FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Semestrální práce z předmětu
Programování v jazyce C

Identifikace spamu naivním bayesovským klasifikátorem

Autor:
Stanislav Kafara
A21B0160P
skafara@students.zcu.cz

Vyučující:
Ing. Kamil Ekštein, Ph.D.
kekstein@kiv.zcu.cz

Obsah

1 Zadání **3**

2 Analýza úlohy **5**
2.1 Datové soubory 5
2.2 Trénování 6
2.3 Klasifikace 6
2.4 Reprezentace dat klasifikátoru 7
2.4.1 Využití datové struktury 7
2.4.2 Vhodná datová struktura 7

3 Popis implementace **9**
3.1 Balík `utilities` 9
3.1.1 Modul `utils` 9
3.1.2 Modul `primes` 9
3.1.3 Modul `arrays` 11
3.2 Balík `structures` 11
3.2.1 Modul `vector` 11
3.2.2 Modul `hashtable` 11
3.3 Modul `classifier` 12
3.3.1 Deklarace a definice 12
3.3.2 Trénování 13
3.3.3 Klasifikace 14
3.4 Modul `spamid` 14

4 Uživatelská příručka **16**
4.1 Přeložení a sestavení programu 16
4.2 Datové soubory 16
4.3 Spuštění programu 17
4.4 Výstup programu 17

5 Závěr **18**
5.1 Testování 18

Kapitola 1

Zadání

Naprogramujte v ANSI C přenositelnou[1] **konzolovou aplikaci**, která bude **rozhodovat, zda úsek textu** (textový soubor předaný jako parametr na příkazové řádce) **je nebo není spam**.

Program bude přijímat z příkazové řádky celkem **sedm** parametrů: První dva parametry budou vzor jména a počet trénovacích souborů obsahujících nevyžádané zprávy (tzv. **spam**). Třetí a čtvrtý parametr budou vzor jména a počet trénovacích souborů obsahujících vyžádané zprávy (tzv. **ham**). Pátý a šestý parametr budou vzor jména a počet testovacích souborů. Sedmý parametr představuje jméno výstupního textového souboru, který bude po dokončení činnosti Vašeho programu obsahovat výsledky klasifikace testovacích souborů.

Program se tedy bude spouštět příkazem

`spamid.exe`[2] ⟨`spam`⟩ ⟨`spam-cnt`⟩ ⟨`ham`⟩ ⟨`ham-cnt`⟩ ⟨`test`⟩ ⟨`test-cnt`⟩ ⟨`out-file`⟩

Symboly ⟨`spam`⟩, ⟨`ham`⟩ a ⟨`test`⟩ představují vzory jména vstupních souborů. Symboly ⟨`spam-cnt`⟩, ⟨`ham-cnt`⟩ a ⟨`test-cnt`⟩ představují počty vstupních souborů. Vstupní soubory mají následující pojmenování: `vzorN`, kde `N` je celé číslo z intervalu $\langle 1; N \rangle$. Přípona všech vstupních souborů je `.txt`, přípona není součástí vzoru. Váš program tedy může být během testování spuštěn například takto:

```
spamid.exe spam 10 ham 20 test 50 result.txt
```

[1] Je třeba, aby bylo možné Váš program přeložit a spustit na PC s operačním prostředím Win32/64 (tj. operační systémy Microsoft Windows NT/2000/XP/Vista/7/8/10/11) a s běžnými distribucemi Linuxu (např. Ubuntu, Debian, Red Hat, atp.). Server, na který budete Vaši práci odevzdávat a který ji otestuje, má nainstalovaný operační systém Debian GNU/Linux 10 (buster) s jádrem verze 4.19.0-11-amd64 a s překladačem gcc 8.3.0.

[2] Přípona `.exe` je povinná i při sestavení v Linuxu, zejm. při automatické kontrole validačním systémem.

Výsledkem činnosti programu bude textový soubor, který bude obsahovat seznam testovaných souborů a jejich klasifikaci (tedy rozhodnutí, zda je o spam či neškodný obsah – ham).

Pokud nebude na příkazové řádce uvedeno právě sedm argumentů, vypište chybové hlášení a stručný návod k použití programu v angličtině podle běžných zvyklostí (viz např. ukázková semestrální práce na webu předmětu Programování v jazyce C). **Vstupem programu jsou pouze argumenty na příkazové řádce – interakce s uživatelem pomocí klávesnice či myši v průběhu práce programu se neočekává.**

Hotovou práci odevzdejte v jediném archivu typu ZIP prostřednictvím automatického odevzdávacího a validačního systému. Postupujte podle instrukcí uvedených na webu předmětu. Archiv nechť obsahuje všechny zdrojové soubory potřebné k přeložení programu, **makefile** pro Windows i Linux (pro překlad v Linuxu připravte soubor pojmenovaný `makefile` a pro Windows `makefile.win`) a dokumentaci ve formátu PDF vytvořenou v typografickém systému TEX, resp. LATEX. Bude-li některá z částí chybět, kontrolní skript Vaši práci odmítne.

Kapitola 2

Analýza úlohy

Program má postupovat ve dvou fázích. Nejdříve nechá trénovacími soubory natrénovat klasifikátor, kterým následně klasifikuje testované soubory. Úloha se skládá z několika problémů, které je nutné analyzovat a jimiž jsou:

- čtení slov ze souborů,
- trénování,
- klasifikace a
- reprezentace dat.

V této kapitole je odkazováno na zadání práce[1].

2.1 Datové soubory

Datové soubory obsahují právě jednu řádku mezerou oddělených slov, a tak není potřeba je nijak předzpracovávat. Je ale potřeba zamyslet se nad tím, kdy a jak budou slova ze souborů načítána.

Je možné zvolit mezi přístupy načtení slov ze souborů dopředu, a nebo načítání slov ze souborů během trénování. Pokud by byly datové soubory příliš velké, načtená slova by se nemusela vejít do operační paměti, čemuž se dá částečně zabránit druhým přístupem, který bude využíván.

Další věc, nad kterou je potřeba se zamyslet, je maximální délka slova. Pokud by datové soubory obsahovaly slova přirozených jazyků, pro načtení slova by postačovala statická vyrovnávací paměť o určité velikosti. Jinou možností může být např. nejprve analyzovat datový soubor a následně s pomocí

[1] `https://www.kiv.zcu.cz/studies/predmety/pc/data/works/sw2022-02.pdf`

dynamické alokace vytvořit dostatečně velkou vyrovnávací paměť. Pro jednoduchost bude zvolen přístup načítání slova do dynamického pole, který může být využit i v případě, že datové soubory obsahují řetězce jiného významu, než slova přirozeného jazyka a ze souboru je tak čteno pouze jednou.

2.2 Trénování

Natrénováním naivního Bayesovského klasifikátoru se rozumí vypočtení hodnot pro každou klasifikační třídu tohoto modelu, na základě nichž model následně klasifikuje testované datové soubory. Těmito hodnotami jsou $P(c)$ a $P(\langle\text{word}\rangle|c)$ a jsou vypočteny podle vzorce 2.1 a 2.2.

$P(c)$ je apriorní pravděpodobnost výskytu klasifikační třídy c v datových souborech. $P(\langle\text{word}\rangle|c)$ je pravděpodobnost výskytu slova $\langle\text{word}\rangle$ v datovém souboru za podmínky klasifikace datového souboru do klasifikační třídy c. Klasifikátor využívá tzv. *bag-of-words model*[2].

$$P(c_i) = \frac{|\mathbf{doc}_i|}{|\mathbf{trénovací\ množina}|} \tag{2.1}$$

$$P(\langle\text{word}\rangle|c_i) = \frac{n_k + 1}{n + |\mathbf{slovník}|} \tag{2.2}$$

Symbol c_i představuje i-tou klasifikační třídu, $|\mathbf{doc}_i|$ počet datových souborů patřící do i-té klasifikační třídy, $|\mathbf{trénovací\ množina}|$ celkový počet trénovacích datových souborů, $\langle\text{word}\rangle$ slovo, n počet slov (včetně duplicitních) v datových souborech patřících do i-té klasifikační třídy, n_k počet slov $\langle\text{word}\rangle$ v datových souborech patřících do i-té klasifikační třídy a $|\mathbf{slovník}|$ počet různých slov ve všech trénovacích datových souborech.

Lze vybírat mezi přístupy postupného trénování a přepočítávání pravděpodobností a nebo jednorázového natrénování několika datových souborů a jednorázového vypočtení pravděpodobností. Přístupy lze také kombinovat. První přístup v této úloze nenajde uplatnění, a samotný navíc zvyšuje časovou složitost trénování, proto bude využíván přístup druhý.

2.3 Klasifikace

Klasifikací datového souboru se rozumí přiřazení klasifikační třídy datovému souboru. Naivní Bayesův klasifikátor přiřazuje datovému souboru klasifikační

[2]Nepracuje se s pozicí výskytu slova v datovém souboru, nýbrž jen se skutečností jeho výskytu v něm.

třídu c na základě pravděpodobností vypočtených během trénování podle vzorce 2.3.

$$c = \arg\max_{c_i \in C} \left(P(c_i) \cdot \prod_{k \in \mathbf{pozice}} P(\langle \mathrm{word}_k \rangle | c_i) \right), \tag{2.3}$$

kde C je množina klasifikačních tříd (v této úloze pouze spam a ham), **pozice** je množina pozic slov v testovaném datovém souboru takových, že slovo na této pozici je ve slovníku klasifikátoru (nacházelo se v nějakém z trénovacích datových souborů) a $\langle \mathrm{word}_k \rangle$ je k-té slovo testovaného datového souboru. Avšak z důvodu možnosti podtečení rozsahu čísel s pohyblivou desetinnou čárkou je využívána zlogaritmovaná verze vzorce 2.4.

$$c = \arg\max_{c_i \in C} \left(\log P(c_i) + \sum_{k \in \mathbf{pozice}} \log P(\langle \mathrm{word}_k \rangle | c_i) \right). \tag{2.4}$$

2.4 Reprezentace dat klasifikátoru

Pro efektivní natrénování klasifikátoru a klasifikaci datových souborů je potřeba využít datovou strukturu poskytující přidání prvku a získání prvku se vhodnou časovou a paměťovou složitostí.

2.4.1 Využití datové struktury

K natrénování klasifikátoru je nutné provést $2 \cdot |\mathbf{slovník}|$ výpočtů pravděpodobnosti $P(\langle \mathrm{word} \rangle | c)$ podle vzorce 2.2, proto je potřeba mít hodnoty efektivně dostupné pro dosazení do vzorce.

Ke klasifikaci datového souboru podle vzorce 2.4 je nutné provést 2 komplexní výpočty vyžadující $|\mathbf{slovník}|$krát získání a dosazení hodnoty pravděpodobnosti $P(\langle \mathrm{word} \rangle | c)$ do vzorce.

2.4.2 Vhodná datová struktura

Nabízí se řešit problém pomocí hash tabulky, prefixové trie nebo binárního (případně i vyvažovaného) vyhledávacího stromu.

Paměťová složitost vybraných datových struktur je shodná v nejhorším případě. V průměrném případě by nejvhodnější volbou byla prefixová trie, potom hash tabulka a binárná vyhledávací strom.

Časová složitost vkládání prvku a získání prvku závisí u hash tabulky jen na délce řetězce, u trie navíc na počtu písmen abecedy jazyka a u binárního vyhledávacího stromu navíc ještě na počtu již vložených prvků v datové struktuře.

Pro očekávané vlastnosti hash tabulky je potřeba zajistit nejlépe rovnoměrné rozptýlení vkládaných prvků, řešení případných kolizí a přiměřenou velikost tabulky, nejlépe rovnu prvočíslu.

Z důvodu předpokladu zvýhodnění časové složitosti nad paměťovou složitostí a jednoduchosti implementace je zvolena datová struktura hash tabulka s dynamickou velikostí a s využitím rychlého testu prvočíselnosti.

Kapitola 3

Popis implementace

Zjednodušená adresářová struktura projektu s ukázkovými daty je vidět na obr. 3.1. Zdrojové kódy programu se nachází v adresáři `src` a datové soubory v adresáři `data`. V adresáři `doc/html` je dostupná vygenerovaná programová dokumentace programem `doxygen` ve formátu HTML.

Zdrojový kód programu je dělen podle významových bloků do balíků a modulů. Deklarace funkcí implementovaných moduly se nachází ve stejnojmenných hlavičkových souborech.

3.1 Balík `utilities`

V balíku se nachází moduly obsahující především pomocné funkce pro vyšší vrstvy programu.

3.1.1 Modul `utils`

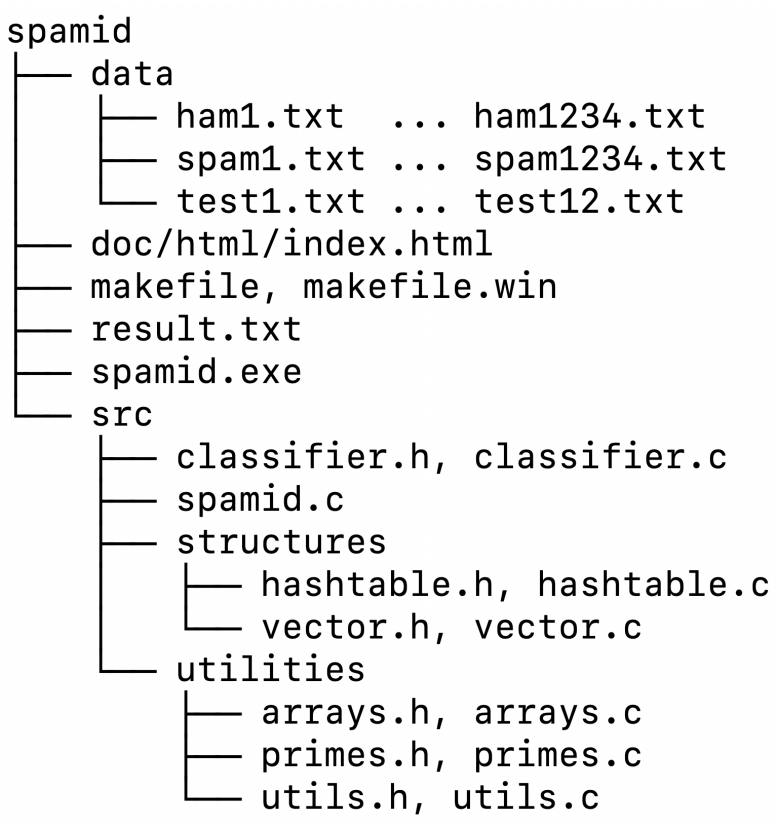
Obsahem modulu je jediná funkce `strdup`, která není standardem ANSI C podporována. Funkce duplikuje řetězec tak, že dynamicky alokuje nový paměťový prostor a do něj řetězec překopíruje.

3.1.2 Modul `primes`

Modul poskytuje funkce pro test prvočíselnosti a získání dalšího prvočísla.

Pro test prvočíselnosti je využíván Millerův-Rabinův test prvočíselnosti. Pro čísla menší než 1 373 653 je zvolena efektivní deterministická varianta testu, pro jiná čísla se jedná o standardní test s $\approx$ 99,998% úspěšností.

Získání dalšího prvočísla probíhá postupným testováním čísel zmíněným testem prvočíselnosti.



Obrázek 3.1: Zjednodušená adresářová struktura projektu s ukázkovými daty

3.1.3 Modul `arrays`

V modulu se nachází funkce pro jednoduchou práci se staticky i dynamicky alokovanými poli.

K dispozici jsou funkce pro dynamické vytvoření pole, vymazání prvků pole, uvolnění paměti pole a získání extremálního prvku pole s využitím deklarovaného komparátoru prvků pole.

3.2 Balík `structures`

Balík obsahuje moduly poskytující funkce pro práci s využívanými datovými strukturami.

3.2.1 Modul `vector`

Obsahem modulu jsou funkce pro práci se staticky či dynamicky založeným dynamickým polem.

K dispozici jsou funkce pro statické či dynamické vytvoření dynamického pole a uvolnění paměti dynamického pole s možností uvolnění i jeho prvků s využitím deklarované funkce pro uvolňení paměti patřící prvku dynamického pole.

Dále obsahuje funkce pro přidání prvku nebo několika prvků, získání ukazatele na prvek na i-té pozici a přenechání dat.

Dalšími funkcemi v modulu jsou funkce pro získání počtu prvků dynamického pole a navýšení nebo ponížení kapacity dynamického pole.

3.2.2 Modul `hashtable`

Modul obsahuje funkce pro práci s dynamicky vytvořenou hash tabulkou a dynamicky vytvořeným iterátorem nad prvky hash tabulky.

Poskytnuty jsou funkce pro dynamické založení hash tabulky a uvolnění paměti hash tabulky s možností uvolnění i jejich prvků (hodnot) s využitím deklarované funkce pro uvolnění paměti patřící prvku hash tabulky.

Dále jsou dostupné funkce pro přidání prvku do hash tabulky, zjištění existence klíče v hash tabulce, získání hodnoty nebo ukazatele na hodnotu přiřazené klíči v hash tabulce a získání počtu prvků v hash tabulce.

Další možnost práce s prvky hash tabulky je pomocí iterátoru. K dispozici jsou funkce pro jeho dynamické vytvoření, uvolnění paměti, návrat na začátek a funkce pro zjištění existence dalšího prvku a získání dalšího prvku.

Implementace hash tabulky

Hash tabulka rozptyluje prvky do pole o velikosti rovné prvočíslu spojových seznamů párů klíče a hodnoty na základě vypočteného hash kódu klíče.

Do hash tabulky se ukládá kopie klíče (řetězce) a kopie hodnoty (ukazatel na libovolný typ nebo přímo libovolná hodnota).

Velikost pole hash tabulky je prvočíselná a dynamická. Pokud nastane situace, že je vkládán nový prvek a počet prvků hash tabulky je roven pětinásobku velikosti pole, velikost pole se zvětší na prvočíslo nejbližší větší nebo rovno dvojnásobku aktuální velikosti pole, dosud vložené prvky jsou v poli přeskládány podle aktuálních hash kódů jejich klíčů a do hash tabulky se vloží nový prvek. Tím je zachována časová složitost vložení prvku v konstantním čase v závislosti na počtu vkládaných prvků.

Prvkem pole hash tabulky je spojový seznam obsahující pár klíč a hodnota vkládaného prvku a ukazatel na další prvek spojového seznamu. V průměrném případě je délka tohoto spojového seznamu nejvýše 5, tedy konstantní. Tím je zachována časová složitost získání prvku v konstantním čase v závislosti na počtu vložených prvků.

Hash kód h klíče s (řetězce délky n složeného z písmen s_0 až s_{n-1}) je vypočten podle vzorce 3.1 efektivně pomocí algoritmu Hornerovo schéma a s využitím vlastností operace `mod` pro zabránění přetečení rozsahu čísla.

$$h = 256s_0 + 256^2 s_1 + ... + 256^n s_{n-1} \mod \textbf{velikost tabulky} \tag{3.1}$$

3.3 Modul `classifier`

Modul obsahuje funkce pro vytvoření klasifikátoru, uvolnění paměti klasifikátoru, natrénování klasifikátoru a klasifikaci datových souborů.

3.3.1 Deklarace a definice

Definice datového typu `nbc` (obecný naivní Bayesův klasifikátor) lze vidět na obr. 3.2, kde `cls_cnt` je počet klasifikačních tříd, `cls_prob` je pole apriorních pravděpodobností výskytu klasifikační třídy c_i v natrénovaných datových souborech $P(c_i)$, `cls_words_cnt` je pole celkových počtů řetězců v natrénovaných datových souborech patřících do klasifikační třídy c_i, `words_cnt` je hash tabulka mající klíč řetězec ⟨word⟩ a hodnotu pole počtů výskytů řetězce ⟨word⟩ v datových souborech patřících do klasifikační třídy c_i, `words_prob` je hash tabulka mající klíč řetězec ⟨word⟩ a hodnotu pole pravděpodobností výskytu řetězce ⟨word⟩ v datovém souboru za podmínky klasifikace datového

Obrázek 3.2: Definice datového typu nbc (obecný naivní Bayesův klasifikátor)

```
typedef struct nbc_ {
    const int cls_cnt;

    double *cls_prob;
    size_t *cls_words_cnt;

    htab *words_cnt;
    htab *words_prob;

    size_t dict_size;
} nbc;
```

souboru do klasifikační třídy c_i $P(\langle \text{word} \rangle | c_i)$ a `dict_size` je celkový počet různých řetězců v natrénované množině datových souborů. Index i představuje index klasifikační třídy a zároveň slouží pro indexaci zmíněných polí.

Funkce pro natrénování klasifikátoru a klasifikaci datového souboru jsou deklarovány následovně:

```
int nbc_learn(nbc *cl, const char *f_paths[], const size_t f_counts[]);
int nbc_classify(const nbc *cl, const char f_path[]);
```

3.3.2 Trénování

Klasifikátor je natrénován podle postupu zmíněném v analýze úlohy (2.2) a podle pseudokódu v zadání práce.

Implementovaný pseudokód algoritmu natrénování klasifikátoru je následovný:

1. Z argumentu `f_counts` vypočítej hodnoty apriorních pravděpodobností výskytu klasifikační třídy c_i v trénovacích datových souborech $P(c_i)$ a hodnoty ulož do pole.

2. Pro každý trénovací datový soubor a klasifikační třídu c_i, do které patří, načítej z datového souboru řetězce $\langle \text{word} \rangle$ a inkrementuj čítač výskytů řetězce $\langle \text{word} \rangle$ v trénovacím datovém souboru patřícím do klasifikační třídy c_i.

3. Z hodnot z přechozího kroku vypočítej celkové počty řetězců vyskytujících se v trénovacích datových souborech patřících do jejich klasifikační třídy a ulož je do pole na index jejich klasifikační třídy.

4. Ulož hodnotu počtu různých slov ⟨word⟩ v trénovací množině datových souborů.

5. Z hodnot čítačů výskytů řetězců v trénovacích datových souborech jejich klasifikační třídy vypočítej pravděpodobnosti výskytu řetězce ⟨word⟩ v datovém souboru za podmínky klasifikace datového souboru do klasifikační třídy c_i $P(\langle \text{word} \rangle | c_i)$ podle vzorce 2.1.

3.3.3 Klasifikace

Klasifikace datového souboru probíhá podle postupu zmíněném v analýze úlohy (2.3) a podle pseudokódu v zadání práce.

Pseudokód implementovaného algoritmu klasifikace datového souboru je následující (v podstatě se jedná jen o implementaci vzorce 2.4):

1. Vytvoř pole pravděpodobností klasifikace testovaného datového souboru do klasifikační třídy c_i a hodnoty inicializuj na zlogaritmovanou hodnotu apriorní pravděpodobnosti výskytu klasifikační třídy c_i v natrénovaných datových souborech $P(c_i)$.

2. Načítej řetězce ⟨word⟩ z testovaného datového souboru a pro každý řetězec ⟨word⟩, pro který existuje vypočtená pravděpodobnost výskytu řetězce ⟨word⟩ v datovém souboru za podmínky klasifikace datového souboru do nějaké klasifikační třídy c_i $P(\langle \text{word} \rangle | c_i)$, přičti $P(\langle \text{word} \rangle | c_i)$ k hodnotě v poli pravděpodobností klasifikace testovaného datového souboru.

3. Index maximální hodnoty v poli pravděpodobností klasifikace testovaného datového souboru je indexem klasifikační třídy, do které testovaný datový soubor patří.

3.4 Modul `spamid`

Hlavní a spouštěný modul programu se stará o celý průběh programu tak, že:

1. ověří vstupní argumenty, případně vypíše manuál,

2. připraví data pro klasifikátor,

3. vytvoří klasifikátor,

4. natrénuje klasifikátor poskytnutou trénovací množinou,

5. klasifikuje poskytnuté testované soubory a
6. výstup programu zapíše do poskytnutého výstupního souboru.

Kapitola 4

Uživatelská příručka

Spuštěním programu se klasifikátor natrénuje na datech z poskytnutých souborů, další soubory následně klasifikuje a nakonec výstup zapíše do výstupního souboru.

Program je multiplatformní a lze přeložit, sestavit a spustit jednotným postupem na všech podporovaných platformách (Win32/64 a Unix).

Všechny příkazy jsou zadávány v kořenovém adresáři projektu a je odkazováno na adresářovou strukturu projektu na obr. 3.1.

4.1 Přeložení a sestavení programu

Překlad zdrojového kódu a sestavení spustitelného programu je řízeno skriptem utility `make`. K překladu a sestavení programu v prostředí Unix dojde po zadání příkazu:

```
spamid$ make
```

resp. příkazu v prostředí Windows:

```
spamid> make -f makefile.win
```

Vykonáním skriptu se v kořenovém adresáři projektu vytvoří spustitelný soubor `spamid.exe`.

4.2 Datové soubory

Datové soubory určené k natrénování klasifikátoru a ke klasifikaci musí být uloženy uvnitř adresáře `data` se závazným pojmenováním a formátem obsahu.

Soubory určené ke trénování jedné klasifikační třídy nebo ke klasifikaci musí mít stejný vzor jména, příponu `.txt`, musí být číslovány od jedné a musí obsahovat jednu řádku mezerou oddělených slov.

4.3 Spuštění programu

Program přijímá právě sedm vstupních argumentů. Vzor příkazu spuštění je:

`spamid.exe ⟨spam⟩ ⟨spam-cnt⟩ ⟨ham⟩ ⟨ham-cnt⟩ ⟨test⟩ ⟨test-cnt⟩ ⟨out-file⟩`

Symboly ⟨`spam`⟩, ⟨`ham`⟩ a ⟨`test`⟩ představují vzory jmen trénovacích a testovaných datových souborů, ⟨`spam-cnt`⟩, ⟨`ham-cnt`⟩ a ⟨`test-cnt`⟩ jejich počty a ⟨`out-file`⟩ představuje jméno výstupního souboru.

Příkaz spuštění programu v prostředí Unix může vypadat následovně:

```
spamid$ ./spamid.exe spam 1234 ham 1234 test 12 result.txt
```

resp. v prostředí Windows:

```
spamid> spamid.exe spam 1234 ham 1234 test 12 result.txt
```

V případě zadání nesprávného počtu nebo neplatných hodnot argumentů program vypíše chybovou hlášku a příručku k ovládání programu. V případě, že dojde k chybě při běhu programu, např. program se pokusí otevřít neexistující soubor nebo dojde k vyčerpání operační paměti, program vypíše chybovou hlášku a je bezpečně ukončen.

4.4 Výstup programu

Výstupem programu je vygenerovaný textový soubor se jménem ⟨`out-file`⟩ předaný programu jako vstupní argument a obsahuje ⟨`test-cnt`⟩ (počet testovaných souborů) řádek.

Každá řádka výstupního souboru má formát:

⟨`test`⟩`[tabulátor][S/H]`

Symbol `S` představuje klasifikaci testovaného souboru ⟨`test`⟩ jako spam, `H` jako ham.

Kapitola 5

Závěr

Navrhnuté a implementované řešení splňuje veškeré požadavky zadání. Program je navrhnut tak, že je přehledný a rozšiřitelný.

Bylo by vhodné problém zezačátku více analyzovat, aby se předešlo nepříliš vhodným programovým řešením jako např. téměř duplicitní hash tabulky klasifikátoru `words_cnt` a `words_prob`, které by jistě mohli být reprezentovány jednou hash tabulkou obsahující jak počty výskytů řetězce, tak i jeho pravděpodobnosti. Z důvodu časového nedostatku nebylo řešení přepracováno.

5.1 Testování

Řešení podstoupilo test[1] natrénování klasifikátoru trénovací množinou datových souborů a klasifikace testovaných datových souborů.

Datová množina obsahuje 380 trénovacích datových souborů třídy spamu ($\approx$ 741 kB, 113 503 slov) a hamu ($\approx$ 767 kB, 103 019 slov) a 100 testovaných datových souborů třídy spamu ($\approx$ 191 kB, 28 799 slov) a hamu ($\approx$ 142 kB, 19 500 slov).

Natrénování klasifikátoru průměrně trvalo 94,56 ms, klasifikace testovaných datových souborů spamu 11,32 ms a hamu 8,47 ms. Hodnoty jsou zatíženy alespoň časem čtení ze souboru.

Klasifikátor správně klasifikoval 100 ze 100 spamů a 93 ze 100 hamů.

[1] Test byl proveden na MacBook Air M1 (2020, 8 GB, macOS Ventura 13.1) a byl 1 000krát opakován nad vzorovou datovou množinou přiloženou k zadání práce: https://www.kiv.zcu.cz/studies/predmety/pc/data/works/sw2022-02-data.zip