

Pracownia problemowa
Projekt końcowy. Rozpoznawanie spamu.

Robert Nowotniak, 120308
Michał Wysokiński, 120404

18 stycznia 2008

Grupa laboratoryjna: Piątek, 12:45

Spis treści

1	Opis problemu	1
2	Opis rozwiązania	2
2.1	Testowe zbiory danych	3
2.2	Stemmer	5
2.3	Synonymizer	5
3	Zaimplementowane klasyfikatory	6
3.1	Naiwny klasyfikator Bayesowski	6
3.2	Log likelihood	7
3.3	Hidden Markov Model	8
3.4	Porównanie wyników	10
4	Wnioski	12

1 Opis problemu

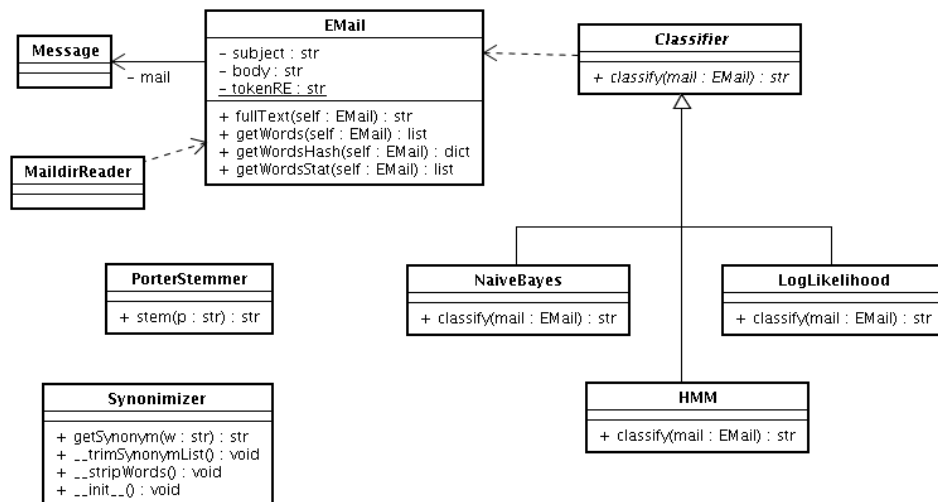
Projekt, który zrealizowaliśmy na „Pracowni Problemowej”, dotyczył zagadnienia rozpoznawania spamu za pomocą różnego rodzaju klasyfikatorów.

Projekt zrealizowaliśmy w języku Python. Wydajność tego języka okazała się wystarczająca dla rozmiarów naszych danych testowych.

Klasyfikatory, które zaimplementowaliśmy i porównaliśmy, to: naiwny klasyfikator Bayesowski, *log likelihood* oraz HMM (ang. *Hidden Markov Model*).

Założyliśmy, że będziemy rozważać tylko maile w języku angielskim (ma to znacznie np. na etapie stemmingu).

2 Opis rozwiązania



Rysunek 1: Diagram klas w programie

Diagram klas naszego systemu przedstawiony jest na rysunku 1. Ponieważ projekt zrealizowaliśmy w języku Python, w którym nie istnieją klasy i metody abstrakcyjne ani interfejsy, w kodzie programu nie pojawia się bezpośrednio klasa *Classifier*.

Pojedynczy mail jest reprezentowany jako obiekt typu *EMail*, w którym interesujące są dla nas jedynie pola tematu oraz treści maila. Nie wykonujemy analizy na podstawie innych przesłanek wynikających z nagłówka maila (adres IP, RevDNS, czas wysłania, strefa czasowa itp).

Klasa *EMail* opakowuje klasę *Message* z pakietu *email* biblioteki standardowej języka Python.

Klasy pomocnicze *PorterStemmer* oraz *Synonimizer* wykonują odpowiednio stemming oraz znajdowanie synonimów (tezaurus), co będzie opisane w dalszej części.

Klasa *EMail* jest wykorzystywana przez każdą z klas rozszerzających naszą abstrakcyjną klasę klasyfikatora *Classifier*: *NaiveBayes*, *HMM*, *LogLikelihood* — w których nadpisywane są definicje abstrakcyjnej metody *classify*, która ma zwracać etykietę przypisywaną mailowi.

W naszej klasie *EMail* znajdują pomocnicze funkcje, przydatne w procesie klasyfikacji. Np. metoda **getWordsHash** zwraca tablicę asocjacyjną zawierającą liczby występowania poszczególnych słów z treści maila:

```
{'help': 1, 'seeing': 1, 'rapture': 1, 'soon': 1, 'paper': 2, 'tempted': 1, 'children': 1, 'certainly': 1, 'thinking': 1, 'young': 2, 'answered': 1, 'pretty': 1, 'smile': 2,
```

Tabela 1: Używane zbiory danych

maildir	ilość maili	zawartość
ham1/	1000	zbiór treningowy z nie-spamem
ham2/	1000	zbiór testowy z nie-spamem
spam1/	500	zbiór treningowy ze spamem
spam2/	500	zbiór testowy ze spamem

'hope': 1, 'good': 1, 'contention': 1, 'listened': 1,
 'leisure': 1, 'choice': 1, 'early': 1, 'earnest': 1,
 ...

2.1 Testowe zbiory danych

Do celów testowych przygotowaliśmy cztery zbiory danych – zbiory maili w postaci maildir(5). Dwa zbiory ze spamem (treningowy i testowy) oraz z hamem¹ (treningowy i testowy).

Używane dane (zbiory maili) pochodziły z naszych własnych kont pocztowych, z pewnego okresu w przeszłości – z folderów zawierających spam oraz „dobre” maile.

Tematy losowej próbki spamu:

```
Re: Confirm pr0pecia Order 24901
Buy cheap Canadian drugs and start saving now with CanadianPharmacy.
Confirm iwc Order 07127
ipomanus
Would you like to work with our team
Confirm pharm4cy Order 5777437
Re: Confirm xan4x Order 16929
CIA-LIS - Cheapest Pri-ces and 100% Satisfaction Guaranteed!
Medications that you need.
Re: Confirm C1alis s0ft T4bs Order 04337
```

Tematy losowej próbki hamu:

```
Re: Problems with pkgconfig regarding avahi and qt-mt
Re: Trouble accessing archlinux.org
Can Gnome-Screensaver be set to only respond to the keyboard?
Re: recently started crashing 4 am
Re: Question about LVM and RAID
Re: AUR Broken ?
Creating Backdoors in Cisco IOS using Tcl
Re: conservative/stable branch
```

¹ham – maile nie będące spamem

Re: FTP Suggestion
Re: conservative/stable branch
IDE adapter for FC System
Old package version

Pewną charakterystyką zbioru spamu i hamu jest statystyka występujących w nich słów — mimo że *de facto* nie jest istotna sama liczność występowania poszczególnych słów, lecz bardziej wyszukane relacje między nimi.

time : 621
bloom : 537
man : 444
rose : 397
look : 394
face : 385
day : 373
head : 364
eyes : 364
replica : 351
young : 340
mother : 328
love : 327
polly : 295
dear : 295
heart : 293
going : 291
saw : 289
soft : 288
gift : 288
things : 287
looked : 283
life : 281
cialis : 275
father : 264
gave : 263
better : 263
won : 260
boys : 258
boy : 258
girls : 252

W naszym systemie wykorzystujemy też listę słów ze słownika biblioteki cracklib (/usr/share/dict/cracklib-small). oraz stoplistę oraz „allowlistę” (w klasyfikatorze HMM).

Listy te dostosowane są do języku angielskiego — początkowy fragment stoplisty wygląda zatem następująco:

```
a
about
above
across
after
afterwards
again
against
...
```

Znajdują się tu wszystkie słowa ze „gotowej” stoplisty dla języka angielskiego oraz słowa dodane przez nas, które zwróciły na siebie uwagę w naszym testowym zbiorze maili.

2.2 Stemmer

Stemming to proces upraszczania odmienionych form wyrazowych – i otrzymywania z nich samego rdzenia wyrazowego.

Używany przez nas stemmer działa wg gotowego algorytmu Portera (Porter Stemming Algorithm). Ten mechanizm działa jedynie dla języka angielskiego.

2.3 Synonymizer

W systemie wykorzystujemy także „synonymizer” (tezaurus), wykorzystując słownik z pakietu biurowego OpenOffice — a dokładniej pliki: `th_en_US_v2.dat` i `th_en_US_v2.idx`.

Synonymizer przeprowadza następujące zamiany słów (na przykładzie fragmentu naszego zbioru testowego).

Jak pokazały ostateczne wyniki, użycie takiego słownika (tezaury) powoduje jedynie pogorszenie rezultatów klasyfikacji, ponieważ — jak można zauważyć poniżej — można w ten sposób uzyskać słowa o nieco odbierającym znaczeniu (np. `installation` → `payment`), co wpływa nie nieprawidłowe klasyfikowanie pewnej części maili.

```
dead -> asleep
locked -> barred
installation -> payment
running -> afoot
time -> lawsuit
shutdown -> closure
message -> communicating
```

failed -> attempted
mind -> encephalon
today -> present
services -> activity
saw -> proverb
services -> activity
status -> government
dead -> asleep
locked -> barred
far -> deep
failing -> attempted
services -> activity
curious -> even
balancing -> equalisation
services -> activity
problem -> trouble
way -> manner
service -> activity
way -> manner
installation -> payment
thanks -> recognition
problem -> trouble
fairly -> reasonably

3 Zaimplementowane klasyfikatory

Klasyfikatory, które zaimplementowaliśmy to klasyfikator bayesowski (*naive*), *log likelihood* oraz ukryty model Markova (*Hidden Markov Model*).

Klasyfikatory oraz otrzymane przy ich użyciu wyniki opisaliśmy w kolejnych podrozdziałach.

3.1 Naiwny klasyfikator Bayesowski

Klasyfikatory Bayesowskie opierają się na twierdzeniu Bayesa (wzór 1), pozwalającym na określenie prawdopodobieństwa zachodzenia jakiejś hipotezy, gdy wiadome jest, że wystąpiło zdarzenie, które jest – z pewnym prawdopodobieństwem – skutkiem tej hipotezy.

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \quad (1)$$

Inaczej mówiąc, można to wykorzystać do przypisywania obiektom prawdopodobieństw przynależności do różnych klas, jeśli potrafimy oszacować prawdopodobieństwo występowania możliwych kombinacji cech w poszczególnych klasach.

Jeżeli mamy zatem pewien zbiór – zadanych z góry – klas $V = \{v_1, v_2, \dots, v_n\}$, to rezultatem działania klasyfikatora będzie wybranie klasy V_{nb} , której iloczyn prawdopodobieństwa apriorycznego $P(v_j)$ oraz prawdopodobieństw warunkowych $P(a_i|v_j)$ jest maksymalny.

$$V_{nb} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_{i=1}^n P(a_i|v_j) \quad (2)$$

gdzie $a_i = a_1, \dots, a_n$ to kolejne atrybuty klasyfikowanego obiektu.

W przypadku klasyfikacji maili, po podzieleniu analizowanej treści na słowa i wykonaniu ewentualnego innego przetwarzania wstępnego (stemming), prawdopodobieństwo zaklasyfikowania maila do którejś grupy przedstawiają wzory 3 i 4.

$$Pr(\text{spam}|\text{tokens}) = \frac{Pr(\text{tokens}|\text{spam})Pr(\text{spam})}{Pr(\text{tokens})} \quad (3)$$

$$Pr(\text{ham}|\text{tokens}) = \frac{Pr(\text{tokens}|\text{ham})Pr(\text{ham})}{Pr(\text{tokens})} \quad (4)$$

Ponieważ w 3 i 4 występują te same wyrażenia w mianowniku, a zależy nam jedynie stwierdzeniu relacji pomiędzy 3 i 4, możemy rozważać tylko wartości pojawiające się w licznikach.

Wersje **naiwna** klasyfikatora Bayesowskiego opiera się na założeniu niezależności prawdopodobieństw kolejnych cech. W takim przypadku możliwe jest przybliżenie:

$$Pr(\text{tokens}|\text{spam}) \approx Pr(t1|\text{spam}) \cdot Pr(t2|\text{spam}) \cdot \dots \cdot Pr(tN|\text{spam}) \quad (5)$$

3.2 Log likelihood

Klasyfikator *log likelihood* w dalszym ciągu korzysta z założeń niezależności prawdopodobieństw cech (5) z poprzedniego podrozdziału, ale dodatkowo przy obliczaniu prawdopodobieństw wykorzystywana jest własność funkcji *log*.

W ogólnym przypadku wartość prawdopodobieństwa *log likelihood* oblicza się za pomocą wzorów 6 i 7, gdzie O_i – *observed frequencies*, E_i – *expected frequencies*, N_i – kardynalności zbiorów słów.

$$E_i = \frac{N_i \sum_i O_i}{\sum_i N_i} \quad (6)$$

$$-2\ln\lambda = 2 \sum_i O_i \ln \left(\frac{O_i}{E_i} \right) \quad (7)$$

Wykorzystanie w wyrażeniach funkcji logarytmicznej ma na celu zredukowanie znacznego przeważania obecności pewnych słów w analizowanych

treściach. Funkcja \log jest funkcją monotoniczną, rosnącą, ale rośnie w coraz wolniejszym tempie – więc można ją wykorzystać do „wysycania” obliczanych wartości.

W naszym zastosowaniu — klasyfikacji maili — bardzo duża częstotliwość występowania jakiegoś słowa nie powinna oznaczać proporcjonalnego wzrostu pewności dokonywanej klasyfikacji, a jedynie powinna zwiększać do prawdopodobieństwa w pewnym stopniu.

W tym klasyfikatorze wykorzystuje się przede wszystkim własność $\log(a \cdot b) = \log(a) + \log(b)$ w następujących przekształceniach prawdopodobieństwa:

$$\log P(\text{words}) \approx \log \prod_{i=1}^n P(t_i) = \sum_{i=1}^n \log P(t_i) \quad (8)$$

3.3 Hidden Markov Model

Kolejnym wykorzystanym przez nas klasyfikatorem jest ukryty model Markowa (ang. *Hidden Markov Model*).

Własność markowa jest spełniona wtedy gdy decyzja w procesie statystycznym jest zdeterminowana jedynie bieżącym stanem, formalnie:

$$P(X_{t+1}|X_1, \dots, X_t) = P(X_{t+1}|X_t)$$

Macierz probabilistyczna to taka macierz dodatnich liczb rzeczywistych, w której suma wartości w wierszach jest równa 1.

Ukryty proces markowa jest probabilistycznym automatem spełniającym własność markowa, jest on zdefiniowany jako trójka:

$$\lambda = (A, B, \pi)$$

gdzie:

A – Probabilistyczna macierz przejścia, $a_{ij} = P(x_{t+1} = j | x_t = i)$

B – Probabilistyczna macierz emisji symbolu, $b_{ij} = P(o_t = v_j | x_t = i)$

π – Probabilistyczny wektor prawdopod. początkowych, $\pi_i = P(x_1 = i)$

Wartość a_{ij} macierzy A mówi jakie jest prawdopodobieństwo przejścia ze stanu i do stanu j. Natomiast b_{ij} określa jakie jest prawdopodobieństwo emisji symbolu j w stanie i. Wartość π_i określa z kolei jakie jest prawdopodobieństwo iż proces rozpocznie się w stanie i.

Użyte oznaczenia: x_t – stan w chwili t

O – obserwowany ciąg wyjściowy

o_t – t-ety element ciągu O

T – długość ciągu O

v_k – k symbol wejściowy z pośród N symboli

N – wielkość alfabetu symboli wyjściowych

M – ilość stanów ukrytych

Pierwszym ważnym zagadnieniem tego modelu jest obliczenie prawdopodobieństwa wygenerowania danego ciągu O przez automat lambda $P(O|\lambda)$ (ang. *evaluation problem*), należy obliczyć sumę prawdopodobieństw wszystkich możliwych ścieżek od długości T (oznaczymy je przez S), które generują ciąg O :

$$\sum_{s \in S} P(O|s)$$

Algorytm taki jest złożoności $O(M^2T^2)$, lepszym rozwiązaniem jest użycie algorytmu “forward” którego złożoność wynosi $O(M^2T)$. Oznaczmy przez α_{tj} prawdopodobieństwo iż w chwili t znaleźliśmy się w stanie j , a zaobserwowaną sekwencją jest o_1, \dots, o_t :

$$\alpha_{tj} = P(o_1, \dots, o_t, x_t = j | \lambda)$$

można je obliczyć z rekurencyjnego wzoru:

$$\begin{aligned} \alpha_{1j} &= \pi_j b_{j o_1} \\ \alpha_{tj} &= b_{j o_t} \sum_{i=1}^N \alpha_{(t-1)i} a_{ij} \end{aligned}$$

Algorytm “forward” definiuje się następującym wzorem:

$$P(O|\lambda) = \sum_{i=1}^T \alpha_{Ti}$$

Drugim zagadnieniem jest obliczenie najbardziej prawdopodobnej ścieżki e , której przejście generuje obserwowany ciąg z największym prawdopodobieństwem, do jego obliczenia służy algorytm viterbi i nie zosanie tu omówiony ponieważ nie został wykorzystany w projekcie.

Trzecim problemem jest odnalezienie ukrytego modelu markova, który maksymalizuje prawdopodobieństwo wygenerowania rozważanego ciągu (ciągów) obserwacji, jest to problem trudny i do tej pory nie został w pełni rozwiązany, znane są gradientowe algorytmy nauki oraz metoda Bauma-Welcha jest to algorytm typu EM (Expectation - Maximalization). EM to uniwersalna metoda postępowania, polega ona na cyklicznym powtarzaniu dwóch kroków: przewidywaniu pewnych parametrów, a następnie wyliczeniu zmiennych maksymalizujących pewną funkcję celu. Algorytm gwarantuje zbieżność jedynie do lokalnych maximów, których jest bardzo dużo na optymalizowanej powierzchni.

Aby opisać ostatni z algorytmów wprowadźmy trzy następujące zmienne: $\beta_t(j)$, $\xi_t(i, j)$ oraz $\gamma_t(i)$. Pierwsza zmienna (ang. backward variable) wyznacza prawdopodobieństwo zaobserwowania ciągu o_{t+1}, \dots, o_T pod warunkiem, że w chwili t znajdowaliśmy się w stanie j , formalnie:

$$\beta_t(j) = P(o_{t+1}, \dots, o_T | x_t = j)$$

jest ona podobnie jak zmienna α dana wzorem indukcyjnym o postaci:

$$\begin{aligned}\beta_T(i) &= 1 \\ \beta_t(i) &= \sum_{j=1}^N a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)\end{aligned}$$

Druga zmienna $\xi_t(i, j)$ oznaczająca prawdopodobieństwo znalezienia się w chwili t w stanie i oraz w chwili $t+1$ w stanie j :

$$\xi_t(i, j) = P(x_t = i, x_{t+1} = j | O)$$

Prawdopodobieństwo to dane jest wzorem:

$$\xi_t(i, j) = \frac{\alpha_{ti} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O | \lambda)}$$

Trzecia zmienna $\gamma_t(i)$ oznaczająca prawdopodobieństwo znalezienia się w chwili t w i -tym stanie, czyli $P(x_t = i | O)$. Oblicza się ją z następującego wzoru:

$$\gamma_t(i) = \sum_{j=1}^N \xi_t(i, j)$$

Mając powyższe wzory możemy przejść do reestymacji parametrów automatu, są one dane wzorami:

$$\begin{aligned}\bar{\pi}_i &= \gamma_1(i) \\ \bar{a}_{ij} &= \frac{\sum_{t=1}^T \xi_t(i, j)}{\sum_{t=1}^T \gamma_t(i)} \\ \bar{b}_{jk} &= \frac{\sum_{t=1}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}\end{aligned}$$

Metoda Bauma-Welcha gwarantuje że $\bar{\lambda}(\bar{A}, \bar{B}, \bar{\pi}) > \lambda(A, B, \pi)$ w sensie prawdopodobieństwa z jakim automaty generują ciąg obserwacji O , bądź $\bar{\lambda} = \lambda$ jeżeli znaleźliśmy się w którymś z maximów. Opis algorytmu został oparty na pracy Lawrence'a R. Rabinera [Rab].

3.4 Porównanie wyników

Przetestowanie wszystkich klasyfikatorów na całym testowym zbiorze, wraz z możliwymi ustawieniami parametrów dodatkowych (stemming, synonimy) możliwe jest za pomocą skryptu `testall.py`.

W wyniku uruchomienia skryptu otrzymujemy ostatecznie następujące zestawienie rezultatów:

Reading training data...

Training data have been read successfully

Reading test data...

Testing data have been read successfully

Classifying...

Classifier: bayes.NaiveBayes (Stemming: False, Synonyms: False)

Spam messages classified: 433/500 (86%)

False positives: 0/500 (0%)

Classification time: 7.768s total, 0.016s/mail

Classifier: bayes.NaiveBayes (Stemming: True, Synonyms: False)

Spam messages classified: 435/500 (87%)

False positives: 0/500 (0%)

Classification time: 31.102s total, 0.062s/mail

Classifier: bayes.NaiveBayes (Stemming: False, Synonyms: True)

Spam messages classified: 423/500 (84%)

False positives: 0/500 (0%)

Classification time: 10.973s total, 0.022s/mail

Classifier: bayes.NaiveBayes (Stemming: True, Synonyms: True)

Spam messages classified: 336/500 (67%)

False positives: 0/500 (0%)

Classification time: 62.580s total, 0.125s/mail

Classifier: loglikelihood.LogLikelihood (Stemming: False, Synonyms: False)

Spam messages classified: 483/500 (96%)

False positives: 17/500 (3%)

Classification time: 8.116s total, 0.016s/mail

Classifier: loglikelihood.LogLikelihood (Stemming: True, Synonyms: False)

Spam messages classified: 478/500 (95%)

False positives: 50/500 (10%)

Classification time: 32.374s total, 0.065s/mail

Classifier: loglikelihood.LogLikelihood (Stemming: False, Synonyms: True)

Spam messages classified: 491/500 (98%)

False positives: 55/500 (11%)

Classification time: 12.136s total, 0.024s/mail

Classifier: loglikelihood.LogLikelihood (Stemming: True, Synonyms: True)

Spam messages classified: 490/500 (98%)

False positives: 275/500 (55%)

```

Classification time: 63.151s total, 0.126s/mail
-----
Przykładowe wyniki hmm:
states number: 10
good: 953, falsenegatives: 440, falsepositives: 100, accuracy: 0.638312
-----
states number: 15
good: 953, falsenegatives: 440, falsepositives: 100, accuracy: 0.638312
-----
states number: 25
good: 962, falsenegatives: 415, falsepositives: 116, accuracy: 0.644340
-----
states number: 25
good: 1048, falsenegatives: 409, falsepositives: 36, accuracy: 0.701942
-----
states number: 45
good: 775, falsenegatives: 416, falsepositives: 302, accuracy: 0.519089

```

4 Wnioski

Za pomocą samego naiwnego klasyfikatora Bayesowskiego udało nam się uzyskać 86% poprawnego rozpoznawania spamu oraz 0% *false positives*. Użycie klasyfikatora *log likelihood* poskutkowało rozpoznaniem 97% spamu jako spam, ale 2% „false positives”. W przypadku używania stemmingu poprawność klasyfikatora Bayesowskiego nieznacznie wzrosła do 87% prawidłowego rozpoznawania spamu, przy zachowaniu 0% false positives. Użycie synonimów w przypadku klasyfikatora Bayesowskiego nieznacznie pogorszyło klasyfikację.

Dla klasyfikatora LogLikelihood poprawność rozpoznawania spamu w każdym przypadku była bardzo duża — powyżej 95%. Niestety jednocześnie, w najlepszym przypadku, „false positives” wynosiło aż 3%. Przy jednoczesnym użyciu stemmingu i synonimizera „false positives” wzrosło aż do, zupełnie nieakceptowalnej, wartości 55%.

W każdym przypadku użycie algorytmu stemmingu (algorytm Portera) zwiększało czas klasyfikacji aż czterokrotnie. Natomiast włączenie synonimów zwiększało ten czas około 1,5-krotnie. Te wyniki dałoby się jednak z pewnością poprawić, optymalizując te algorytmy lub przepisując je w języku C lub C++.

W naszym projekcie mieliśmy możliwość zapoznania się z najbardziej podstawowymi metodami wykorzystywanymi w klasyfikacji tekstu (stemming, używanie tabeli synonimów). Skupiliśmy się jedynie na klasyfikacji na podstawie zawartości tematu oraz treści maili. Obecnie najlepsze rezulta-

ty w wykrywaniu spamu uzyskuje się przez łączenie wielu — także bardzo prostych — metod (zwłaszcza Greylisting, SPF).

Zapoznaliśmy się z działaniem klasyfikatora Bayesowskiego przy klasyfikacji spamu — jednego z najczęściej używanych klasyfikatorów w wielu współczesnych systemach antyspamowych (SpamAssassin, Bogofilter, Spam-Bayes, Mozilla Thunderbird). Możliwości tego klasyfikatora są powszechnie wykorzystywana w takim zadaniu od czasu publikacji [Gra02, Gra03].

Klasyfikator oparty na ukrytych modelach Markowa dawał gorsze wyniki niż dwie pozostałe metody. Najlepszym uzyskanym przez nas wynikiem było 70% przy 2% maili niepoprawnie sklasyfikowanych jako spam. Największą przeszkodą w zwiększeniu poprawności klasyfikacji był czas potrzebny na wyuczenie tego klasyfikatora. Przeprowadzenie dwóch cykli uczących na klasyfikatorze, ze stosunkowo małą liczbą pól ukrytych wynoszącą 25, trwało około dwóch godzin i nie dawało praktycznie żadnej poprawy wyników. Można przyjąć, że na optymalizowanej powierzchni było sporo niewysokich lokalnych maksimów, w których utyka algorytm Bauma-Welcha. Jedynym praktycznym sposobem na sprawdzenie tej hipotezy byłoby przeprowadzenie wielu testów na kilku automatach, jednak z powodów praktycznych (powolności nauki) było to niemożliwe. Zaobserwowaliśmy również, iż zmniejszanie alfabetu znacznie pogarsza wyniki, spowodowane jest to odrzucaniem wielu słów świadczących o przynależności maila do danej klasy np.: w niektórych alfabetach odrzucane było słowo „*cialis*”, które prawie napewno klasyfikuje daną pocztę do grupy spamu. Dodatkowym ograniczeniem ukrytych modeli Markova jest możliwość działania na z góry określonym alfabecie, co w przypadku rzeczywistym jest raczej nie praktyczne. Słowa spoza tego alfabetu zostają odrzucone i nie są brane pod uwagę. Możliwe jest wykorzystanie ogromnego alfabetu, składającego się z wszystkich słów analizowanego języka oraz pewnych nazw własnych. Jednak nasze testy, przeprowadzone dla największego alfabetu złożonego z około 20 000 słów, wskazały, że praktyczne nauczenie takiego modelu jest bardzo trudne. Tak więc przy bardzo dobrej skuteczności i prędkości działania dwóch uprzednio opisanych klasyfikatorach model HMM zdaje się być niepraktyczny w naszym zastosowaniu.

W naszych klasyfikatorach stosowaliśmy także zasadę, że liczy się jedynie różnica między prawdopodobieństwami przynależności do jednej z klas. Przy niewielkich różnicach między tymi prawdopodobieństwami można byłoby jednak klasyfikować maile do jeszcze innej klasy (np. „*unsure*”), aby minimalizować ryzyko nieprawidłowego zaklasyfikowania maila, nie będącego spamem.

Literatura

- [Klo01] Mieczysław Alojzy Kłopotek. *Inteligentne wyszukiwarki internetowe*. EXIT, Warszawa, 2001
- [Cic07] Paweł Cichosz. *Systemy uczące się*. WNT, 2007
- [PS03] Fuchun Peng, Dale Schuurmans *Combining Naive Bayes and n-Gram Language Models for Text Classification*. 2003
- [Gra02] Paul Graham. *A Plan for Spam*. 2002
- [Gra03] Paul Graham. *Better Bayesian Filtering*. 2003
- [SDHH] M. Sahami, S. Dumais, D. Heckerman, E. Horvitz *A Bayesian approach to filtering junk e-mail*. *AAAI'98 Workshop on Learning for Text Categorization*. 1998
- [AC05] David Ahn, Balder Cate *Simple language models and spam filtering with Naive Bayes*. 2005
- [Rab] Lawrence R. Rabiner *A Tutorial on Hidden Markov Model and Selected Applications in Speech Recognition*