelor: Calculăm $P(f_i \mid C)$ pentru fiecare cuvânt din vocabular.

4. Clasificare: Pentru un text nou, folosim teorema lui Bayes pentru a calcula probabilitatea pentru fiecare clasă și alegem clasa cu cea mai mare probabilitate.

Utilizare

Pentru a folosi clasificatorul, importați codul și urmați exemplul de mai jos:

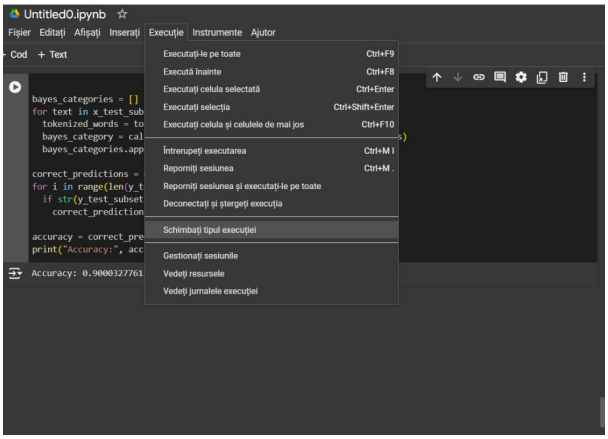

Figure 1: Menu

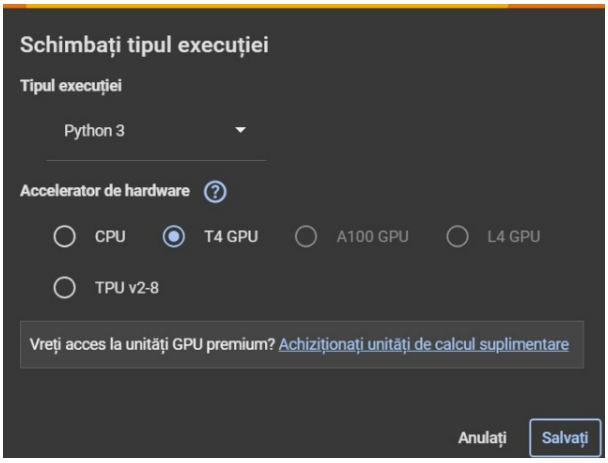

Figure 2: GPU

Evaluarea Performanței

Introducere

Acest document descrie implementarea unui clasificator Naiv Bayes Multinomial pentru clasificarea textelor în categorii, realizat de la zero în Python. Clasificatorul Naiv Bayes folosește teorema lui Bayes pentru a calcula probabilitatea ca un text aparține unei clase date, pe baza cuvintelor sale.

Codul Python Explicat

Se recomandă folosirea unui accelerator der hardware: T4 GPU.

Importarea bibliotecilor necesare

Pentru a începe, importăm bibliotecile esențiale pentru acest proiect: `pandas` pentru manipularea datelor, `os` și `re` pentru lucrul cu fișiere și procesarea textului, și `Counter` și `log` din `collections` și `math` pentru numărarea elementelor și calculul logaritmului:

```
import kagglehub
import pandas as pd
import os
import re
from collections import defaultdict
```

```
from math import log
from typing import List
from collections import Counter
```

Descărcarea și încărcarea setului de date

Folosește `kagglehub` pentru a descărca setul de date de la Kaggle și pentru a specifica calea fișierului:

```
path = kagglehub.dataset_download("akash14/news-category-dataset")
file_path1 = os.path.join(path, 'Data_Train.csv')

train_data = pd.read_csv(file_path1, encoding='latin-1')
```

`train_data` conține datele de antrenament, iar fiecare text din coloana `STORY` este asociat cu o etichetă de categorie în coloana `SECTION`.

Împărțirea datelor în seturi de antrenament și testare

Împărțim setul de antrenament într-un subset pentru antrenare (80%) și unul pentru testare (20%) pentru a valida modelul:

```
total_samples = len(x_train)
subset_size = int(0.8 * total_samples)
test_size = int(0.2 * total_samples)

x_train_subset = x_train[:subset_size]
y_train_subset = y_train[:subset_size]
x_test_subset = x_train[:subset_size]
y_test_subset = y_train[:subset_size]
```

Calculul probabilităților

Funcția `probabilities` calculează probabilitățile fiecărei clase:

```
def probabilities(y_train_subset) -> dict[str, float]:
    n = len(y_train_subset)
    result = {"0": 0, "1": 0, "2": 0, "3": 0}

    for row in y_train_subset:
      result[str(row)] += 1

    for key in result.keys():
        result[key] /= n

    return result

a_priori_probs = probabilities(y_train_subset)
```

Aceasta numără instanțele fiecărei etichete și apoi calculează probabilitățile prin împărțirea la numărul total de exemple.

Tokenizarea textului

Pentru a prelucra textul, folosim funcția `tokenize`, care transformă textul în litere mici, elimină caracterele speciale și elimină conjuncțiile, prepozițiile și cuvintele de legătură.

```
stop_words = {
    "the", "and"....
}
```

```
def tokenize(text):
    text = text.lower()
    text = re.sub(r'\W+', ' ', text)
    words = set(word for word in text.split() if word not in stop_words)
    return list(words)
```

Parsează datele în categorii de cuvinte

Funcția `parse_data` organizează textele în funcție de etichetă.

```
def parse_data(X_train_subset, y_train_subset):
    categories = {0: [], 1: [], 2: [], 3: []}
    for text, label in zip(x_train_subset, y_train_subset):
        tokenized_words = tokenize(text)
        categories[label].append(tokenized_words)
    return categories
```

Selectarea celor mai frecvente cuvinte

Funcția `top_k_frequent_words` selectează cele mai frecvente cuvinte din datele de antrenament.

```
def top_k_frequent_words(paragraphs: List[List[str]], k: int) -> List[tuple[str, int]]:
    word_count = Counter()
    for paragraph in paragraphs:
        words = set(paragraph)
        word_count.update(words)

    number=(k/100)*len(word_count)
    top_k_words = word_count.most_common(int(number))
    return top_k_words
```

Calculul probabilităților pentru cuvintele cheie

Funcția `find_probabilities_for_top_words` calculează probabilitatea fiecărui cuvânt pentru fiecare categorie, aplicând și o tehnică de regularizare cunoscută sub numele de **Laplace Smoothing**. Aceasta este necesară pentru a evita probabilitățile zero pentru cuvintele care nu apar într-o categorie specifică. Laplace Smoothing adaugă o valoare constantă (în acest caz, 1) la numărul de apariții al fiecărui cuvânt în fiecare categorie, precum și la numărul total de paragrafe, asigurând astfel că nicio probabilitate nu devine zero.

```
def find_probabilities_for_top_words(data: dict[int, List[List[str]]], top_words : List[tuple[str, i

    d_res = {}
    total_paragraphs = sum(len(paragraphs) for paragraphs in data.values())

    for word, count in top_words:
      # +1 Offset to avoid  0%
        d_res[word] = {"0": 1, "1": 1, "2": 1, "3": 1, "total": 1}
        for key in data.keys():
            for paragraph in data[int(key)]:
                if word in paragraph:
                    d_res[word][str(key)] += 1
                    d_res[word]["total"] += 1

    for word in d_res.keys():
        for key in d_res[word].keys():
            if key != "total":
                d_res[word][str(key)] /= len(data[int(key)])
```

```
            else:
                d_res[word][str(key)] /= total_paragraphs

    return d_res

d_res = find_probabilities_for_top_words(data, top_words)
d_res
```

Folosirea **Laplace Smoothing** ajută la prevenirea problemelor cauzate de lipsa unui cuvânt într-o categorie, menținând calculul probabilităților robust.

Funcția Bayes pentru clasificare

Funcția `calculate_bayes` aplică teorema lui Bayes pentru a clasifica un text nou.

```
def calculate_bayes(a_priori_probs: dict[str, float] , probabilities: dict[str, dict[str, float]], p
    p0, p1, p2, p3 = a_priori_probs["0"], a_priori_probs["1"], a_priori_probs["2"], a_priori_probs["
    for word in paragraph:
        if word not in probabilities.keys():
            continue
        p0_given_word = probabilities[word]["0"]
        p1_given_word = probabilities[word]["1"]
        p2_given_word = probabilities[word]["2"]
        p3_given_word = probabilities[word]["3"]
        p_word = probabilities[word]["total"]

        p0 = p0_given_word * p0 / p_word
        p1 = p1_given_word * p1 / p_word
        p2 = p2_given_word * p2 / p_word
        p3 = p3_given_word * p3 / p_word

    probs = [(p0, "0"), (p1, "1"), (p2, "2"), (p3, "3")]
    probs = sorted(probs, reverse=True)
    return probs[0][1]
```

Calculul acurateței

Acum rulăm clasificatorul pe datele de testare și calculăm acuratețea.

```
bayes_categories = []
for text in x_test_subset:
  tokenized_words = tokenize(text)
  bayes_category = calculate_bayes(a_priori_probs, d_res, tokenized_words)
  bayes_categories.append(bayes_category)

correct_predictions = 0
for i in range(len(y_test_subset)):
  if str(y_test_subset.iloc[i]) == bayes_categories[i]:
    correct_predictions += 1

accuracy = correct_predictions / len(y_test_subset)
print("Accuracy:", accuracy)
```

Concluzii

Codul implementat reușește să clasifice textele pe baza unui algoritm simplu Naiv Bayes, utilizând probabilități priors și probabilități condiționate pentru cuvintele din fiecare categorie, având o acuratețe de 90

Exemplu de Set de Date

Setul de date utilizat se poate găsi aici. Acestea reprezintă o serie de articole de știri clasificate în 4 categorii: politică, tehnologie, divertisment și afaceri, categorii indicate sub forma unui număr (0, 1, 2, 3).

Un exemplu de format al setului de date:

STORY	SECTION
"Aceasta este o poveste de știri..."	1
"Un alt exemplu de poveste aici..."	2

Fiecare rând conține un text (în coloana `STORY`) și eticheta corespunzătoare (în coloana `SECTION`).

Referințe

- Clasificator Naiv Bayes pe Wikipedia
- Clasificator Naiv Bayes în Scikit-Learn
- Laplace Smoothing