FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Identifikace spamu naivním bayesovským klasifikátorem

Semestrální práce KIV/PC

Mikuláš Mach
A21B0202P

5. ledna 2023

Obsah

Kapitola 1

Zadání

Naprogramujte v ANSI C přenositelnou **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 ⟨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 ⟨1; N⟩. 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

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).

Kapitola 2

Analýza úlohy

2.1 Naivní bayesovský klasifikátor

Algoritmus má dvě fáze: **1) Fáze učení** a **2) Fáze klasifikace**.

2.1.1 Fáze učení

Načteme **slova** z **trénovacích souborů** a vytvoříme z nich **slovník**. Při načítání budeme vědět zda aktuálně načtené **slovo** patři do množiny **spam** nebo do množiny **ham**. Ve slovníku si budeme uchovávat u každého slova počet výskytů v každé množině. Následně vypočteme **podmíněnou pravděpodobnost výskytu slova** v **trénovacích souborech** pro množinu **spam** a pro množinu **ham** zvlášť. Využijeme tzv. *bag-of-words*, to znamená že nám nebude záležet na pozici **slov**, ale jenom na jejich výskytu.

2.1.2 Fáze klasifikace

Zvlášť **počítáme pravděpodobnost** toho zda je **testovaný soubor spam** a zvlášť pravděpodobnost toho že je soubor **ham**. Ze souboru načteme **slovo**. Koukneme se jestli se **slovo** nachází ve **slovníku**, pokud ano, tak přičteme **logaritmus** jeho **podmíněné pravděpodobnosti výskytu** v dané množině k danému **výpočtu pravděpodobnosti**. Tento proces budeme opakovat dokud nenačteme všechny **slova** z **testovaného souboru**. Na konci dostaneme dvě čísla. Jedno udává jaká je **pravděpodobnost**, že **testovaný soubor** je **spam** a druhé udává pravděpodobnost toho že **soubor** je **ham**. Výpočet je popsán rovnicí 2.1 na straně 4.

$$c = \arg\max_{c_i \in C} \left(\sum_{k \in \textbf{pozice}} \log(P(<word_k>|c_i)) \right) \tag{2.1}$$

- C - Množina obsahující dvě podmnožiny (spam, ham)
- c_i - Konkrétní prvek množiny
- arg max - Pro všechny hodnoty c_i, vyčíslí hodnotu argumentu a vrátí c_i, pro kterou byla vypočtená hodnota nejvyšší
- $P(<word_k>|c_i)$ – podmíněná pravděpodobnost výskytu slova

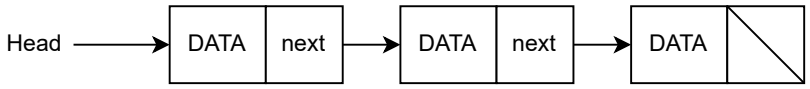
2.2 Datová struktura

Pro vytvoření **slovníku** bude nutné implementovat vhodnou **datovou strukturu**. **Struktura** by měla mít nízkou **časovou složitost** u vkládání a hledání prvku.

2.2.1 Spojový seznam

Na rozdíl od **pole** nebudeme muset u **spojového seznamu** řešit jeho velikost, protože se dá jednoduše rozšířit. **Spojový seznam** má **ukazatel** na jeho počáteční prvek tzv. *head*, ten dále **ukazuje** právě na jeden další **prvek**. Všechny prvky jsou takhle **zřetězeny**. **Koncový prvek** má ukazatel nastavený na **NULL**. Spojový seznam je ukázán na obrázku 2.1 na straně 4.

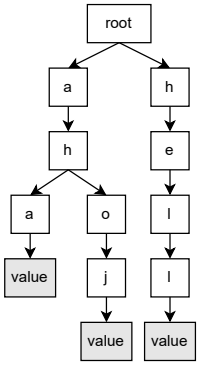
Operace vložení má **časovou náročnost** O(1), pouze pokud si uložíme ukazatel i na **poslední prvek**, ale **operace nalezení prvku** má časovou složitost O(n), tudíž je tato datová struktura **nevhodná** pro naší úlohu, protože **hledání prvku** bude často využíváno.



Obrázek 2.1: Spojový seznam

2.2.2 Trie

Trie je *"stromová"* struktura pro hledání specifického **klíče**. Někdy se taky nazývá jako **prefixový strom**. **Klíč** je do **trie** vložen po znacích jako posloupnost **uzlů**, tzn. že **vložení** má **časovou náročnost** O(k), kde k je **počet písmen** vkládaného klíče. Stejná časová složitost platí i pro **vyhledávání**. Na obrázku 2.2 na straně 5 je ukázána **trie**, do které byla **vložena** slova: *aha, ahoj, hell*.

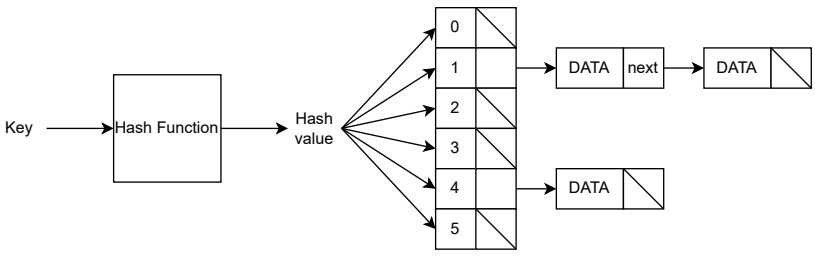


Obrázek 2.2: Trie

2.2.3 Rozptýlená tabulka

Jedná se o **datovou strukturu**, která je postavena nad polem. Na rozdíl od normálního pole využívá *hashovací funkci*. Díky této vlastnosti by **nalezení** i **vložení** daného **prvku** pomocí **klíče** mělo **průměrně** trvat O(1) a v **nejhorším případě** O(n).

V této datové struktuře může dojít ke **kolizi**, tzn. že při vložení různých **klíčů** do **hashovací funkce** dostaneme stejný výsledek. Tento problém by se řešil pomocí přidání **spojového seznamu** na každý index **tabulky**. Tím pádem nemůže dojít místo v **tabulce**. Pro zmenšení pravděpodobnosti **kolize** musí být **tabulka** dostatečně veliká a **hashovací funkce** musí dávat dostatečně **rozptýlené** hodnoty, ale zároveň nesmí být složitá, aby **nezpomalovala** program. Datová struktura je ukázána na obrázku 2.3 na stráně 5.



Obrázek 2.3: Rozptýlená tabulka

2.2.4 Rozhodnutí

Slovník bude implementován pomocí **rozptýlené tabulky**, protože za použití dobré **hashovací funkce** dosáhneme dobré **časové náročnosti**, což v průměru je O(1) pro **vkládání** i pro **vyhledávání**.

Kapitola 3

Implementace

Program je **rozdělen** na několik částí, konkrétně na: **Hashtable**, **Input**, **Bayes**, **Config** a **Main**. V části **Config** jsou pouze definice a v části **Main** jsou volány všechny důležité **funkce**.

3.1 Hashtable

Tabulka je využita jako **slovník**. Jsou v ní **uloženy** všechny hodnoty důležité pro **klasifikaci**. Tabulka obsahuje dvě **struktury**: **hashtable** a **item**.

3.1.1 Struktura item

Jedná se o **strukturu**, ve které jsou uložena data pro jedno konkrétní **slovo**. Tato **struktura** má představovat jednu položku v **tabulce**. Položka obsahuje svůj **klíč**, který je implementovaný jako řetězec charakterů, dále **obsahuje** obsahuje počet výskytů ve všech **spam souborech** a **podmíněnou pravděpodobnost výskytu**. **Struktura** má v sobě uloženy stejné **hodnoty** i pro všechny **ham soubory**. Poslední **uložená** věc ve **struktuře** je ukazatel na další **položku**, aby bylo možné **položky** zřetězit při **kolizi**.

3.1.2 Struktura hashtable

Jednoduchá **struktura**, která obsahuje **velikost tabulky**, počet vložení **slov typu spam**, počet vložení **slov typu ham** a pole **ukazatelů** typu **item**.

3.1.3 Funkce tabulky

item_create()

Alokuje paměť potřebnou pro **strukturu** a vloží do ní **klíč**. Jako **parametry** potřebuje **klíč** a číslo **0** (pokud **klíč** je ze **spam souboru**) nebo **1** (pokud klíč je z **ham souboru**).

hashtable_create()

Alokuje paměť potřebnou pro **strukturu**, vloží do ní velikost tabulky a **alokuje** paměť pro **pole ukazatelů** typu **item**.

hash()

Jako **parametry** přijímá **řetězech** a ukazatel na **tabulku** a vrací **index** v dané **tabulce**.

hashtable_find()

Pomocí funkce **hash()** vypočte **index**, na kterém bude **vyhledávat**. Dokud nenarazí na **NULL**, tak bude **procházet** spojový seznam, který se nachází na daném **indexu**. Pokud funkce daný **item** nenajde, tak vrátí **NULL**, když ho najde, tak vrátí **ukazatel** na daný **item**.

add_count()

Jako **parametry** přijímá **ukazatel na položku** a číslo **0** nebo **1**. Funkce pouze přičte jedničku buď k počtu výskytu dané položky ve **spam souborech** nebo **ham souborech** podle **číselného parametru**. Pokud byl parametr **0**, tak přičte k výskytu ve **spam souborech** a pokud **1**, tak k výskytu v **ham souborech**.

hashtable_insert()

Funkce má jako parametry **ukazatel na tabulku**, **klíč** a číslo **0** (jedná se o **spam**) nebo **1** (jedná se o **ham**). Napřed pomocí funkce **hashtable_find()** zjistí, jestli se prvek už nachází v tabulce, pokud ano, tak zavolá funkci **add_count()** a funkce končí, pokud položka **nebyla nalezena**, tak zavolá funkci **item_create()**. Následně pomocí funkce **hash()** vyhledá **index**, na který má **položku** vložit a projde spojový seznam, dokud nenarazí na *"volné místo"*, kam následně **vytvořenou položku** vloží. Funkce vrátí číslo **1**, pokud vše proběhlo správně.

3.2 Input

Tato část programu se stará o **načítání trénovacích dat**.

3.2.1 Funkce

read_word()

Jako parametr přijímá **ukazatel** na **otevřený soubor**.

Funkce si **alokuje paměť** pro řetězec o předen **definované** velikosti. Následně funkce bude postupně číst znaky ze souboru a bude je vkládat do **řetězce**. Skončí až když narazí na **znak mezery** nebo na **konec souboru**. Po dokončení čtení se **alokuje** nový **řetězec**, který má stejnou velikost jako **počet načtených znaků** + 1 (**pro koncový znak**). Původní řetězec se **uvolní** a nově vytvořený **řetězec** funkce **vrátí jako výsledek**.

load()

Parametry funkce jsou: **ukazatel na tabulku**, **vzor názvu načínaných souborů** a **počet souborů**.

Funkce si **alokuje paměť** pro řetězec. Ve **smyčce** postupně **vytváří názvy trénovacích souborů**, pomocí **vzoru** a **počtu**. V této **smyčce** se nachází další **smyčka**, ve které se volá funkce **read_word()**, dokud se nenarazí na konec souboru. Po načtení slova je **slovo vloženo** do **tabulky** pomocí **hashtable_insert()**.

3.3 Bayes

V této části se provádí **klasifikace** testovaných souborů a všechny potřebné **výpočty**.

3.3.1 Funkce

count()

Jednoduchá funkce, která přijme jako parametr **ukazatel na tabulku**. Projde přes všechny **prvky** v tabulce a **aktualizuje** počet vložených **slov typu ham** a počet vložených **slov typu spam**.

probabilities()

Jednoduchá funkce, která přijme jako parametr **ukazatel na tabulku**. Projde přes všechny **prvky** v tabulce a u každého z nich **vypočte** jejich **podmíněnou pravděpodobnost výskytu** v množině **spam** a v množině **ham**.

bayes_one_file()

Jako parametry přijímá: **Ukazatel na tabulku**, **řetězec s názvem testovaného souboru**, **ukazatel na otevřený výstupní soubor**.

Tato funkce **klasifikuje** vždy jeden soubor. Uchovává si vždy **pravděpodobnost** toho, že soubor je **spam** a **pravděpodobnost** toho, že je **ham**. Funkce **otevře testovaný soubor** a bude ve smyčce pomocí funkce **read_word()** číst soubor slovo po slově. Když načte slovo, tak se koukne do tabulky, když se tam nachází, tak přičte **zlogaritmovanou** danou **podmíněnou pravděpodobnost výskytu** k dané **uchovávané hodnotě**. Po průchodu celého souboru se **porovnají uchovávané hodnoty** a **klasifikujeme** soubor podle toho, která z nich je **větší**. **Výsledek** se **zapíše** do otevřeného **výstupního souboru**.

bayes()

Parametry funkce jsou: **Ukazatel na tabulku**, **řetězec vzoru názvu testovaných souborů**, **počet testovaných souborů**, **řetězec názvu výstupního souboru**.

Funkce **alokuje paměť** pro řetězec a poté jsou zavolány funkce **count()** a **probabilities()**, aby bylo vše připraveno pro **klasifikaci**. Následuje smyčka, ve které jsou *"skládány"* řetězce názvů **testovaných souborů**. Po každém *"poskládání"* se zavolá funkce **bayes_one_file()**. Takto jsou postupně **klasifikovány** všechny souboru určené pro **testování**.

Kapitola 4

Uživatelská příručka

4.1 Přeložení programu

Pro překlad je využit **makefile**. Pro jeho funkci je nutné mít kompilátor **gcc** nebo **mingw**. Spouští se přes **terminál/příkazovou řádku**. Pro jeho spuštění musíme být ve stejném adresáři a použít příkaz **makefile** (pro gcc kompilátor) nebo příkaz **mingw32-make** (pro mingw kompilátor).

4.2 Spuštění programu

Po přeložení můžeme program spustit tímto příkazem:

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

- ⟨spam⟩, ⟨ham⟩ - Vzor názvu trénovacích spam a ham souborů
- ⟨spam-cnt⟩, ⟨ham-cnt⟩ - Počet trénovacích souborů
- ⟨test⟩ - Vzor názvu testovaných souborů
- ⟨test-cnt⟩ – Počet testovaných souborů
- ⟨out-file⟩ - Celý název výstupního souboru

Soubory pro trénování a testování se musí nacházet v **adresáři data**. Tento adresář musí být na stejné úrovni jako spouštěný program **spamid.exe**

Výstupní soubor se vytvoří na stejné úrovni jako **spamid.exe**. Ve výstupním souboru je na každé řádce napsán název testovaného souboru a vedle toho je napsáno jak byl soubor **klasifikován**.

Příklad příkazu pro spuštění:

spamid.exe spam 100 ham 100 test 10 result.txt

Kapitola 5

Závěr

Vytvořený program **splňuje** všechny body zadání. Klasifikace proběhne dostatečně rychle, přesnost klasifikace je vyšší než 90% a program uvolňuje všechnu alokovanou paměť. **Kompilátor** při kompilaci neoznamuje žádné chyby.

Program by mohl být navržen více **abstraktněji**. Například **hashtable**, tak jak je navržen teď, tak by s ničím jiným než s tímto zadáním nefungoval. Dále by šlo určitě dosáhnout rychlejšího běhu programu.

Při klasifikaci jsem narazil na problém se vzorcem, který byl poskytnut v zadání. Po odstranění násobení **prior pravděpodobností**, klasifikace proběhla s dostatečnou přesností. Kromě tohoto problému jsem během implementace na žádný jiný problém nenarazil a všechno proběhlo v pořádku.