NAI – mały projekt programistyczny 4

Naiwny klasyfikator Bayesa

Napisz naiwny klasyfikator Bayesa odróżniający spam od nie-spamu w wiadomościach e-mail, na podstawie występowania *n* słów kluczowych w danym e-mailu.

Program powinien pobierać następujące argumenty:

- `word-list`: lista słów kluczowych – plik tekstowy zawierający po 1 słowie w kolejnych liniach. Słowa składają się z małych liter; mogą zawierać polskie znaki.
  Przykładowa lista słów kluczowych (dla *n* = 160) znajduje się w Teamsach w lokalizacji `mpp4/word_list.txt`.
- `train-data`: plik z danymi treningowymi w formacie CSV (oddzielonymi przecinkami), w którym każdy wiersz odpowiada jednej wiadomości e-mail i zawiera *n*+1 kolumn. W *i*-tej kolumnie (dla *i* <= *n*) znajduje się 1, jeśli e-mail zawiera przynajmniej 1 wystąpienie *i*-tego słowa kluczowego i 0 w przeciwnym wypadku.
  W ostatniej kolumnie znajduje się 1 w przypadku nie-spamu i 0 w przypadku spamu.
  Przykładowy plik treningowy znajduje się w Teamsach w lokalizacji `mpp4/data.csv`.
- `test-data`: plik z danymi testowymi w formacie takim samym, jak `train-data`.

Program powinien:

1. Sklasyfikować każdy przykład testowy na podstawie zbioru treningowego.
2. Wypisać macierz omyłek (wyniki klasyfikacji vs faktyczne kategorie dla danych testowych).
3. Wypisać dokładność oraz F-miarę przeprowadzonej klasyfikacji.

Uwaga: Należy radzić sobie z dowolną wartością *n*.

Rady techniczne:

1. Dane treningowe i testowe należy sobie przygotować samodzielnie. W wersji mniej pracochłonnej wystarczy podzielić przykładowy plik `data.csv` na część treningową i testową (w tym wypadku proszę zadbać o rozsądną proporcję wielkości, ale też rozsądną reprezentację obu kategorii w obu zbiorach!).
   Ciekawszą, choć trudniejszą opcją jest użyć `data.csv` jako danych treningowych, zaś testowe zbudować w oparciu o własne maile, które trzeba przerobić na wektory zero-jedynkowe w zależności od występowania słów kluczowych. (Warto przy tym

skonwertować treść maila na małe litery). Mogą też Państwo podmienić sobie listę słów kluczowych na bardziej odpowiadającą Państwa danym.

2. Mnożenie ciągu prawdopodobieństw – zwłaszcza niewielkich – może prowadzić do sporych błędów precyzji (zwłaszcza w językach typu Java / C). Oczywiście można temu zaradzić zwiększając precyzję obliczeń (`float` → `double` → `BigDecimal` itp.), jednak standardowym podejściem jest tu pracowanie na logarytmach (o dowolnej ustalonej podstawie) odpowiednich prawdopodobieństw. Mnożenie / dzielenie / porównywanie prawodpodobieństw odpowiada wtedy dodawaniu / odejmowaniu / porównywaniu logarytmów, a w praktyce precyzja rachunków może znacznie wzrosnąć.
3. Oczywiście prawdziwe klasyfikatory spamu są skuteczniejsze, i to z kilku powodów, takich jak rozmiar danych oraz zrozumienie języka. Ważnym aspektem jest tu też personalizacja – listę słów kluczowych i reguły klasyfikacji warto dostosować do konkretnego adresata. Przykładowa lista słów pochodzi z moich maili spamowych i nie-spamowych, ale jej przydatność (również dla mnie) spadła wskutek usunięcia z niej danych “osobistych”.

Uwaga: Program należy wykonać samodzielnie. **Plagiat** lub **niezrozumienie** rozwiązania skutkuje **brakiem zaliczenia projektu**.

Nie można korzystać z gotowych bibliotek do uczenia maszynowego ani operacji na wektorach. Wszystkie szczegóły algorytmu należy samodzielnie przećwiczyć kodując.

Termin: **29 maja**