ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačov

Bakalářská práce

Adaptácia používateľského rozhrania založená na analýze textových súborov

Michal Sivoň

Máj 2019

Poděkování / Prohlášení

Chcel by som sa poďakovať pánovi Ing. Jiřímu Šebkovi za vedenie bakalárskej práce, cenné rady a príjemnú spoluprácu.

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o etickej príprave záverečných prací. Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb. a autorského práva vo znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohto zákona týmto udeľujem nevýhradné oprávnenie (licenciu) k použitiu tejto práce a to vrátane všetkých počítačových programov, ktoré sú súčasťou či prílohou tejto práce a všetkej dokumentácie k tejto práci (ďalej len Dielo) a to všetkým osobám, ktoré si prajú Dielo použiť. Tieto osoby sú oprávnené Dielo použiť akýmkoľvek spôsobom, ktorý nesnižuje hodnotu Diela a za akýmkoľvek účelom (vrátane použitia k zárobkovým účelom). Toto oprávnenie je časovo, teritoriálne i množstvom neobmedzené. Každá osoba ktorá využije vyššie uvedenú licenciu sa ale zaväzuje udeliť ku každému dielu, ktoré vznikne (hoc i len čiastočne) na základe Diela, úpravou Diela, spojením Diela s iným dielom, zaradením Diela do diela súborného či spracovaním Diela (vrátane prekladu), licenciu aspoň vo výške uvedeného rozsahu a zároveň sprístupní zdrojový kód takéhoto diela aspoň zrovnateľným spôsobom a v zrovnateľnom rozsahu ako je sprístupnený zdrojový kód Diela.

Abstrakt / Abstract

Táto bakalárska práca sa zaoberá úlohou vytvorenia a otestovania frameworku na adaptáciu používateľského rozhrania aplikácie na základe analýzy a klasifikovania textových súborov. Tento framework umožní vybratie položiek používateľského rozhrania, ktoré používateľ s väčšou pravdepodobnosťou použije vzhľadom na textový súbor, s ktorým používateľ pracuje. Na klasifikáciu súborov bol použitý SVM a bayesovský algoritmus. Demonštračná aplikácia bola vyvinutá pomocou existujúcej práce M. Mishchenka a otestovaná pomocou keystroke level modelu. Testovanie ukázalo zlepšenie času prístupu k často používaným položkám rozhrania, avšak prístup k nepoužívaným položkám sa zhoršil.

Kľúčové slová:
adaptívna štruktúra, textová klasifikácia, analýza textu, adaptácia používateľského rozhrania

The goal of this bachelor thesis is to create and test a framework for adaptation of user interfaces based on analysis of and classification of text files. This framework will allow for selection of user interface items based on calculating probabilities of their usage from text the user works on. We used SVM algorithm and Bayesian classification. Demonstrative application was developed with existing work M. Mishchenko and tested with keystroke level model. Testing showed improvement of access time to frequently used elements, but made access time to unused elements worse.

Keywords:
adaptive structure, text classification, text analysis, user interface adaptation

Title translation:
Adaptation of user interface based on analysis of textual files

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

1.1 Motivácia

Existuje mnoho programov v ktorých používateľ pracuje s textovými súbormi. Najčastejšími príkladmi sú textové procesory, e-mailové klienty, zobrazovače pdf a podobne. Tieto programy majú často komplikované UI s veľkým množstvom rôznych položiek v používateľskom rozhraní. Pri používaní týchto aplikácií musí používateľ často vykonať množstvo krokov aby sa dostal k položkám používateľského rozhrania, ktoré chce použiť. Musí prejsť cez rôzne menu, podmenu vyskakovacie okná a podobné navigačné štruktúry. Toto môže spomaľovať prácu s aplikáciou.
Pri práci s textovými súbormi sa spôsob použitia aplikácie a najčastejšie používané položky používateľského rozhrania môžu meniť v závislosti na otvorenom súbore. Pri e-mailovom kliente používateľ pracuje inak s pracovným e-mailom, e-mailom od priateľa, elektronickými letákmi, výpismi z bankového účtu, faktúrami a podobne. Používateľ napríklad presunie faktúry a bankové výpisy do špeciálneho priečinka, pridá elektronický podpis alebo komentáre ku dokumentu z práce, použije emoji pri práci s e-mailom od priateľa a podobne. Najčastejšie používané položky menu pdf čítača sa určite menia v závislosti na na otvorenom pdf dokumente. Pri práci s elektronickým formulárom bude používateľ častejšie klikať na položky pre vpisovanie textu do dokumentu a pridávanie a elektronických podpisov. Pri práci s elektronickými skriptami bude s väčšou pravdepodobnosťou požívať nástroje na kreslenie v dokumente, zvýrazňovač a podobne.

1.2 Cieľ práce

Cieľom tejto práce je napísať framework, ktorý pomocou techník analýzy obsahu textu vyberie položky používateľského rozhrania aplikácie, ktoré používateľ s najväčšou pravdepodobnosťou použije pri práci s práve otvoreným textovým dokumentom. Následne framework integrujem do už existujúcej bakalárskej práce Nikitu Mishchenka [1] zaoberajúcou sa adaptáciou menu aplikácie. Takto vzniknutú aplikáciu otestujem.

Kapitola 2
Rešerše

V tejto kapitole sú vykonané rešerše informácií potrebných k návrhu frameworku.

2.1 Adaptívne aplikácie

Adaptívna aplikácia je aplikácia, ktorá sa prispôsobuje potrebám konkrétneho používateľa a robí mu použitie aplikácie jednoduchšie. Prispôsobenie prebieha vzhľadom na informácie, ktoré aplikácia získala o tomto konkrétnom používateľovi z predchádzajúcich interakcií používateľa s aplikáciou. [2]. Pre adoptovanie používateľského rozhrania pre konkrétneho používateľa je vhodné vytvoriť model používateľa [3]. K tomu sa dajú použiť techniky strojového učenia. [4].

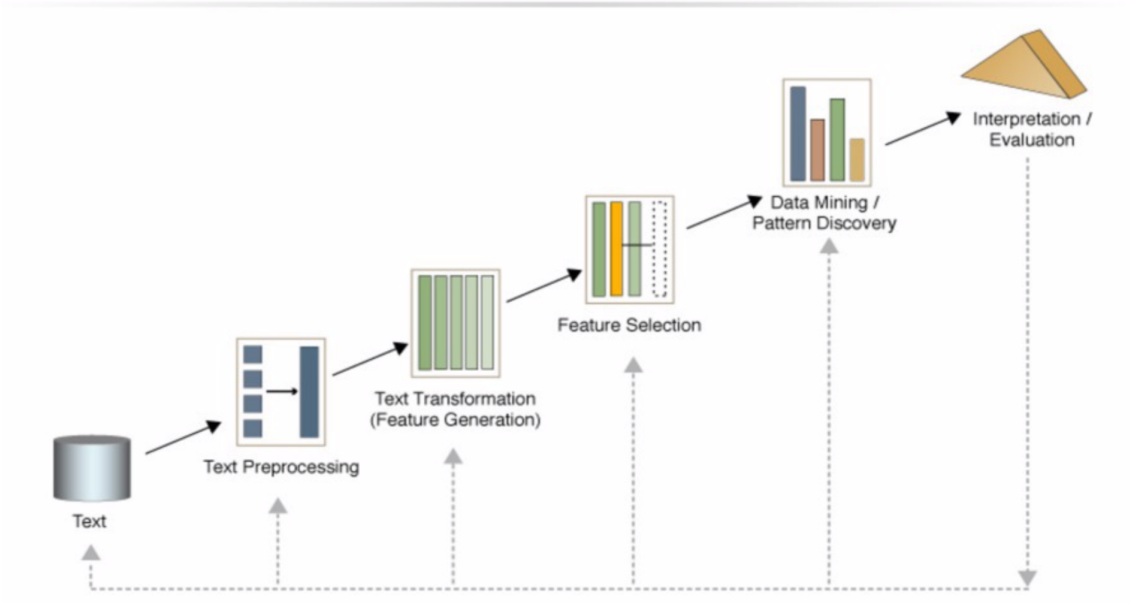
2.2 Text Mining

Text mining je podkategóriou strojového učenia. "Účelom text miningu je extraktovať užitočnú informáciu z textových dátových zdrojov pomocou identifikácie a štúdia zaujímavých vzorov." [5]. Dátovými zdrojmi v text miningu sú neštrukturované textové dokumenty. Text mining používa mnoho techník a algoritmov vyvinutých pre iné typy strojového učenia. Zásadný rozdiel medzi strojovým učením a text miningom je nutnosť predpripraviť neštruktúrované textové dáta, pred použitím učiaceho algoritmu. Proces text miningu je znázornený na obrázku 2.1.

2.2.1 Korpus

Text mining pracuje s množinami dokumentov. Množina dokumentov sa zvykne nazývať korpus. V praxi väčšina korpusov používaná v text miningu obsahuje veľké množstvá dokumentov od mnoho tisíc až po milióny individuálnych neštruktúrovaných dokumentov [5]. Existuje mnoho verejne prístupných korpusov k dispozícií na internete. Tieto korpusy vznikli zbieraním veľkého množstva textu ako príspevky na diskusných fórach, novinové články, abstrakty výskumných papierov, nevyžiadaná elektronická pošta a podobne. Ku korpusom bývajú vo väčšine prípadov pridávané nejaké dodatočné dáta ako napríklad rozdelenie dokumentov do kategórií. Uvádzam niekoľko málo príkladov korpusov:

- Reuters-21578 - kolekcia 21578 správ zo spravodajskej siete Routers z roku 1987 roztriedená podľa kategórií správ. [6]
- Twenty Newsgroups - kolekcia 20000 príspevkov z diskusného fóra rozdelená podľa kategórií na fórach. [7]
- TREC 2005 Spam Track - kolekcia 226293 e-mailov kategorizovaných ako nevyžiadaná pošta alebo nezávadná pošta. [8]



Obrázek 2.1. Proces text mining-u [10].

2.2.2 Predpripravenie dokumentu

Predpripravenie dokumentu vytvorí z neštruktúrovaného textu reprezentáciu s definovanou štruktúrou, s ktorou je potom možné ďalej pracovať. Dokument je reprezentovaný ako množina vlastností (features). Úlohou predpripravenia dokumentu je potom vytvoriť k danému dokumentu množinu vlastností. V ďalšej práci s dokumentom je potom tento dokument reprezentovaný už iba ako táto množina vlastností. Každá vlastnosť sa dá chápať ako jedna dimenzia v priestore. Veľkosť dimenzie býva určená množstvom výskytov vlastnosti v konkrétnom dokumente (napr. ak dokument obsahuje 3 krát vlastnosť x na dimenzii vlastnosti x bude mať hodnotu 3). Vlastností dokumentu môže byť veľké množstvo a vo výsledku sa dokument namapuje do priestoru s veľkým množstvom dimenzií [5].

2.2.3 Vlastnosti dokumentu (document features)

Snažíme sa vybalansovať efektivitu analýzy a množstvo použitých vlastností. Typicky používané vlastnosti sú slová vybrané z dokumentu. Môžme použiť každé unikátne slovo v dokumente. To ale vedie k veľmi veľkému množstvu vlastností. Množstvo vlastností je možné znížiť vybraním podmnožiny slov, ktoré poskytujú najväčšiu informáciu o dokumente a to často s minimálnym negatívnym alebo dokonca aj pozitívnym dopadom na efektivitu analýzy [9]. Slová je ďalej možné redukovať na ich koreň. Ďalšou často používanou optimalizáciou je odstraňovanie najčastejšie používaných slov v jazyku z dokumentu pomocou slovného filtru [5]. Slová nie sú jediné možné vlastnosti dokumentu. Je možné vyhľadávať preddefinované frázy alebo zisťovať prítomnosť konceptov v dokumente. Konceptom môže byť napríklad “mobilný telefón”. Dokument nemusí obsahovať slová “mobilný” a “telefón” na to aby obsahoval koncept mobilný telefón [5].

2.2.4 Klasifikácia textových dokumentov

Klasifikácia textových dokumentov spadá pod použitia text miningu. Klasifikácia pracuje s korpusom dokumentov, v ktorom je každý dokument označený ako patriaci do nejakej (kľudne i viac) triedy. Úlohou klasifikácie je vytvoriť model z tohto korpusu tak,

že tento model je možné neskôr použiť na klasifikovanie nejakého neznámeho textového dokumentu, to jest priradiť mu niektorú zo známych tried dokumentov [10].

2.2.5 Naivný Bayes

Naivný bayes je technika často používaná v klasifikácií textových dokumentov. Ako vyplýva z názvu je založená na Bayesovej vete:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \tag{1}$$

kde v našom prípade "c" je trieda dokumentu a "d" je dokument. Pre nejaký neznámy dokument d chceme vypočítať $P(c_i|d)$ pre každu triedu c_i. Ako výsledok vezmeme tú triedu do ktorej dokument patrí s najväčšou pravdepodobnosťou. Menovateľa $P(d)$ môžme vypustiť. Za $P(c)$ vezmeme:

$$P(c) = \frac{N_c}{N_{doc}} \tag{2}$$

kde N_c je množstvo dokumentu s kategóriou c a N_{doc} je množstvo všetkých dokumentov.

$$P(d|c) = P(x_1, x_2, x_3, ..., x_n|c) \tag{3}$$

kde $x_1, x_2, x_3, ..., x_n$ sú vlastnosti dokumentu. Potom môžme spraviť :

$$P(x_1, x_2, x_3, ..., x_n|c) = P(x_1|c) \cdot P(x_2|c) \cdot P(x_3|c) \cdot ... \cdot P(x_n|c) \tag{4}$$

Toto ale predpokladá, že $x_1, x_2, x_3, ..., x_n$ majú navzájom nezávislé pravdepodobnosti. Pravdepodobnosti slov v texte na sebe navzájom nezávislé nie sú. Slová nasledujú po určitých slovách s väčšou pravdepodobnosťou ako po iných slovách. Vzájomná nezávislosť vlastností je ale podmienkou na použitie tohto klasifikátoru. To je dôvodom prečo sa tento klasifikátor nazýva naivným (často aj idiot's Bayes). V praxi to nepredstavuje veľký problém, pretože aj napriek nesplneniu predpokladov pre jeho použitie v prakticky akomkoľvek textovom súbore, naivný Bayes dosahuje pôsobivé výsledky a je jedným z najlepších klasifikátorov.
Zostáva vyriešiť podmienenú pravdepodobnosť vlastnosti v triede dokumentu. Tu vypočítame ako:

$$P(x_i|c) = \frac{count(x_i, c)}{\sum_{x \in X} count(x, c)} \tag{5}$$

čili množstvo výskytov vlastnosti x_i v triede c vydelené množstvom výskytov všetkých vlastností v tej istej triede c.
Tu môže ešte nastať problém s delením nulou. To sa vyrieši pridaním konštantného člena takto:

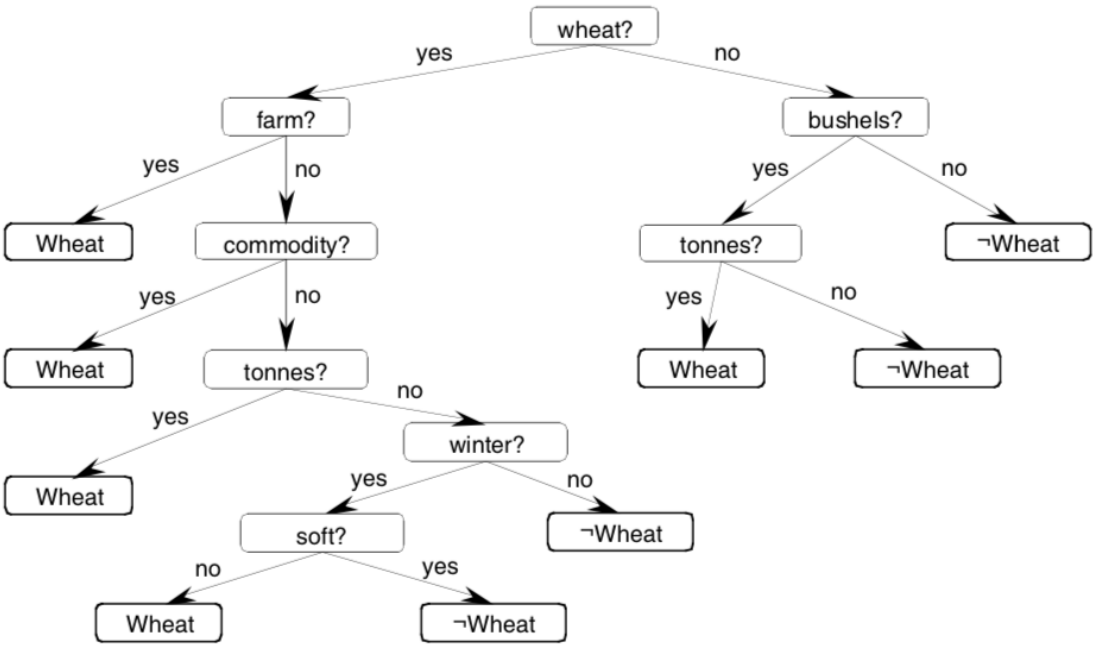
$$P(x_i|c) = \frac{count(x_i, c) + 1}{\sum_{x \in X} count(x, c) + |V|} \tag{6}$$

kde $|V|$ je množstvo všetkých unikátnych vlastností známych klasifikátoru [11].

2.2.6 K-Nearest Neighbor (kNN)

KNN klasifikátor pri klasifikovaní nejakého neznámeho dokumentu vracia binárnu odpoveď patrenia alebo nepatrenia do zvolenej kategórie. Pri použití KNN klasifikátoru sa jednotlivé dokumenty v korpuse namapujú do priestoru o n dimenziách kde n je počet všetkých features v korpuse. Klasifikátor si tieto dokumenty zapamätá ako body v priestore. Dokumenty z korpusu sa následne použijú ako takzvaný príklad. Pri zisťovaní či dokument d patrí do kategórie c sa vypočíta k najbližších dokumentov z trénovacieho korpusu ku dokumentu d. Pod vzdialenosťou sa tu myslí Euklidovská vzdialenosť v n dimenzionálnom priestore. Pokiaľ dostatočné množstvo najbližších dokumentov je z kategórie c klasifikátor vráti kladnú odpoveď inak zápornú.
Pri použití knn klasifikátoru je nutné vybrať hodnotu k. [19] doporučuje k medzi 35 a 45. Jedná sa o jeden z najpresnejších klasifikátorov avšak má veľmi vysokú výpočetnú náročnosť pretože pri klasifikovaní dokumentu je nutné vypočítať Euklidovskú vzdialenosť od všetkých dokumentov z trénovacieho korpusu. [5]

2.2.7 Klasifikačné stromy

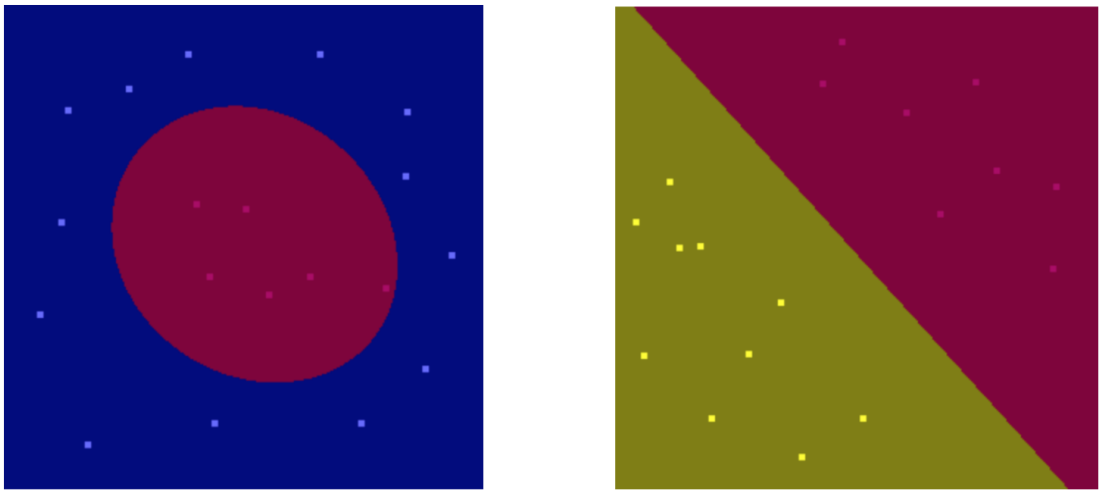


Obrázek 2.2. Binárny klasifikačný strom [5].

Klasifikačné stromy sú založené na stromoch, v ktorých uzly sú jednotlivé features, hrany vedúce z týchto uzlov predstavujú testy na dokumente týkajúce sa feature v uzle a listy stromu sú kategórie dokumentov. Pri klasifikovaní dokumentu klasifikátor začne v korene stromu a postupuje ním podľa vyhodnocovania podmienok na dokumente pokiaľ nedôjde do nejakého listu, ktorý predstavuje kategóriu. Väčšina klasifikačných stromov používa binárne podmienky a teda sa vo väčšine prípadov jedná o binárne stromy. Výhodou klasifikačný stromov je to, že im je po predtrénovaní jednoduché porozumieť narozdiel od iných klasifikátorov. Ich efektivita je ale zmiešaná a iné klasifikátory podávajú lepší výkon. Klasifikačné stromy sú väčšinou generované automaticky [5]. Na obrázku 2.2. je ukážka binárneho klasifikačného stromu.

2.2.8 Support Vector Machine (SVM)

SVM algoritmus je veľmi rýchly a efektívny algoritmus na klasifikáciu dát. Je možné použiť ho prakticky na akýkoľvek typ dát vrátane textových súborov [12]. SVM pracuje len s dvomi triedami dokumentov a dáva binárnu odpoveď na klasifikáciu dokumentu. Jednotlivé dokumenty sa namapujú do n-dimenzionálneho priestoru, kde n je počet všetkých vlastností dokumentu v celom korpuse. Algoritmus následne vypočíta nadrovinu, tak že táto nadrovina rozdeľuje dve triedy dokumentov (rozdeľuje priestor dokumentov na dva polpriestory) s čo najväčším súčtom vzdialeností jednotlivých dokumentov od tejto nadroviny.
Pri klasifikovaní neznámeho dokumentu sa tento dokument namapuje do priestoru dokumentov a spočíta sa v ktorom polpriestore sa nachádza. Ako odpoveď sa zvolí trieda príslušiaca tomuto polpriestoru [5].
Rozdeľujúca nadrovina je definovaná tak zvanou kernelovou funkciou. V niektorých prípadoch nie je možné rozdeliť dve triedy dokumentov. V týchto prípadoch sa priestor dokumentov môže namapovať do priestoru vyššej dimenzie a na definíciu rozdeľujúcej nadroviny sa použije nelineárna kernelová funkcia [5]. Existuje mnoho SVM verzií, ktoré používajú rôzne kernelové funkcie pre rozdeľujúcu nadrovinu. Pri klasifikovaní dát je vhodné zistiť, ktorá kernelová funkcia prinesie najlepšie výsledky s klasifikovaným typom dát [12]. Obrázok 2.3 znázorňuje SVM v 2 dimenzionálnom priestore.



Obrázek 2.3. SVM v 2 dimenziách s nelineárnym kernelom (vľavo) a linárnym kernelom (vpravo) [13].

Kapitola 3
Analýza a návrh

Táto kapitola sa zaoberá analýzou, požiadavkami na návrh frameworku, príkladmi použitia frameworku a návrhom frameworku. Návrh je načrtnutý v podkapitole "Konceptuálny model" a vysvetlený bližšie v podkapitole "Detailná štruktúra frameworku". Taktiež popisuje implementáciu algoritmov.

3.1 Požiadavky

Framework budem implementovať v programovacom jazyku Java 1.8. Hlavným faktorom pri výbere jazyka boli už existujúce práce na ČVUT, ktoré sú takmer všetky napísané v Jave, obzvlášť práca Nikitu Mishchenka, do ktorej integrujem tento framework.
Framework by mal byť jednoducho použiteľný, zapuzdrený vo svojom vlastnom balíčku, ktorý je jednoduché pridať do už existujúcich aplikácií. Framework by mal podporovať tieto operácie:

- Zisťovanie pravdepodobností jednotlivých UI elementov pre rôzne otvorené dokumenty.
- Pridávanie UI elementov do frameworku.
- Predtrénovanie frameworku.
- Trénovanie frameworku postupne počas používania aplikácie používateľom.
- Ukladanie modelu používateľa na rôzne typy úložísk.
- Trénovanie a analýza súborov by nemali byť výpočetne príliš náročné.
- Jednoduchá rozšíriteľnosť a pridávanie ďalších funkcií/algoritmov.

3.2 Príklady použitia

Typické použitie frameworku by vyzeralo takto: Developér pridá framework do svojej aplikácie. Každý adaptovaný UI element je pridaný do frameworku. Developér môže a nemusí predtrénovať framework pre svoju aplikáciu. V prípade, že sa ho predtrénovať rozhodne, predtrénuje framework na korpuse dokumentov, pričom ku každému dokumentu poskytne najviac používané elementy UI pri práci s týmto dokumentom. Takto predtrénovaný framework by poskytol "defaultný" model používateľa.
Počas behu aplikácie zakaždým keď používateľ otvorí súbor framework poskytne najpravdepodobnejšie elementy UI k tomuto dokumentu. Taktiež ak používateľ klikne na element UI pri práci s dokumentom aplikácia o tom informuje framework a ten aktualizuje model používateľa.

Nasledujú dva jednoduché príklady možného využitia tohto frameworku.

3.2.1 Pdf reader

Obrázek 3.1. Pdf reader na iOS 12.1. Možnosť zvýrazňovať je v menu 1. stupňa, možnosť podpisovať je v menu 2. stupňa.

Obrázek 3.2. Pdf reader s navrhnutým UI prvkom.

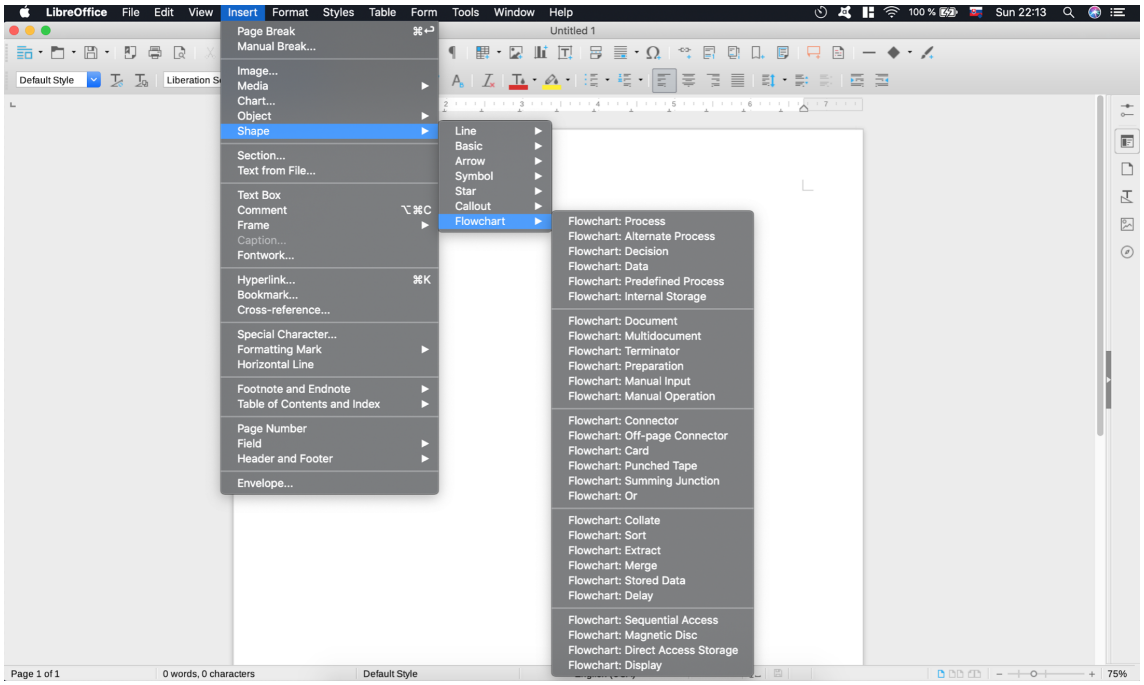
Framework by bolo možné napríklad použiť na vybratie nástroja pre prácu s pdf na základe otvoreného pdf. Na mobilnom telefóne je obmedzené miesto na obrazovke a preto v mobilných pdf zobrazovačoch sú všetky funkcie schované v menu. Týmto frameworkom by bolo možné zobraziť jeden nástroj, ktorý bude použitý s najväčšou pravdepodobnosťou. Vyberanie UI prvku by prebehlo následujúcim spôsobom:

1. Používateľ otvorí súbor.

2. Program analyzuje obsah pdf.
3. Program ponúkne možnosti napríklad:
   - Zistí že sa jedná o pdf skriptá - zobrazí podčiarkovací nástroj alebo poznámkovací nástroj.
   - Zistí že to je oficiálny dokument - zobrazí stamping tool alebo signing tool.

3.2.2 Aplikácia s položkami UI v menu, ktorá pracuje s textovými súbormi.

Niektoré aplikácie pre desktop počítače majú veľmi komplikovane štrukturované menu, s veľkým množstvom položiek a podmenu. Framework by bolo možné použiť na asociovanie položiek v menu s typmi dokumentov a automatické vyberanie tlačidiel pre práve otvorený dokument. Bolo by možné napríklad zmeniť poradie položiek v menu tak aby najčastejšie používané položky boli navrchu alebo vytvoriť špeciálnu, malú lištu s navrhnutými tlačidlami.



Obrázek 3.3. Program s množstvom položiek v menu (libre office na macOs).

Program by sa učil vždy keď používateľ klikne na nejakú položku v menu.

Učenie sa:

1. Používateľ otvorí súbor v aplikácií (napríklad word processor)
2. Program vykoná analýzu obsahu.
3. Program si všíma na ktoré položky v menu používateľ kliká.

Keď je program naučený:

1. Používateľ otvorí súbor.
2. Program analyzuje obsah.
3. Program vypočíta pravdepodobnosti použitia jednotlivých položiek v menu.
4. Program zmení poradie položiek v menu, podľa vypočítaných pravdepodobností

3.3 Výber algoritmov

Vývojárovi aplikácie bude určite k úžitku zoznam položiek UI zoradených podľa pravdepodobností. Existuje mnoho algoritmov pomocou ktorých je toto možné dosiahnuť. Rozhodol som sa vybrať naivný bayesovský algoritmus. Síce to nie je najefektívnejší algoritmus k dispozícií, ale poskytuje veľmi dobrú efektivitu spojenú s nízkymi nárokmi na výpočetný výkon, čo ho robí ideálnym kandidátom na použite v tomto frameworku. Druhým algoritmom, ktorý som sa rozhodol použiť je Support Vector Machine. Support vector machine poskytuje väčšiu efektivitu ako naivný bayes [14]. SVM poskytuje binárne odpovede na klasifikáciu súborov. Toto umožní rozšírenie možných použití frameworku.
Bayes sa by sa dal použiť napríklad na upravenie poradia položiek v menu, alebo vybratie najpravdepodobnejších tlačidiel.
SVM by bolo možné použiť na veľmi efektívnu binárnu detekciu typu dokumentu a zobrazenie položiek užívateľského rozhrania náležiacich tomuto typu.

3.4 Konceptuálny model

Táto podkapitola obsahuje stručný, zjednodušený popis štruktúry frameworku a UML diagram s vynechanými detailami 3.4. Pre detailnejšiu informáciu o štruktúre frameworku prejdite na podkapitolu "Detailná štruktúra frameworku". Kde nájdete detailný class diagram 3.8.

Trénovanie prebieha po jednotlivých triedach dokumentov. Pre každú triedu dokumentov pri trénovaní programátor poskytne množinu označenú ako patriacu a ako nepatriacu do tejto triedy. Každý algoritmus sa musí trénovať zvlášť.

Klasifikácia prebieha na jednotlivých dokumentoch, pričom výsledok je buď binárna odpoveď alebo pravdepodobnosť, v závislosti na použitom algoritme.

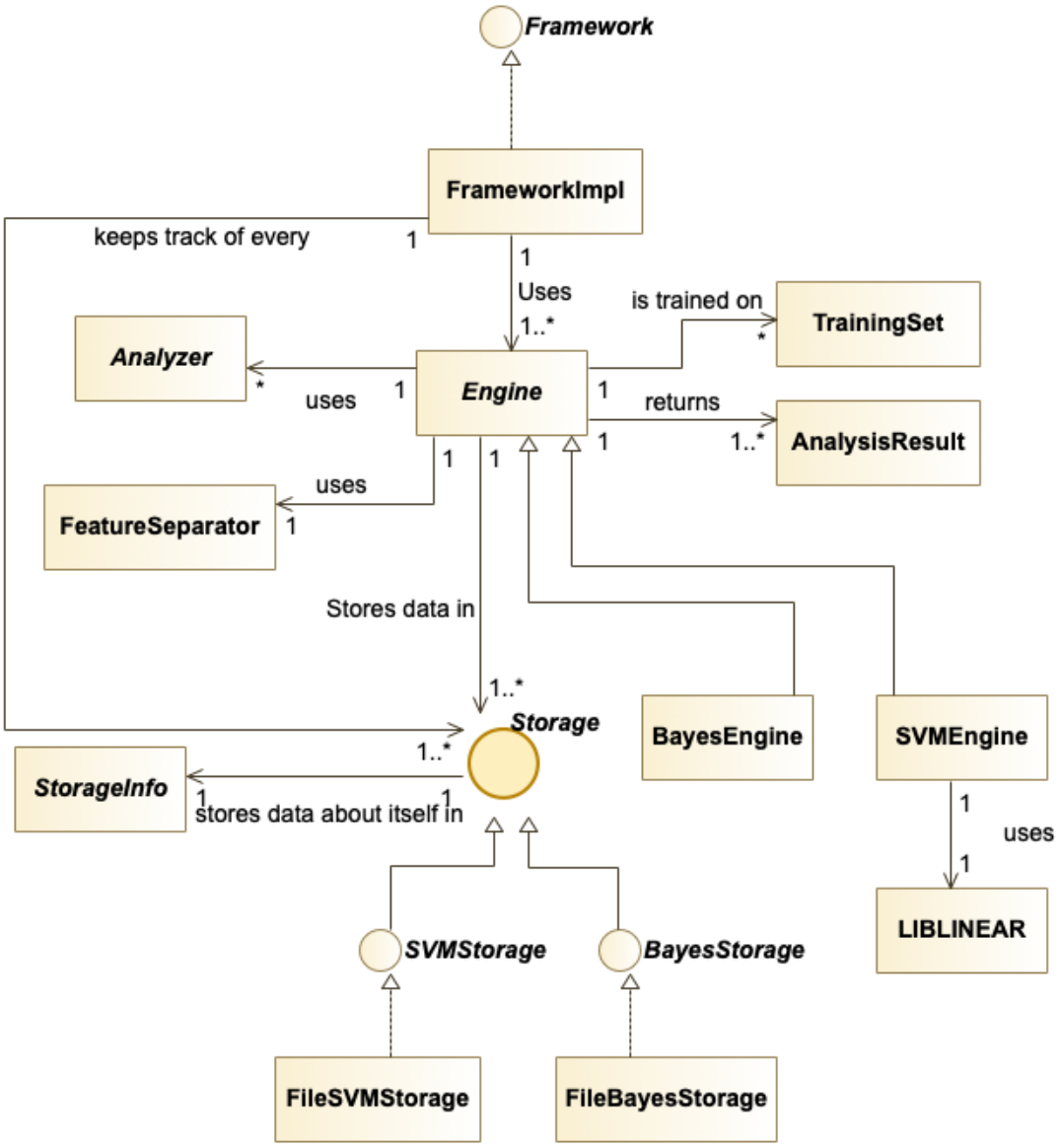
Hlavným rozhraním, s ktorým komunikuje používateľ frameworku je **Framework**. Ten je implementovaný triedou **FrameworkImpl**. *Framework* obsahuje všetky top-level funkcie na trénovanie a analýzu dokumentov a prístup k úložisku.

Samotná práca na dokumentoch prebieha v **Engine**. Vo frameworku sa môže nachádzať viac enginov. V súčasnej verzii v ňom sú dva (SVM a Bayes) a je možné ho jednoducho rozšíriť. *Engine* reprezentuje jeden klasifikačný algoritmus a zapúzdruje všetku algoritmickú prácu s ním. Všetky *Engine* musia byť odvodené od abstraktnej triedy *Engine*.

Pri trénovaní sa používa **TrainingSet**, ktorý zapúzdruje korpus dokumentov, ktoré sú binárne klasifikované ako patriace alebo nepatriace do nejakej kategórie. Trénovanie môže prebehnúť na rôznych *TrainingSet* pre rôzne kategórie, každý *Engine* sa musí trénovať zvlášť.

Trénovanie je možne aj s jedným dokumentom, bez *TrainingSet*, za účelom postupnej aktualizácie modelu používateľa, viz podkapitola "Detailná štruktúra frameworku".

FeatureSeperator generuje z nejakého dokumentu features. Defaultne "rozseká" dokument podľa whitespace znakov. Pokiaľ chcete nejaké iné vyberanie features z



Obrázek 3.4. Konceptuálny model frameworku.

dokumentu použite *Analyzer* alebo rozšírte triedu *FeatureSeperator*.

Framework je možné prinútiť pracovať s akoukoľvek vlastnosťou dokumentov, k tomu je nutné implementovať abstraktnú triedu **Analyzer** a predať ju frameworku.

Výsledky analýzy sú vrátené v triede **AnalysisResult**.

Framework ukladá model používateľa do rozhrania **Storage**. Každý typ *Engine* má svoje vlastné rozhranie pre ukladanie dát odvodené od rozhrania *Storage*. To kvôli tomu, že každý algoritmus potrebuje rozličný spôsob ukladania dát.

Každá implementácia *Storage* musí poskytovať implementáciu abstraktnej triedy **StorageInfo**. *StorageInfo* je úložisko dát o *Storage*, pomocou ktorého je možné vytvoriť nový objekt *Storage* s rovnakými dátami (napríklad pre *Storage* ukladajúce do databáze

bude *StorageInfo* obsahovať prihlasovacie údaje do tejto databáze).

3.5 Popis implementácie algoritmov

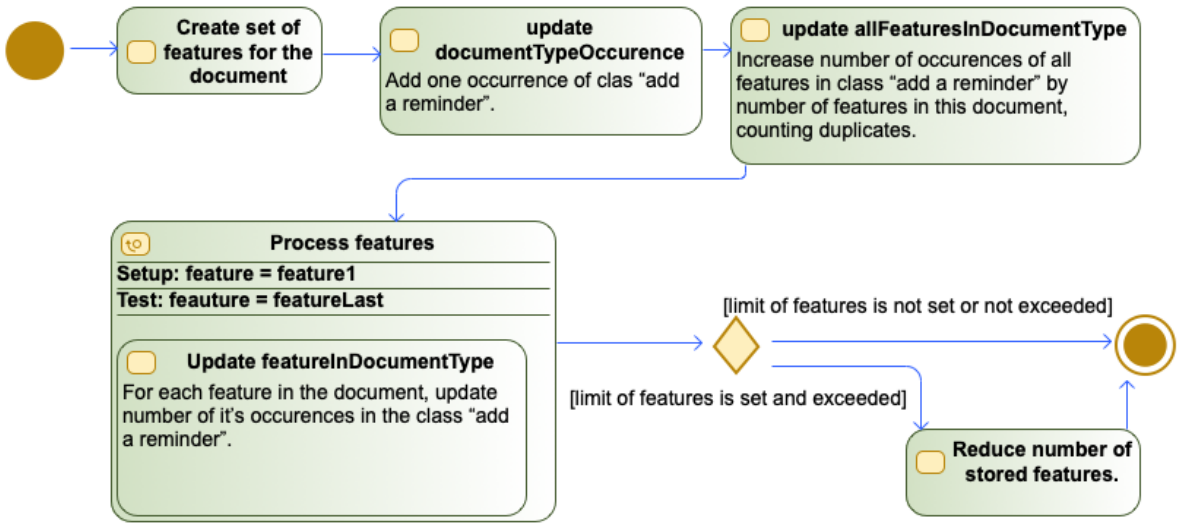
3.5.1 Naivný Bayes

Implementácia naivného Bayesa je pomerne jednoduchá záležitosť. Pre správne fungovanie algoritmu je nutné pamätať si následujúce údaje (v zátvorkách sú názvy týchto údajov použité v zdrojovom kóde frameworku):

- **počet výskytov konkrétnej feature v nejakej triede dokumentu** (*featureInDocumentType*) - napr. koľko krát sa feature "január" vyskytlo v triede "vytvoriť pripomienku".
- **počet výskytov všetkých feature v nejakej triede dokumentu** (*allFeaturesInDocumentType*) - koľko features bolo vo všetkých dokumentoch triedy "pridať pripomienku" dohromady, počíta sa každý výskyt feature.
- **množstvo dokumentov v nejakej triede dokumentu** (*documentTypeOccurance*)
- **množstvo všetkých dokumentov, v celom korpuse** (*allDocumentsCount*)
- **množstvo jednotlivých feature v triede dokumentu** (*distinctFeaturesInDocType*) - veľkosť slovníka nejakej triedy, to jest všetky feature, za každú feature sa počíta nanajvýš jeden výskyt.

Trénovanie Bayesovského klasifikátoru je znázornené na obrázku 3.5. Pri trénovaní klasifikátoru sa každý dokument prepracuje na množinu features. Následne algoritmus:

1. Aktualizuje množstvo všetkých dokumentov, množstvo dokumentov v danej triede, množstvo všetkých feature v triede dokumentu s počítaním duplikátov aj bez.
2. Prejde cez zoznam všetkých feature v tomto dokumente a aktualizuje množstvo výskytov každej feature v triede dokumentu.



Obrázek 3.5. Spracovanie jedného dokumentu pri trénovaní Bayesovského klasifikátoru.

Klasifikovanie Bayesovským klasifikátorom je znázornené na obrázku 3.6. Pri klasifikovaní dokumentu:

1. Dokument sa premení na množinu features.

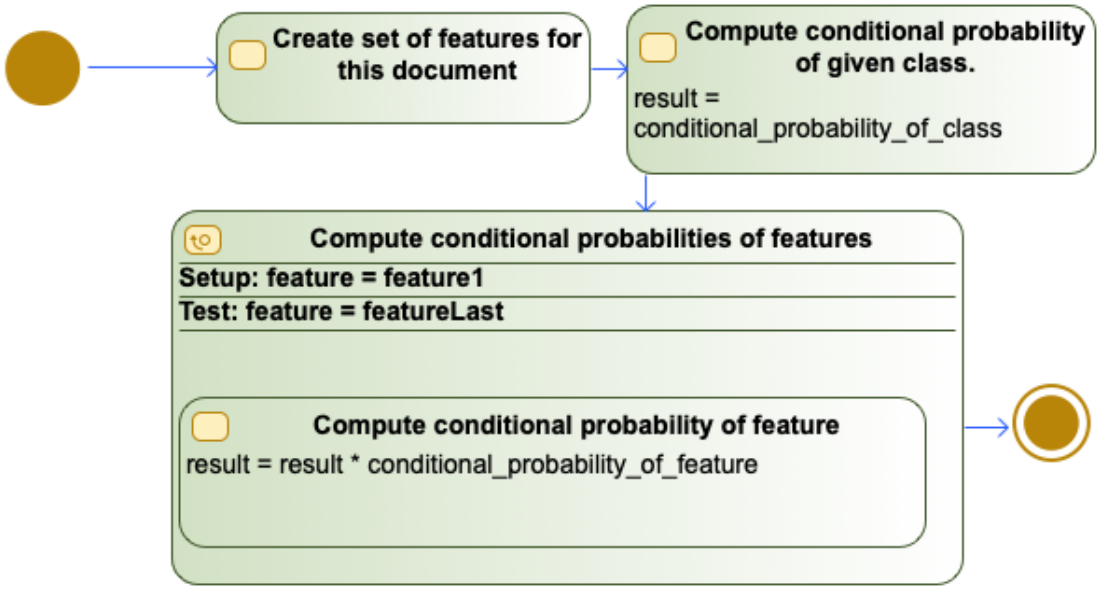
2. Vypočíta sa pravdepodobnosť, že dokument patrí do danej triedy podľa vzorca (2) kde s použitím názvov vyššie bude

$$P(c) = \frac{documentTypeOccurance}{allDocumentsCount}$$

3. Pre každú feature sa sa vypočíta podmienená pravdepodobnosť podľa vzorca (6) , ktorý v našom prípade s použitím názvov zo zdrojového kódu vyššie bude mať tvar:

$$P(feaure|trieda) = \frac{featureInDocumentType + 1}{allFeaturesInDocumentType + distinctFeaturesINDocumentType}$$

4. Vypočíta pravdepodobnosť, že dokument patrí do danej triedy podľa vzorca $P(d|c) = P(x_1|c) \cdot P(x_2|c) \cdot P(x_3|c) \cdot ... \cdot P(x_n|c) \cdot P(c)$ kde $x_1, x_2, ..., x_n$ sú jednotlivé features.



Obrázek 3.6. Klasifikovanie dokumentu pomocou Bayesovského klasifikátoru.

Jedným problémom pri programovaní Bayesovského algoritmu sú veľmi malé čísla vznikajúce pri násobení pravdepodobností pri väčších korpusoch. Na implementáciu Bayesa nie je možné použiť vstavané dátové typy s pohyblivou desatinnou čiarkou, pretože do nich nie je možné uložiť tak malé hodnoty. Ak by som použil java typ double výsledkom pre akúkoľvek klasifikáciu by bola nula. Preto som na numerické operáciu použil triedu BigDecimal zo štandardnej knižnice jazyka java [17]. BigDecimal umožňuje numerické operácie s neobmedzenou presnosťou. BigDecimal reprezentuje číslo ako základ (uložený ako string) a exponent. Operácie s BigDecimal sú o dosť pomalšie ako so vstavanými dátovými typmi. Časová náročnosť operácie násobenia je priamo úmerná veľkosti základu činiteľov narozdiel od operácií so vstavanými typmi, ktoré prebehnú v konštantnom čase. Toto spomalí klasifikátor avšak nepredstavuje to problém pokiaľ neklasifikujete neštandardne veľký dokument. V mojom testovaní som skúšal klasifikovať približne 300 stranovú knihu a klasifikácia prebehla za menej ako 2 sekundy.
Keďže trénovanie frameworku prebieha po jednotlivých triedach príkladmi dokumentov, ktoré patria a nepatria do nejakej triedy, Bayesovský klasifikátor automaticky vytvára negáciu triedy, ktorú trénujete. To jest ak napríklad trénujete triedu

`pridať_pripomienku` s príkladmi dokumentov patriacich a nepatriacich do nej, klasifikátor automaticky vytvorí triedu `NOT_pridať _pripomienku` a označí ňou všetky negatívne príklady dokumentov, s ktorými trénujete. Z tohto dôvodu nemožno žiadnu triedu dokumentu pomenovať názvom začínajúcim na reťazec `NOT_`. Ak sa o to pokúsite, dostanete výnimku. Z toho vyplýva, že sa môžte dotazovať na pravdepodobnosti týchto automaticky vytvorených tried. Ak ste natrénovali framework pre triedu `pridať_pripomienku` pozitívnymi aj negatívnymi príkladmi, môžte sa dotazovať na pravdepodobnosť patrenia do triedy `pridať_pripomienku` aj `NOT_pridať_pripomienku`.
Pri používaní Byaesovského klasifikátoru je možné zlepšiť alebo aspoň výrazne nezhoršiť úspešnosť klasifikácie redukovaním množstva použitých features [15]. To zároveň zníži výpočetný čas potrebný na klasifikáciu. Implementácia v tomto programe umožňuje nadstaviť maximálny počet features pamätaný klasifikátorom. V prípade, že maximálny počet features bol nadstavený a aktuálny počet features prekročí tento limit, algoritmus vyberie features, ktoré sa v dokumente vyskytovali s najväčšou frekvenciou. Toto je pomerne primitívny spôsob redukovania dimenzionality priestoru dokumentov, avšak prináša uspokojivé výsledky.
Redukovanie množstva features týmto spôsobom som testoval na korpuse Twenty Newsgroups [7]. Z dokumentov boli odstránené hlavičky a všetky metadáta, tak že zostalo iba telo. Ako features boli použité slová, rozdelené whitespace znakmi. Z každej kategórie bola vybraná polovica dokumentov na trénovanie a druhá polovica na testovanie. Úspešnosť klasifikácie, vzhľadom na redukovanie množstva features vyššie opísaným spôsobom je v tabuľke č. 1:

Maximálny počet features	žiadny limit	10000	20000	40000	700000
Úspešnosť klasifikácie	80,02%	82,6%	83,09%	84,03%	81,15%

Tabulka 3.1. Úspešnosť Bayesovksého klasifikátoru v závislosti na množstve pamätaných features.

Preto som nadstavil predvolený limit množstva features na 40000. Limit ide samozrejme zmeniť (zmenením konštanty BayesStorageTrimming v triede Constants).

3.5.2 Support Vector Machine

Korektná implementácia SVM klasifikátoru je netriviálna záležitosť. Našťastie existujú knižnice, v ktorých je SVM implementovaný. Jednou z populárnych je SVMLIB, vyvíjaná na Národnej Tajvanskej Univerzite. SVMLIB je k dispozícií vo viac ako dvoch tuctoch verzií pre rôzne programovacie jazyky a operačné systémy, vrátane programovacieho jazyka java.
Existujú dve verzie SVMLIB - SVMLIB samotná, ktorá pracuje s nelineárnymi kernelmi a špeciálna verzia LIBLINEAR, podporujúca iba lineárne kernely. Tvorcovia LIBSVM doporučujú používať LIBLINEAR namiesto LIBSVM v prípadoch kde:

- Množstvo features je výrazne väčšie ako množstvo klasifikovaných inštancií (dokumentov).
- Množstvo features aj množstvo inštancií (dokumentov) je veľmi veľké (ako príklad bol uvedený korpus so 47000 features a 20000 jednotlivých dokumentov) [12].

Používanie lineárnych kernelov s LIBLINEAR je obzvlášť vhodné pri klasifikovaní textu, kvôli veľmi veľkej dimenzionalite textových súborov. Efektivita LIBSVM aj

LIBLINEAR pri tomto type korpusov je takmer bez rozdielov avšak LIBLINEAR pracuje niekoľkonásobne rýchlejšie. [12] uvádza až 120 násobné zrýchlenie a len 0,2% negatívny rozdiel v efektivite. V tomto frameworku teda používam LIBLINEAR, ktorej java verzia je prístupná tu [16].
Pri použití LIBLINEAR je každý dokument reprezentovaný ako vektor dimenzie rovnej množstvu všetkých features v korpuse. Vo vektore dokumentu je na i-tom mieste:

- 0 - ak dokument neobsahuje žiadnu feature číslo i
- $x - x > 0$; kde x je množstvo výskytov feature i v tomto dokumente.

Tvorcovia LIBLINEAR stresujú dôležitosť škálovania trénovacieho korpusu, tak aby každý vektor obsahoval na všetkých pozíciach iba čísla patriace do [0; 1]. Použitie neškálovaných dát prinesie suboptimálne výsledky [12]. Každý klasifikovaný dokument musí byť samozrejme škálovaný tým istým faktorom ako trénovací korpus.
Ďalej je nutné zvoliť si parametre pre kernelovú funkciu, ktorú používate. Lineárny kernel pracuje iba s jedným parametrom označeným C v knižnici LIBLINEAR. Tvorcovia LIBLINEAR doporučujú nájsť optimálnu hodnotu parametru pre dáta, ktoré budete klasifikovať rozdelením trénovacieho korpusu na dve časti. Následne je klasifikátor predtrénovaný na jednej časti a otestovaný na druhej opakovane, s rozličnými parametrami. Doporučené hodnoty parametru pre testovanie sú ...1/8; 1/4; 1/2; 1; 2; 4; 8; 16; 32....1024... Predpokladom je, že dáta, ktoré klasifikujete sú podobné dátam na ktorých trénujete, a teda optimálny parameter pre trénovacie dáta bude aj optimálnym parametrom pre klasifikované dáta [12]. Hľadanie optimálneho parametru ale zaberie značný čas, ktorý nie je k dispozícií v aplikácií pre koncového používateľa. V mojom testovaní s korpusom Twenty Newsgroups [7] som dosiahol čo najlepšie výsledky vo všetkých kategóriach s čo najväčším C. Preto som nastavil parameter C na 2^{32}. Twenty Newsgroups obsahuje písanú angličtinu od množstva používateľov rozdelenú podľa kategórie obsahu. Predpokladám, že pre použitie v tomto frameworku môžem zvoliť rovnaký parameter, pretože tu bude LIBLINEAR použitá tiež na klasifikovanie písanej angličtiny.

Pri používanie LIBLINEAR sú dôležité tri triedy:

- **LIBLINEAR.Feature** - reprezentuje jeden typ feature v dokumente. Musí byť označená číslom svojej dimenzie a obsahovať množstvo výskytov (škálované podľa postupu vyššie).
- **LIBLINEAR.Problem** - obsahuje trénovací korpus. Pozostáva z poľa vektorov dokumentov, a poľa ktoré klasifikuje tieto dokumenty. Pole vektorov dokumentov má tvar *Feature [] []*. Nie je nutné aby v každom vektore bolo množstvo features rovné dimenzionalite korpusu, LIBLINEAR bude považovať chýbajúce dimenzie za nulové. Kvôli tomu je požadované aby každá *Feature* bola označená číslom dimenzie, ktorej náleží a aby v každom vektore i bolo pole Features [i] [] zoradené podľa čísla dimenzie.
- **LIBLINEAR.Model** - trieda pomocou, ktorej sa klasifikujú neznáme dokumenty. Získame ju volaním *LIBLINEAR.train(problem, parameter)*. Predikcie z nej získame pomocou *LIBLINEAR.predict(model, dokument)*.

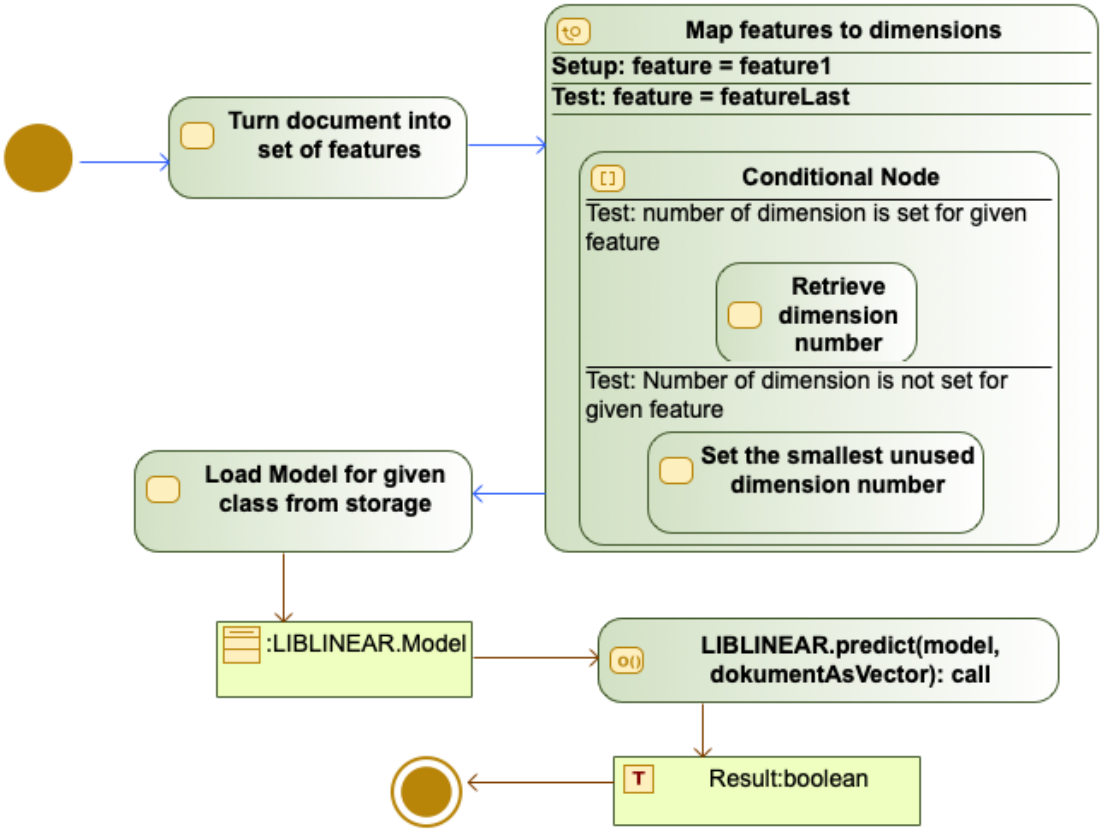
Postup pri trénovaní SVM klasifikátoru pre nejakú triedu dokumentu je znázorneneý na obrázku B.3:

1. Dokument sa premení na množinu features. Kvôli požiadavkám LIBLINEAR je nutné aby sa každá feature vždy namapovala do tej istej dimenzie. Ku každej feature je získané číslo jej dimenzie a výsledná reprezentácia dokumentu ako množiny features je zoradená podľa čísla dimenzie.

2. Získa sa už existujúci *LIBLINEAR.Problem* pre túto triedu, vyplnený vektormi predchádzajúcich dokumentov. Vektor práve spracovaného dokumentu sa k nemu pridá.
3. Keď sa do *Problem* pridajú všetky trénovacie dokumenty, *Problem* sa optimalizuje tak aby v ňom bol rovnaký počet pozitívnych a negatívnych príkladov. Z dôvodu optimalizovania rýchlosti je výber príkladov náhodný. V prípade že je negatívnych príkladov viac ako pozitívnych, sa z *Problem* náhodne vyhadzujú negatívne príklady, kým sa počet pozitívnych a negatívnych nerovná a naopak.
4. Problém je normalizovaný a normalizačný faktor pre túto triedu dokumentu sa uloží. Zavolá sa *LIBLINEAR.train(problem, parameter)*, získaný model sa uloží pod triedu dokumentu.

Postup pri klasifikácií neznámeho dokumentu je znázornený na obrázku 3.7:

1. Premeň dokument na množinu features.
2. Ku každej features získaj číslo jej dimenzie a vytvor vektor dokumentu podľa požiadavok LIBLINEAR.
3. Načítaj *LIBLINEAR.Model* pre daná triedu z úložiska.
4. Zavolaj *LIBLINEAR.predict(model, reprezentacia_dokumentu_ako_vektor)* a vráť klasifikačný výsledok.



Obrázek 3.7. Klasifikovanie dokumentu pomocou LIBLINEAR.

3.6 Detailná štruktúra frameworku

Táto podkapitola obsahuje detailný UML class diagram štruktúry frameworku, podrobný popis tried a sekvenčné diagramy práce frameworku.

3.6.1 Popis tried

Framework sa vytvorí konštruktovaním novej inštancie objektu FrameworkImpl. Konštruktor má 3 parametry:

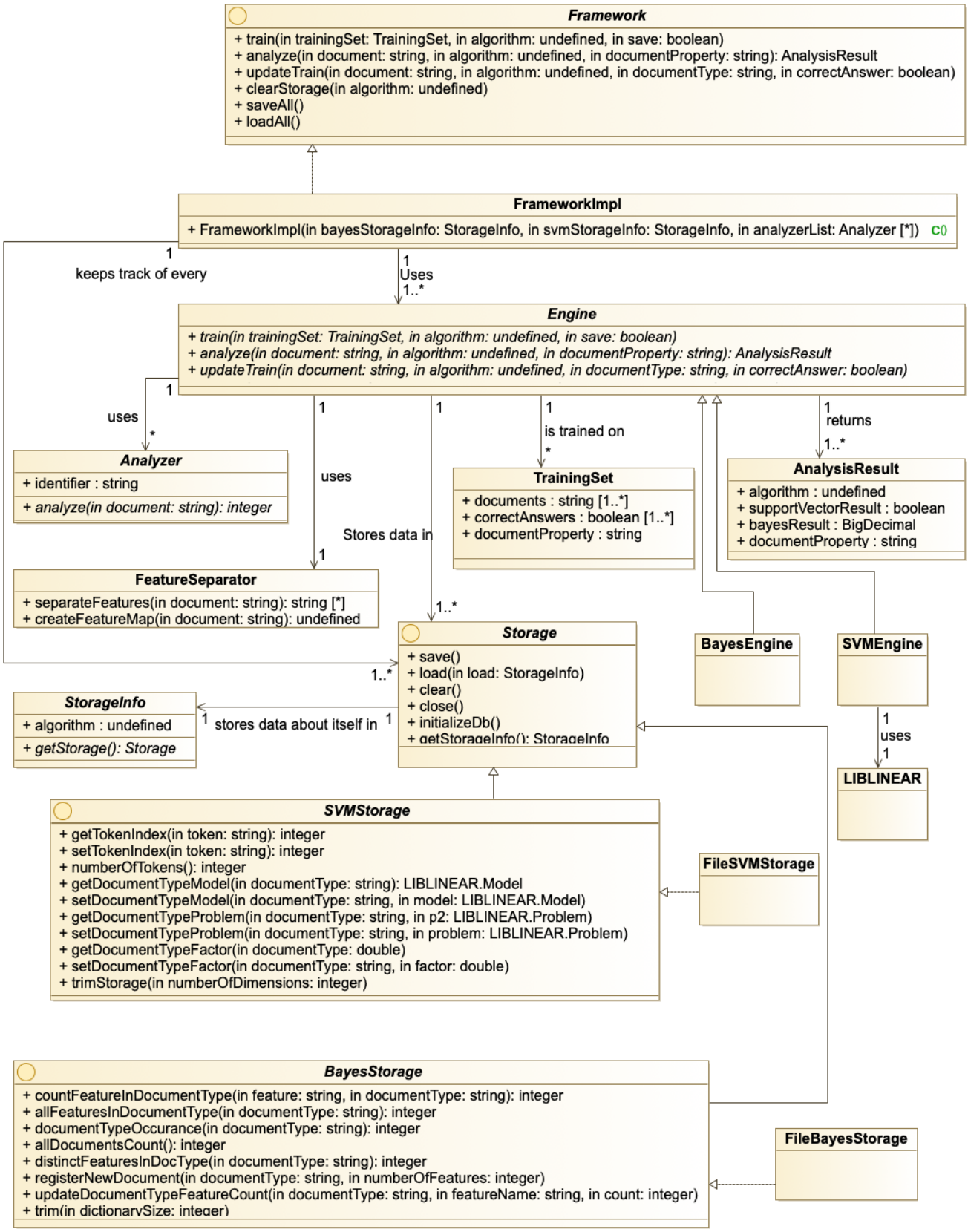
- *BayesStorageInfo* - objekt obsahujúci informácie o úložisku pre model Bayesovského klasifikátoru. Nadstavte na null pokiaľ nechcete použiť Bayesovský klasifikátor.
- *SVMStorageInfo* - objekt obsahujúci informácie o úložisku pre model SVM klasifikátoru. Nadstavte na null pokiaľ nechcete použiť SVM klasifikátor.
- *List < Analyzer >* - zoznam analyzátorov implementovaných programátorom. Každý analyzátor generuje z dokumentu jeden typ feature. Nadstavte na null pokiaľ nechcete použiť žiadne vlastné features.

Použitie frameworku ďalej prebieha podľa rozhrania Framework, ktoré FrameworkImpl implementuje:

- *train* - slúži na trénovanie na korpuse dokumentov. Ako parametre potrebuje korpus dokumentov; typ klasifikátoru, ktorý použiť; a boolean určujúci či sa model ihneď po klasifikácií uloží do úložiska(pre veľké korpusy môže ukladanie zabrať dlhšiu dobu).
- *updateTrain* - slúži na postupné trénovanie iba s jedným dokumentom. Model používateľa je aktualizovaný o tento jeden dokument. Typické použitie je volanie updateTrain zakaždým keď používateľ klikne na nejaký prvok UI. Parametre sú dokument; trieda dokumentu; informácia či dokument patrí alebo nepatrí do tejto triedy; a trénovaný klasifikátor.
- *analyze* - vracia klasifikáciu pre nejaký dokument. Parametre sú dokument; trieda dokumentu; a použitý klasifikátor. Pri vybraní Bayesovského klasifikátoru vráti pravdepodobnosť, že dokument patrí do danej triedy, pri použití SVM klasifikátoru vráti binárnu odpoveď.
- *saveAll* - uloží všetky modely, všetkých klasifikátorov
- *loadAll* - načíta modely všetkých klasifikátorov.

Funkcie *train, updateTrain* a *analyze* sú v rozhraní Framework zadefinované ako hádžuce výnimky. Implementácia nutne žiadnu výnimku hádzať nemusí, ale sú tam zadefinované pre prípad potreby.

Analyzer je abstraktná trieda, ktorou sa dá rozšíriť množstvo features použitých vo frameworku. Framework defaultne berie ako features dokumentu dokument "rozsekaný" whitespace znakmi. *Analyzer* musí mať názov feature, ktorú detekuje v dokumente. Tento názov má formu textového stringu. Názov musí byť unikátny a nesmie sa vyskytovať v analyzovaných dokumentoch ako slovo. Taktiež sa nesmie začínať žiadnym reťazcom uvedeným v triede *Constans.RESERVED_CATEGORY_NAME_PREFIXES* čo je zoznam názvov vyhradených pre vnútorné použitie vo frameworku. Následne implementujte asbtraktnú funckciu analyze - tá si vezme ako parameter analyzovaný dokument a vráti množtvo fetures, ktoré tento *Analyzer* detekuje v dokumente. Napríklad ak je názov analyzátoru `____koncept_mobil` a jeho funckia analyze vráti hodnotu 3



Obrázek 3.8. UML class diagram frameworku.

znamená to že v dokumente sa nachádzajú 3 features typu `____koncept_mobil`.

Zoznam vašich analyzátorov, predáte triede *FrameworkImpl* pri konštrukcii.

FeatureSeparator je trieda, ktorá z dokumentu vytvorí zoznam features. V súčasnej implementácií jednoducho "rozseká" dokument whitespace znakmi a vráti zoznam výskytov jednotlivých slov.

TrainingSet je trieda reprezentujúca korpus dokumentov na ktorej je framework trénovaný. Každý *TrainingSet* slúži na trénovanie iba pre jednu triedu dokumentu/element používateľského rozhrania. Táto trieda je v člene *documentProperty*. Do zoznamu *documents* sa pridávajú jednotlivé dokumenty, zoznam *correctAnswers* binárne klasifikuje všetky dokumenty z *documents* ako patriace alebo nepatriace do triedy *documentProperty*.

AnalysisResult obsahuje výsledok analýzy. Jeho členmi sú klasifikátor, ktorý bol použitý a trieda dokumentu ktorej sa klasifikácia týkala. Pre SVM klasifikátor je výsledkom binárna hodnota či dokument patrí alebo nepatrí do danej triedy, pre Bayesovský klasifikátor je výsledkom pravdepodobnosť že dokument patrí do danej triedy.

Engine je abstraktná trieda zapúzdrujúca samotné algoritmy klasifikácie. Každý klasifikátor implementuje vlastnú implementáciu triedy Engine. Funkcie *Engine* sú:

- *train* - slúži na trénovanie na korpuse dokumentov. Ako parametre potrebuje korpus dokumentov, typ klasifikátoru, ktorý použiť, boolean určujúci či sa model ihneď po klasifikácií uloží do úložiska(pre veľké korpusy môže ukladanie zabrať dlhšiu dobu).
- *updateTrain* - slúži na postupné trénovanie iba s jedným dokumentom. Model používateľa je aktualizovaný o tento jeden dokument. Typické použitie je volanie updateTrain zakaždým keď používateľ klikne na nejaký prvok UI. Parametre sú dokument, trieda dokumentu, informácia či dokument patrí alebo nepatrí do tejto triedy a trénovaný klasifikátor.
- *analyze* - vracia klasifikáciu pre nejaký dokument. Parametre sú dokument, trieda dokumentu a použitý klasifikátor. Pri vybraní Bayesovského klasifikátoru vráti pravdepodobnosť, že dokument patrí do danej triedy, pri použití SVM klasifikátoru vráti binárnu odpoveď.
- *getStorage* - vráti Storage objekt, v ktorom je uložený model klasifikátoru.

Funkcie *train, updateTrain* a *analyze* hádžu výnimky. Výnimka v implementácií *Engine* môže byť vyvolaná z akéhokoľvek dôvodu, v mojej implementácií sú výnimky hádzané v prípade, že názov triedy dokumentu nie je povolený alebo bol funkcii predaný argument algoritmu, ktorý sa nezhoduje s typom *Engine*.

Existujú dve implementácie *Engine* - *SVMEngine* pre SVM klasifikátor, ktorý používa LIBLINEAR, a *BayesEngine* s mojou vlastnou implementáciou naivného Bayesa.

Všetky objekty pre ukladanie modelu klasifikátorov sú odvodené od rozhrania **Storage**. Každá implementácia *Engine* predpisuje svoje vlastné rozhranie pre ukladanie modelu odvedené od rozhrania *Storage*. Každé *Storage* obsahuje objekt **StorageInfo**, ktorý popisuje destináciu kam sa dáta ukladajú. Toto môže cesta k

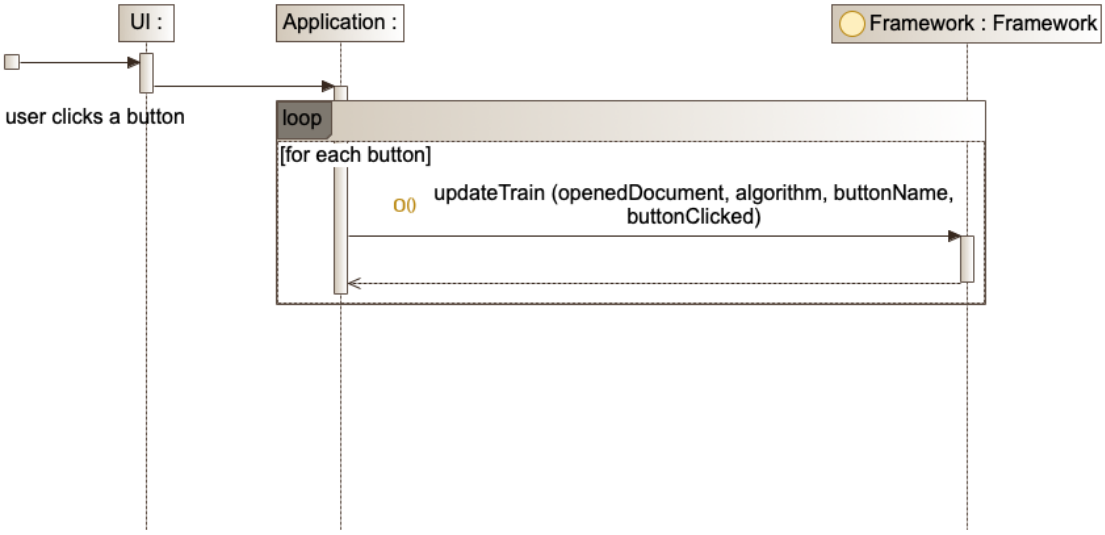
súboru, prihlasovacie údaje k databáze a podobne.
Framework obsahuje implementácie *Storage* pre oba algoritmy, ktoré ukladajú údaje do súborov na pevnom disku. Ukladanie prebieha pomocou serializácie objektov z java collections. Experimentálne som skúšal implementáciu úložiska pre Bayesovksý klasifikátor, ktorá by ukladala dáta do databáze, (používal som MYSQL) avšak toto bolo pri väčších korpusoch neprípustne pomalé.

3.6.2 Sekvenčné diagramy

Táto podkapitola obsahuje sekvenčné diagramy pracovania frameworku.

3.6.3 Príklad použitia v aplikácií

Vývojár by typicky volal framework v dvoch prípadoch. V prípade že používateľ klikne na prvok používateľského rozhrania, je potrebné asociovať otvorený textový súbor s týmto dokumentom. Toto je znázornené na obrázku 3.9. V prípade, že používateľ otvorí nový dokument je potrebné zistiť, ktoré prvky používateľského rozhrania pravdepodobne použije a adaptovať podľa tohto rozhranie. To je znázornené na obrázku 3.10.

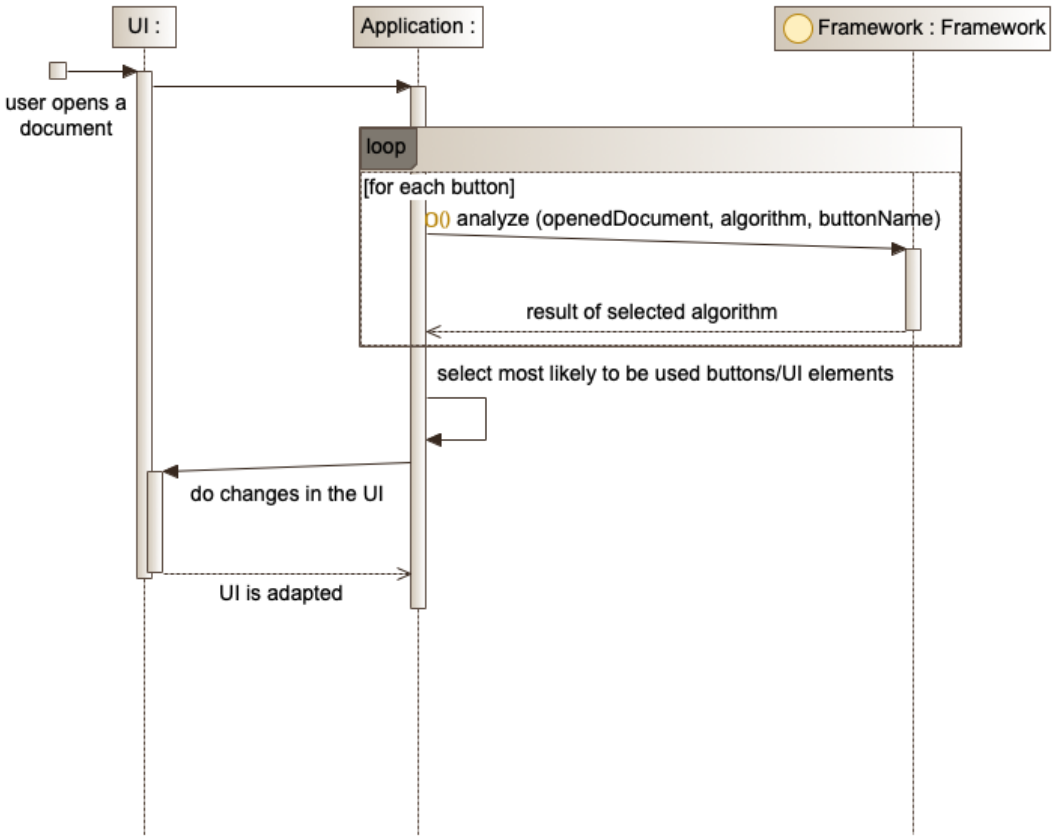


Obrázek 3.9. Aktualizácia modelu používateľa po kliknutí na element používateľského rozhrania.

3.6.4 Trénovanie a klasifikovanie

Obrázok B.4 popisuje postup volaní pri spracovaní jedného dokumentu počas trénovania Bayesovského klasifikátoru. Najprv sa dokument premení na množinu features volaním FeatureSeparator, ktorý vráti features ako HashMapu <meno_feature, počet_výskytov>. Potom sa do tejto HashMapy pridajú všetky features získané zoznamom Analyzer, ktorý poskytol vývojár aplikácie. Nový dokument sa zaregistruje u úložiska, ktoré aktualizuje relevantné hodnoty, a následne sa separátne spracuje každá feature z HashMapy a aktualizované údaje sa uložia.

Obrázok B.5 popisuje postup volaní pri klasifikovaní neznámeho dokumentu pomocou Bayesovského klasifikátoru. Opäť, dokument sa premení na množinu



Obrázek 3.10. Adaptácia používateľského rozhrania pomocou frameworku.

features uloženú v HashMape <názov_feature, počet_výskytov>. Následne sa vypočíta podmienená pravdepodobnosť dokumentu z množstva výskytov uložených v úložisku a následne sa táto pravdepodobnosť násobí podmienenou pravdepodobnosťou každej feature ako je popísané na obr 3.6

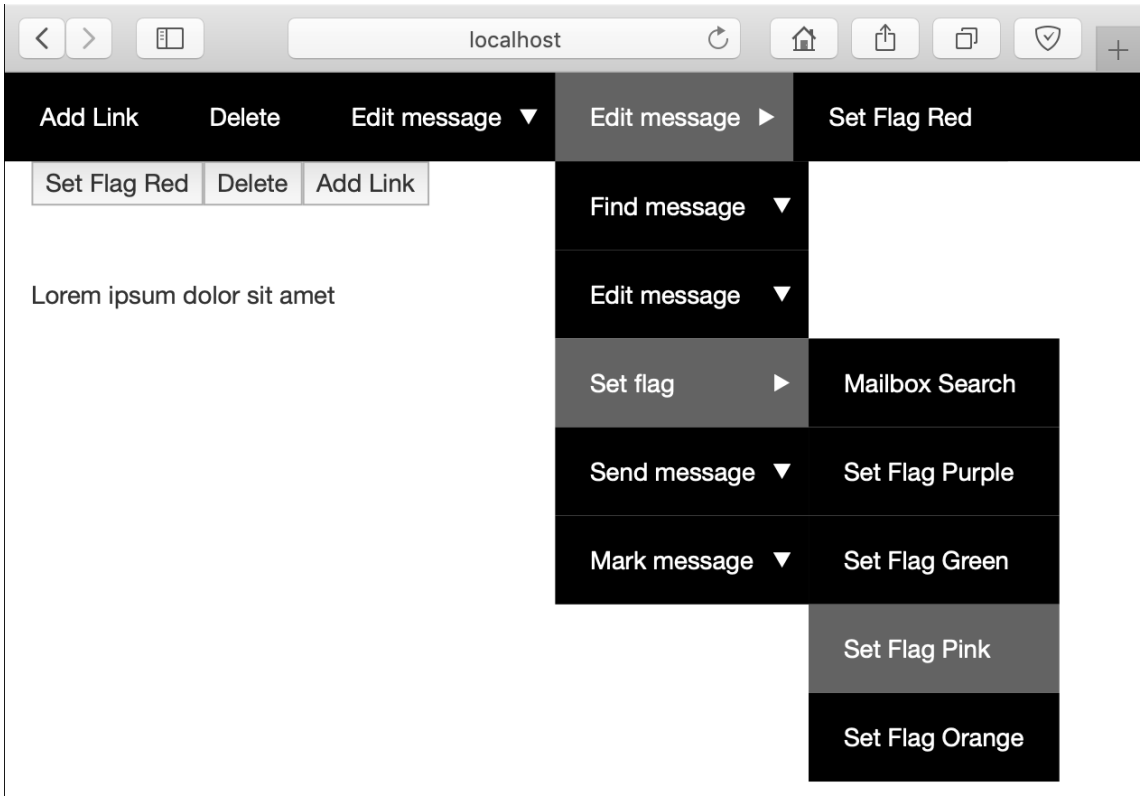
Obrázok B.6 popisuje postup volaní pri spracovaní jedného dokumentu počas trénovania SVM klasifikátoru. Najprv sa dokument premení na množinu features volaním FeatureSeparator, ktorý vráti features ako HashMapu <meno_feature, počet_výskytov>. Potom sa do tejto HashMapy pridajú všetky features získané zoznamom Analyzer, ktorý poskytol vývojár aplikácie. Následne sa získa LIBLINEAR.Problem pre danú triedu. Do tohto problému sa pridá práve spracovaný dokument. Problém sa optimalizuje, tak aby mal rovnaký počet pozitívnych a negatívnych príkladov. Znormalizuje sa a potom sa opäť uloží. Následne je zavolaná funkcia train knižnice Liblinear a získaný model na predikcie sa uloží pod daná triedu dokumentu.

Obrázok B.7 popisuje postup volaní pri klasifikovaní neznámeho dokumentu pomocou SVM klasifikátoru. Najprv sa dokument premení na množinu features volaním FeatureSeparator, ktorý vráti features ako HashMapu <meno_feature, počet_výskytov>. Potom sa do tejto HashMapy pridajú všetky features získané zoznamom Analyzer, ktorý poskytol vývojár aplikácie. Následne sa dokument prepracuje na vektor podľa požiadavok Liblinear, získa sa z úložiska model pre

danú triedu dokumentu a zavolá sa funkcia predict knižnice Liblinear, ktorej binárny výsledok sa vráti.

Kapitola 4
Vývoj testovacej aplikácie

Táto kapitola popisuje vývoj aplikácie na otestovanie frameworku. Stručne popisuje bakalársku prácu N. Mishchenka, ktorá bola použitá na vývoj, integráciu frameworku do jeho práce, štruktúru a návod na spustenie projektu. Na obrázku 4.1 môžte vidieť túto testovaciu aplikáciu.



Obrázek 4.1. Testovacia aplikácia používajúca Mishchenkov framework.

4.1 Aspektově orientovaný vývoj adaptivní struktury aplikace pro Java EE aplikace od Nikitu Mishchenka

Nikita Mishchenko pre svoju bakalársku prácu navrhol a implementoval framework na adaptáciu menu [1]. Napísal ho v aspektovo orientovanom frameworku Spring 1.8 v jazyku java enterprise. Mischenkov framework pracuje s menu v aplikácie ako so stromovou štruktúrou. Každý uzol v strome predstavuje podmenu a každý list je položkou v menu, na ktorú je možné kliknúť, a ktorá vykonáva nejakú akciu v aplikácií. Framework funguje na princípe usporiadania stromu tak aby sa listy s najväčšou váhou nachádzali čo najbližšie ku koreňom stromu.

Pri použití Mishchenkovho frameworku, vývojár definuje množinu všetkých tlačidiel, ktoré sa majú nachádzať v menu. Ku každému tlačidlu je potrebné poskytnúť následujúce údaje:

- *Kategória tlačidla* (UPDATE, CREATE, READ, DELETE).
- *Meno tlačidla* - meno používané frameworkom, nie to to, ktoré uvidí používateľ.
- *Meno tlačidla viditeľné používateľovi*
- *Dočasnú váhu tlačidla* - váha, ktorá sa použije na vytvorenie stromu menu, pred tým ako sa strom optimalizuje učením od používateľa. Táto váha má dobu vypršania platnosti po uplynutí ktorej sa vynuluje.
- *Kľúčové slová pre tlačidlo* - tlačidlo by malo mať kľúčové slová pozostávajúce z podstatných mien, ktoré opisujú akciu, ktorú toto tlačidlo vykonáva. Framework sa pokúsi zoskupiť tlačidlá, ktoré používajú rovnaké kľúčové slová.
- *Kľúčové slová akcie tlačidla* - slovesá popisujúce akciu tohto tlačidla. Tieto slovesá by mali mať všeobecný charakter a mali by sa opakovať pre rôzne tlačidlá. Pri vytváraní stromu menu framework použije tieto slovesá na pomenovanie uzlov podmenu v tomto strome. Podmenu sú pomenované automaticky kombináciou kľúčových slov, tak aby vystihovali tlačidlá nachádzajúce sa v tomto podmenu.

Optimalizácia menu prebieha na základe váh jednotlivých tlačidiel. Framework je navrhnutý tak, že každé kliknutie na nejaké tlačidlo zvýši váhu tohto tlačidla o 1. Výsledná váha tlačidla sa vypočíta ako počet kliknutí od používateľa + dočasná váha tlačidla. Aktualizácia štruktúry prebieha zakaždým potom čo sa používateľ prihlási do aplikácie.

Ako demonštračnú aplikáciu som vytvoril jednoduchý e-mail klient, používajúci môj a Mischenkov framework. Táto aplikácia obsahuje iba používateľské rozhranie, neimplementuje žiadnu funkcionalitu pretože jej účelom je otestovať adaptáciu rozhrania, k čomu skutočná funkcionalita nie je nutná. Vytvoril som aplikáciu v Mishchenkovom frameworku, v ktorej sú mená jednotlivých tlačidiel vzaté z menu vstavaného e-mail klienta v MacOs 10.14.

4.2 Integrácia do Mishchenkovho frameworku

Prvým problémom, na ktorý som narazil bolo to, že Mishchenkov framework aktualizuje štruktúru menu, vždy po tom ako sa používateľ prihlási do aplikácie. Moja aplikácia ju ale potrebuje aktualizovať zakaždým keď sa zmení zobrazený e-mail. Mishchenko nezahrnul možnosť vynútiť aktualizáciu menu vo svojej práci preto som ju do nej musel doplniť. To som spravil vytvorením REST služby, v hlavnom Interface Mishchenkovej práce, ktorá vynúti prepočítanie stromu, tak že Mishchenkove funkcie na prístup ku štruktúre stromu už vrátia aktualizovanú verziu.
Druhým problémom bolo vyriešenie prenosu výstupu z môjho frameworku do Mishchenkovej práce. Mishchenkova práca pracuje s množstvom kliknutí na jednotlivé položky. Moja poskytuje binárnu klasifikáciu dokumentov (SVM) alebo pravdepodobnosť patrenia do nejakej triedy dokumentu (Bayes). Binárnu klasifikáciu zo SVM by nebolo možné vložiť do Mishchenkovho frameworku, to ju teda vylučuje. Výstup z Bayesovského klasifikátoru sa dá ale veľmi jednoducho premeniť na frekvencie klikania, s ktorými pracuje Mishchenko.
V aplikácií je každá položka v menu reprezentovaná ako trieda dokumentu. V

prípade, že používateľ klikne na nejakú položku, je frameworku predaný práve zobrazený e-mail ako dokument patriaci do triedy tlačidla, na ktoré bolo kliknuté. To jest používateľ má zobrazený nejaký email a kliká na tlačidlá v menu. Aplikácia následne vykonáva Bayesovskú analýzu pomocou môjho frameworku, ktorou vytvorí model závislosti zobrazeného e-mailu a používaných položiek v menu.
Keď sa e-mail zmení, môj framework vypočíta ku každému tlačidlu pravdepodobnosť, že naň používateľ klikne Bayesovským klasifikátorom. Táto pravdepodobnosť musí byť nejako predaná do Mishchenkovho frameworku. Postup je následovný:

1. Nájdi tlačidlo, ktoré má najmenšiu pravdepodobnosť.
2. Nájdi mocninu 10, takú že po vynásobení pravdepodobnosti tlačidla z bodu 1 touto mocninou je výsledok nie menší ako 1.
3. Vynásob pravdepodobnosti všetkých tlačidiel mocninou z bodu 2
4. Zaokrúhli všetky čísla z bodu 3 na celé čísla.
5. Predaj čísla z bodu 3 Mishchenkovmu frameworku ako množstvo kliknutí na jednotlivé tlačidlá.

Po predaní pravdepodobností Mishchenkovmu frameworku sa vynúti aktualizácia stromovej štruktúry menu a adaptované menu je zobrazené používateľovi.
Aplikáciu som tiež doplnil tromi rýchlymi tlačidlami do ktorých sú vybraté 3 položky s najväčšími pravdepodobnosťami pre zobrazený e-mail.

Vykonané úpravy Mishchenkovej práce:

- Pridanie funckie doUserClicks a forceUpdateMenu priamo v Mishchenkovom frameworku. Slúžia na aktualizáciu váh o viac ako +1 naraz, a vynútenie prepočítania stromu. (súbor: module-bachelor/com.wibkit/service/UserStructureService
- Zmeny na Mishchenkovej demonštračnej aplikácií, ktorú som upravil na moju testovaciu aplikáciu. (balíček: module-core/...)

4.3 Štruktúra projektu

Projekt pozostáva z Mishchenkovej práce, do ktorej bol vložený môj framework. Pozostáva z troch balíčkov:

- *module-bachelor* - Mishchenkov framework.
- *module-core* - Mishchenkova demonštračná aplikácia, ktorú som upravil na e-mailový klient popísaný vyššie.
- *module-test* - Mishchenkove unit testy.
- *text_adaptation_framework* - obsahuje môj framework.

Balíček *text_adaptation_framework* ďalej obsahuje podbalíčky:

- *test_application* - triedy potrebné pre testovaciu aplikáciu.
- *text_adaptation_framework* - samotný framework.
- *test* - testy pre SVM a Bayesovský klasifikátor, vyhodnocujúce efektivitu, a unit test pre BayesStorage.

4.4 Návod na spustenie aplikácie

1. Stiahnite si vývojarské prostredie IntelliJ IDEA dostupné na https://www.jetbrains.com/idea/

2. V úvodnom okne IntelliJ zvoľte možnosť importovať projekt
3. Importujte projekt ako projekt zo systému Graddle
4. Keď Intellij skončí sťahovanie závislostí a indexovanie projektu nájdite zdrojový súbor `BachelorThesisApplication.java` v balíčku module_core (module_core/src/main/java/com.wibkit/BachelorThesisApplication.java). Kliknite naň pravým tlačidlom a zvoľte `run`.
5. Počkajte kým IntelliJ zkompiluje a spustí spring aplikáciu.
6. Otvorte Váš webový prehliadač a choďte na adresu `http://localhost:8080`.
7. Vytvorte používateľský účet a prihláste sa.
8. Teraz máte možnosť klikať na položky menu.
9. Zobrazený e-mail zmeníte písaním na štandardný vstup programu v konzole IntelliJ.

4.4.1 Návod na spustenie testov klasifikátorov

Testy pracujú s korpusom dokumentov, ktorý je rozdelený na kategórie. Štruktúra korpusu musí pozostávať z koreňového adresára, v ktorom sú podadresáre, každý reprezentujúci jednu triedu dokumentov. Názov triedy dokumentov je názvom podadresára. Každý z týchto podadresárov potom smie obsahovať ľubovoľné množstvo textových súborov.
Testy efektivity sú *BayesEngineTest* a *SVMEngineTest*. Oba majú člena *path*. Ten nadstavte na cestu ku koreňovému priečinku korpusu. Taktiež nadstavte položku *relearn* na hodnotu true. Pokiaľ nadstavíte *relearn* na false a test ešte nikdy predtým nebežal, test sa pokúsi nájsť uložený model na disku, nenájde ho a zlyhá s chybou.
Po náležitých úpravách test spustíte pravým klikom na jeho zdrojový súbor v IntelliJ a zvolením položky “run”.

Kapitola 5
Testovanie

Táto kapitola sa zaoberá testovaním použitých klasifikátorov a otestovaním aplikácie vyvinutej v predchádzajúcej kapitole.

5.1 Testovanie efektivity klasifikátorov

Testovanie klasifikátorov prebehlo na korpuse 20 Newsgropus [7]. Korpus pozostáva z príspevkov na diskusné fóra z 90. rokov. Je roztriedený na 20 kategórií podľa oddelení na diskusnom fóre. Každá kategória obsahuje 500 príspevkov. Všetky príspevky obsahujú metadáta a hlavičky. Pred testovaním som zo všetkých príspevkov tieto metadáta a hlavičky odstránil, tak, že zostalo iba telo textu, ktoré niekto napísal na diskusné fórum. Každá kategória boľa rozdelená na dve polovice. Z prvej polovice sa trénoval klasifikátor a druhá polovica sa použila na overenie efektivity.

5.1.1 Bayesovský klasifikátor

Pre každý testovací dokument bola vypočítaná pravdepodobnosť, že patrí do každej kategórie. Ako predikcia sa vzala kategória s najväčšou pravdepodobnosťou. Pre účely adaptovania používateľského rozhrania nie je tak veľmi dôležité aby sa dokument klasifikoval správne, ale aby správna kategória bola čo najvyššie v zostupne zoradenom zozname pravdepodobností. Preto som zahrnul výčet koľko krát správna kategorizácia skončila s druhou najväčšou pravdepodobnosťou, treťou najväčšou pravdepodobnosťou a tak ďalej.

Správna kategorizácia [počet dokumentov]: 4203
Nesprávna kategorizácia [počet dokumentov]: 798
Efektivita: 84%

Pozícia správneho výsledku na zozname zostupne zoradených pravdepodobností patrenia do kategórie je v tabuľkách 5.1, 5.2, 5.3 a znázornená na grafe 5.1.

Pozícia	1	2	3	4	5	6	7
Množstvo dokumentov	4203	414	113	79	38	23	18

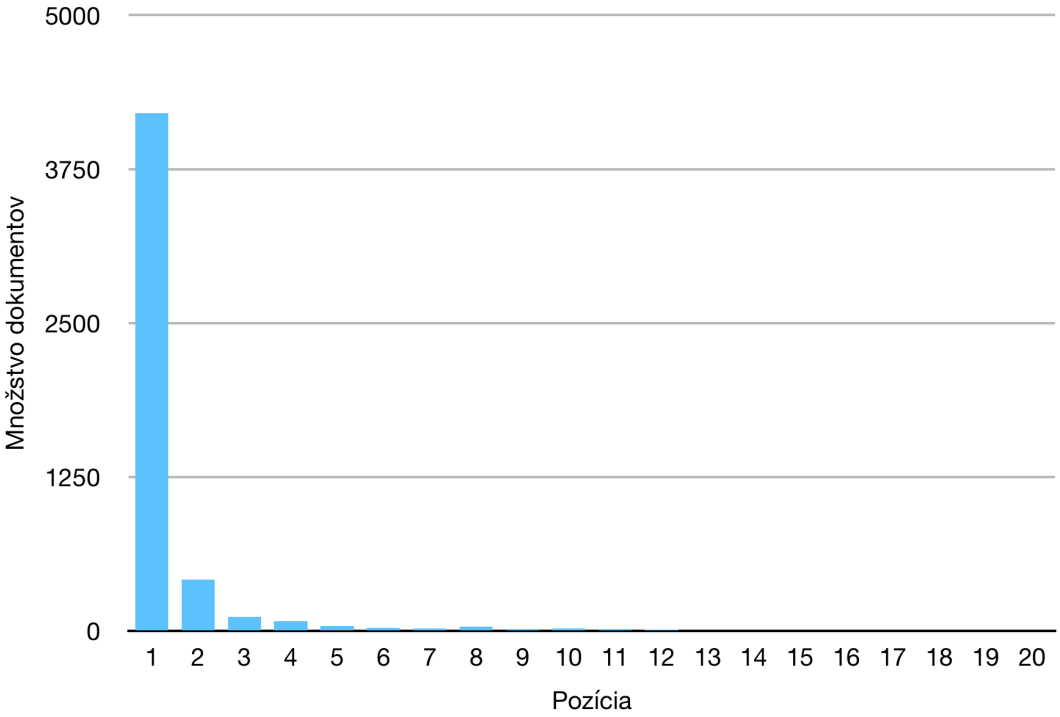
Tabulka 5.1. Závislosť množstva dokumentov a pozície správnej klasifikácie na zozname zostupne zoradených pravdepodobností pri klasifikovaní naivným Bayesom. (časť 1)

Pozícia	8	9	10	11	12	13	14
Množstvo dokumentov	33	15	20	12	10	8	3

Tabulka 5.2. Závislosť množstva dokumentov a pozície správnej klasifikácie na zozname zostupne zoradených pravdepodobností pri klasifikovaní naivným Bayesom. (časť 2)

Pozícia	15	16	17	18	19	20
Množstvo dokumentov	4	2	2	2	1	1

Tabulka 5.3. Závislosť množstva dokumentov a pozície správnej klasifikácie na zozname zostupne zoradených pravdepodobností pri klasifikovaní naivným Bayesom. (časť 3)



Obrázek 5.1. Úspešnosť klasifikácie bayesovským klasifikátorom.

5.1.2 SVM klasifikátor

Testovanie prebehlo no korpuse 20 Newsgroups [7]. Každá kategória bola trénovaná zvlášť. Každá kategória sa rozdelila na dve polovice. Pri trénovaní každej kategórie sa použila polka tejto kategórie ako pozitívne príklady a polka zo všetkých ostatných kategórií ako negatívne príklady. Pri trénovaní každej kategórie bolo teda negatívnych príkladov 19 krát viac ako pozitívnych. To ale nie je problém, pretože moja implementácia automaticky upraví korpus tak aby bol počet negatívnych a pozitívnych príkladov rovnaký. Následne sa sa druhá polovica danej kategórie a druhá polovica všetkých ostatných kategórií použila na overenie efektivity. Sledoval som následujúce údaje:

False Positive - dokumenty nesprávne klasifikované ako patriace do danej kategórie
False Negative - dokumenty nesprávne klasifikované ako nepatriace do danej kategórie
True Positive - dokumenty správne klasifikované ako patriace do danej kategórie
True Negative - dokumenty správne klasifikované ako nepatriace do danej kategórie

Priemer z binárneho klasifikovania všetkých kategórií:

False Negative: 0,1%
False Positive: 12,8%
True Positive: 4,8%
True Negative: 82,3%

Positive spolu: 87,1%
Negative spolu: 12,9%

5.2 Testovanie aplikácie

Na otestovanie aplikácie som použil Keystroke level model (KLM). KLM predpovedá ako dlho zaberie používateľovi vykonanie akcie v používateľskom rozhraní počítača. KLM bol publikovaný v roku 1980 a neskôr zahrnutý knihe The Psychology of Human-Computer Interaction, ktorá je jednou z najpopulárnejších kníh v obore HCI (interakcia človek - počítač) [18].
KLM rozdeľuje interakciu s používateľským rozhraním na 6 operácií:

- **K** - stlačenie tlačidla alebo klávesu.
- **P** - presunutie kurzora (myši) na položku na obrazovke.
- **H** - presun ruky z jedného vstupného zariadenia na druhé (napríklad klávesnica $< - >$ myš).
- **D** - manuálne nakreslenie čiary myšou na obrazovke (ako napríklad pri označovaní položiek "lasom" na obrazovke).
- **M** - mentálna príprava/rozmýšľanie pred vykonaním akcie.
- **R** - čakanie na odpoveď systému.

Každá operácia má zadefinovaný čas, ktorý trvá. Pri odhadovaní času trvania interakcie týmto modelom interakciu rozdelíme na sekvenciu šiestich základných operácií a sčítame čas všetkých týchto operácií v sekvencii. Pri zostavovaní sekvencie základných operácií sa musíme riadiť následujúcimi pravidlami[18]:

- Vložte **M** pred všetky **K**, ktoré nie sú súčasťou samotného príkazu ale zadávajú argumenty; Vložte **M** pred všetky**P** ktoré vyberajú príkazy, nie argumenty.
- Ak je operátor po **M** plne očakávaný, vymažte M.
- Vymažte všetky M, okrem prvého **M** v každej sekvencii, ktorá tvorí jednotný kognitívny celok.
- Ak je **K** nepotrebný operátor (ako napríklad **K** na ukončenie príkazu následujúce po **K** na zadanie argumentu) odstráňte **M** pred týmto **K**.
- Ak **K** ukončuje konštantný reťazec (napríklad názov príkazu), vymažte **M** pred týmto **K**. Ak **K** ukončuje premenlivý reťazec (napríklad argumenty príkazu), tak **M** pred týmto **K** ponechajte.

Časy jednotlivých operácií som zvolil podľa odporúčaní z [18] ako následujúce:

- **K** = 0.1 s (na myši, klávesnicu tu nepoužívame)
- **P** = 1.1 s
- **H** - tu nepoužívame, všetko prebieha s pomocou myši
- **D** - nepoužívame, nič nie je potrebné označovať ani kresliť
- **M** = 1.35 s

- **R** = 0.05 s

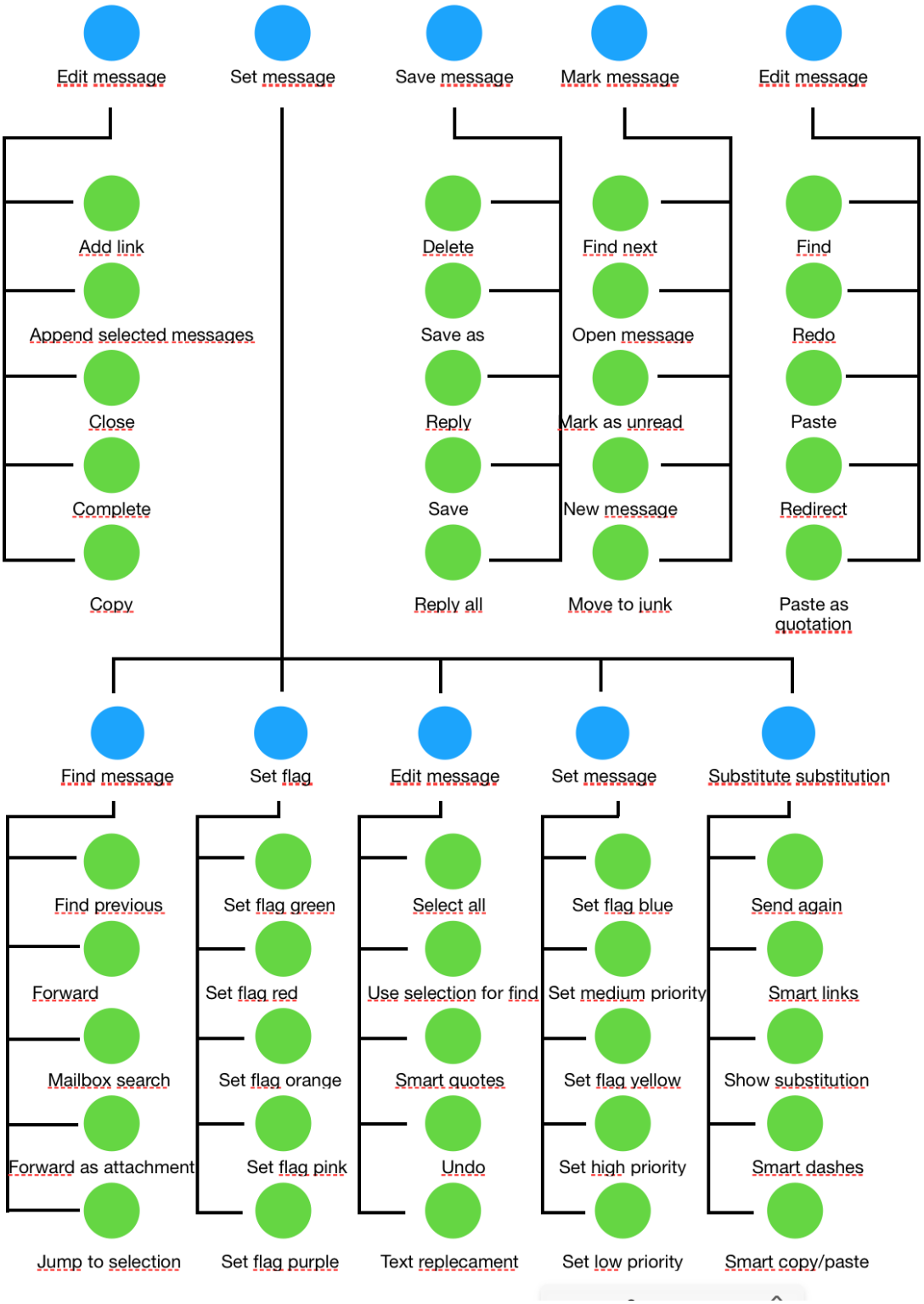
Testovanie prebiehalo na korpuse 20 Newsgroups [7]. Najprv bola otestovaná neadaptovaná aplikácia. Pre každé tlačidlo bola zaznamenaná jeho hĺbka v strome, jeho váha v Mishchenkovom frameworku, percento váhy pripadajúce na každé tlačidlo a čas potrebný na dosiahnutie tohto tlačidla. Čas potrebný na dosiahnutie tlačidla bol vypočítaný KLM modelom podľa vzorca:
$M + K + (h\acute{l}bka + 1) \cdot (P + R)$
Výsledky pre neadaptovanú aplikáciu sú v tabuľkách C.8 , C.9. Tabuľky sú zahrnuté v obrázkovom formáte kvôli problémom so sadzobným systémom. Tabuľky nájdete vo formáte Libre Office v prílohe k tejto práci. Ku tabuľke je zahrnutý obrázok 5.2 stromovej štruktúry menu vytvorenej Mishchenkovým frameworkom (modrá farba je uzol, zelená farba je list).

Následne som korpus rozdelil na dve časti. V každej z dvoch častí bolo 10 kategórií, každá po 500 dokumentoch. Ku každej časti som zvolil množinu tlačidiel na ktoré bude používateľ klikať. Zvolené tlačidlá sú v tabuľkách 5.4 pre časť 1 a 5.5 pre časť 2.

Z každej časti sa zobrala prvá polovica všetkých jej dokumentov a použila sa na predtrénovanie frameworku s klikaním tlačidiel v každej časti podľa zoznamu vyššie. Následne bola druhá polovica každej časti použitá na otestovanie efektivity. Pri každom testovanom dokumente sa vypočítala pravdepodobnosť pre každé tlačidlo zvlášť. Tieto pravdepodobnosti jednotlivých tlačidiel vypočítané z jednotlivých dokumentov boli potom použité na vypočítanie aritmetického priemeru pravdepodobnosti každého tlačidla pre časť 1 a časť 2. Z tohto priemeru bola potom vypočítaná priemerná frekvencia každého tlačidla a táto frekvencia predaná Mishchenkovmu frameworku, ktorý vykonal adaptáciu menu. Stromové štruktúry sú znázornené na obrázku 5.3 pre časť 1 a obrázku 5.4 pre časť 2. Výsledky testovania po adaptácií sú v tabuľkách C.10, C.11, C.12 pre časť 1 a tabuľkách C.13, C.14, C.15 pre časť 2. Tabuľky sú zahrnuté v obrázkovom formáte kvôli problémom so sadzobným systémom. Tabuľky nájdete vo formáte Libre Office v prílohe k tejto práci.

Názov tlačidla	frekvencia
Set Flag Red	2
Set Flag Blue	8
Reply	1
Select All	4
Set High Priority	1
New Message	3
Mark as Unread	4

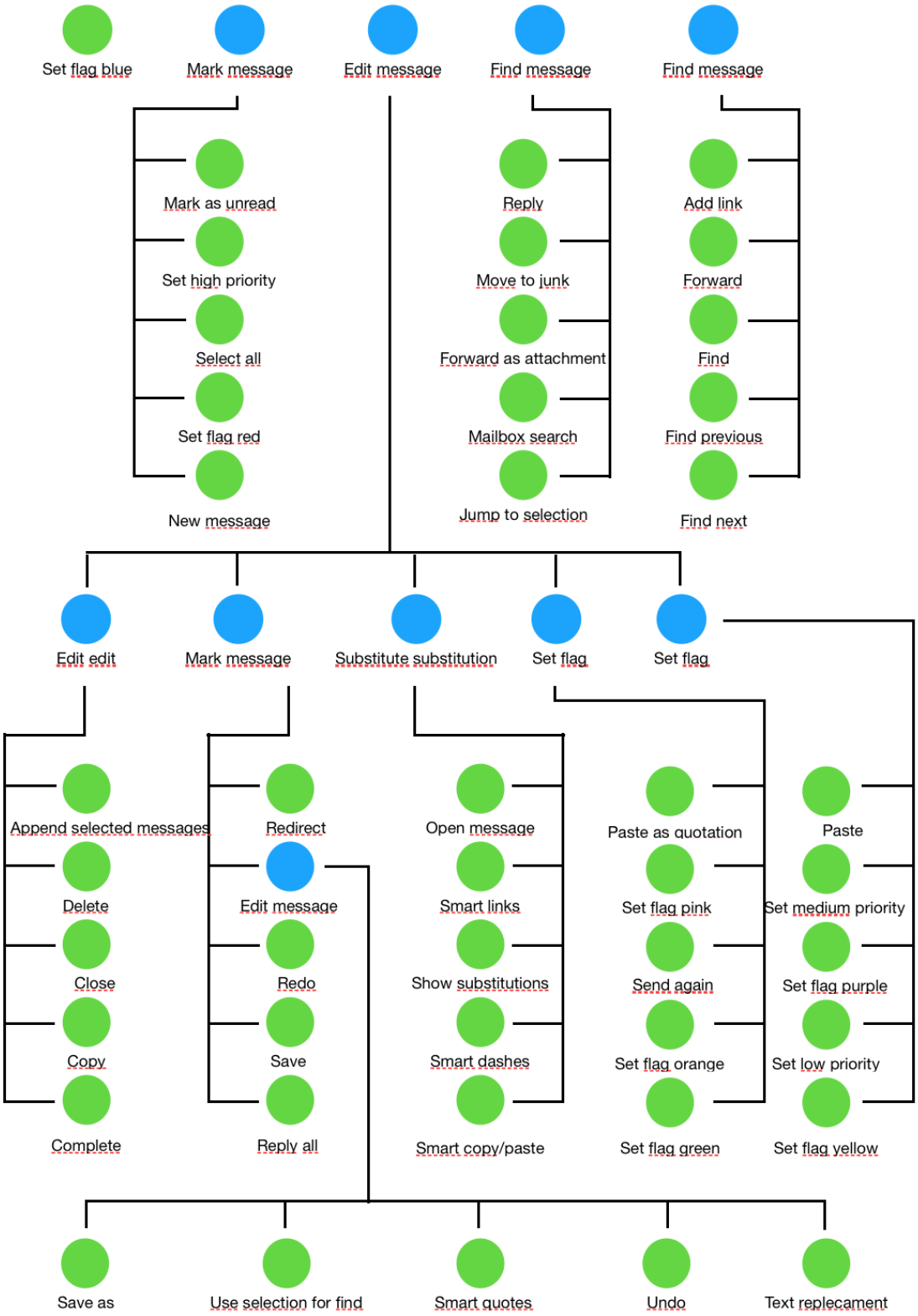
Tabulka 5.4. Frekvencie klikania pre časť 1.



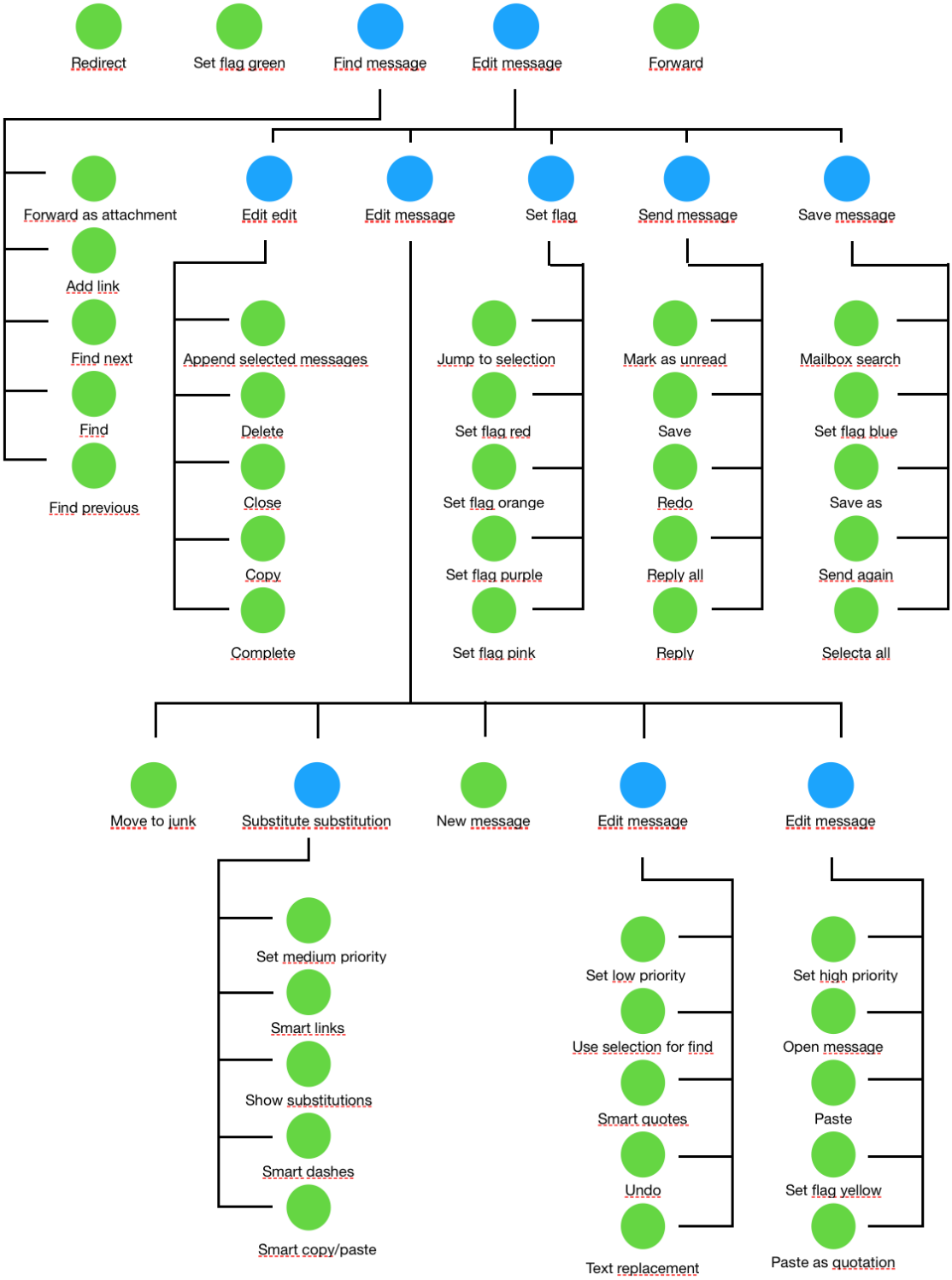
Obrázek 5.2. Štruktúra menu pred adaptáciou.

Názov tlačidla	frekvencia
Set Flag Green	1
Forward	2
Redirect	1
Find	8
Find Next	4
Find Previous	4
Forward as Attachment	3

Tabulka 5.5. Frekvencie klikania pre časť 2.



Obrázek 5.3. Priemerná štruktúra menu po adaptácií pre časť 1.



Obrázek 5.4. Priemerná štruktúra menu po adaptácií pre časť 2.

Vidíme, že pri oboch častiach boli používané tlačidlá presunuté do vyššej časti stromu, kde prístup k nim trvá menej času. V dôsledku presunu týchto tlačidiel nahor museli byť ale nejaké nepoužívané tlačidlá presunutú nadol, čo spôsobilo zväčšenie množstva stupňov v stromoch. V oboch prípad teda adaptácia zjednodušila

prístup k často používaným tlačidlám ale zároveň zhoršila prístup k nepoužívaným tlačidlám.

Kapitola 6
Návrh ďalších rozšírení

Pre budúcu prácu na frameworku sa dajú navrhnúť tieto rozšírenia:

- Pridanie podpory pre viac používateľov. Framework by obsahoval viac úložísk (jednu verziu pre každého používateľa) a prepínal by sa medzi nimi na základe mena používateľa predaného funkciám hlavného rozhrania.
- Pridanie viacerých algoritmov, napríklad k-nearest neighbor.
- Pridanie konfigurovateľného algoritmu na automatické hľadanie optimálnych parametrov algoritmov (počet pamätaných feature, parameter C pre SVM…).

Kapitola 7
Záver

Cieľom bakalárskej práce bolo vytvoriť framework na adaptáciu používateľského rozhrania, založený na analýze textových súborov. V rámci rešerše bola preskúmaná problematika analýzy textových súborov a niekoľko algoritmov na klasifikovanie textových súborov. Následne bol navrhnutý a implementovaný framework v jazyku java používajúci support vector algoritmus a naivnú bayesovskú analýzu. Tento framework môže byť použitý na vyberanie prvkov rozhrania, ktoré budú s veľkou pravdepodobnosťou použité vzhľadom ku súboru, s ktorým pracuje používateľ. Následne bola použitá existujúca bakalárska práca M. Mishchenka "Aspektově orientovaný vývoj adaptivní struktury aplikace pro Java EE aplikace" [1] na vývoj testovacej aplikácie používajúcej tento framework a Mishchenkov framework na adaptáciu položiek v menu. Táto aplikácia bola otestovaná pomocou techniky keystroke level model [18]. Testovanie ukázalo, že po adaptácii sa zlepšil prístup k často používaným položkám menu, avšak zhoršil sa prístup k nepoužívaným položkám.

Literatura

[1] Mishchenko, N., 2017. Aspektově orientovaný vývoj adaptivní struktury aplikace pro Java EE aplikace (Bachelor's thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum.).

[2] Langley, P., 1999. User modeling in adaptive interface. In UM99 User Modeling (pp. 357-370). Springer, Vienna.

[3] Langley, P., 1999. User modeling in adaptive interface. In UM99 User Modeling (pp. 357-370). Springer, Vienna.

[4] Papatheodorou, C., 1999, July. Machine learning in user modeling. In Advanced Course on Artificial Intelligence (pp. 286-294). Springer, Berlin, Heidelberg.

[5] Feldman, R. and Sanger, J., 2007. The text mining handbook: advanced approaches in analyzing unstructured data. Cambridge university press.

[6] Reuters-21578 Text Categorization Collection Data Set. [online], [cit. 1.5. 2019]. Dostupné z:
https://archive.ics.uci.edu/ml/datasets/
Reuters-21578+Text+Categorization+Collection

[7] Twenty Newsgroups Data Set. [online], [cit. 1.5. 2019]. Dostupné z:
https://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups

[8] TREC 2005 Spam Track. [online], [cit. 1.5. 2019]. Dostupné z:

https://plg.uwaterloo.ca/ gvcormac/trecspamtrack05/

[9] Schohn, G. and Cohn, D., 2000, June. Less is more: Active learning with support vector machines. In ICML (Vol. 2, No. 4, p. 6).

[10] KROHA, P. Text Mining, 2015, Prednáškové materiály.

[11] DAN JURAFSKY. Text Classification and Naive Bayes. [online], [cit. 1.5. 2019]. Dostupné z: https://web.stanford.edu/class/cs124/lec/naivebayes.pdf

[12] Hsu, C.W., Chang, C.C. and Lin, C.J., 2003. A practical guide to support vector classification.

[13] LIBSVM – A Library for Support Vector Machines. [online] [cit. 1.5. 2019]. Dostupné z: https://www.csie.ntu.edu.tw/ cjlin/libsvm/

[14] Joachims, T., 1998, April. Text categorization with support vector machines: Learning with many relevant features. In European conference on machine learning (pp. 137-142). Springer, Berlin, Heidelberg.

[15] Chen, J., Huang, H., Tian, S. and Qu, Y., 2009. Feature selection for text classification with Naive Bayes. Expert Systems with Applications, 36(3), pp.5432-5435.

[16] liblinear-java. [online], [cit. 1.5. 2019]. Dostupné z:
https://liblinear.bwaldvogel.de

[17] Dokumentácia štandardnej knižnice jazyka java 8. [online], [cit. 1.5. 2019] Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html

[18] Card, S.K., 2018. The psychology of human-computer interaction. CRC Press

[19] Larkey, L.S. and Croft, W.B., 1996, August. Combining classifiers in text categorization. In SIGIR (Vol. 96, pp. 289-297).

Příloha A
Zadanie práce

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Sivoň** Jméno: **Michal** Osobní číslo: **453583**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Softwarové systémy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Adaptace uživatelského rozhraní založená na analýze textových souborů

Název bakalářské práce anglicky:

Pokyny pro vypracování:

Pomocí technik analýzy obsahu textových souborů navrhnu framework, který přizpůsobí uživatelské rozhraní aplikace vzhledem k souborům, se kterými právě uživatel pracuje. Framework bude používat Bayesovskou analýzu [3] a support vector algoritmus [2].
Cíle práce:
1. Prozkoumat problematiku analýzy obsahu textových souborů pomocí support vector algoritmu[2] a Bayesovské analýzy[3].
2. Prozkoumat možnosti adaptace rozhraní na základě souborů, se kterými uživatel pracuje.
3. Navrhnout a implementovat adaptivní framework.
4. Využít framework k vytvoření testovací aplikace, a tuto aplikaci otestovat.
5. Vyhodnotit testy, výhody a možné omezení.

Seznam doporučené literatury:

[1] KROHA, P. Text Mining, 2015
[2] CHRISTOPHER D. MANNING, PRABHAKAR RAGHAVAN a HINRICH SCHÜTZE. Introduction to Information Retrieval. Cambridge University Press, 2008. ISBN: 0521865719. https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-and-machine-learning-on-documents-1.html
[3] DAN JURAFSKY. Text Classification and Naïve Bayes. https://web.stanford.edu/class/cs124/lec/naivebayes.pdf
[4] ŠEBEK, J. a K. RICHTA. Usage of Aspect-Oriented Programming in Adaptive Application Structure. In: New Trends in Databases and Information Systems. 20th East-European Conference on Advances in Databases and Information Systems, Praha, 2016-08-28/2016-08-31. Wien: Springer, 2016. p. 217-222. 637. ISSN 1865-0929. ISBN 978-3-319-44065-1. DOI 10.1007/978-3-319-44066-8_22.
[5] ŠEBEK, J. and K. RICHTA. Impact of User?s Emotion on Software Adaptation. In: RICHTA, K., P MORAVEC, and J. ŠEBEK, eds. DATESO 2017. DATESO 2017 - Data, Texty, Specifikace a Objekty, Rančířov, 2017-04-10/2017-04-12. Praha: Česká technika - nakladatelství ČVUT, 2017. pp. 1-14. ISBN 978-80-01-06138-1. Available from: http://www.cs.vsb.cz/dateso/2017/

Obrázek A.1. Zadanie práce (1/2).

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek, kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2019** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

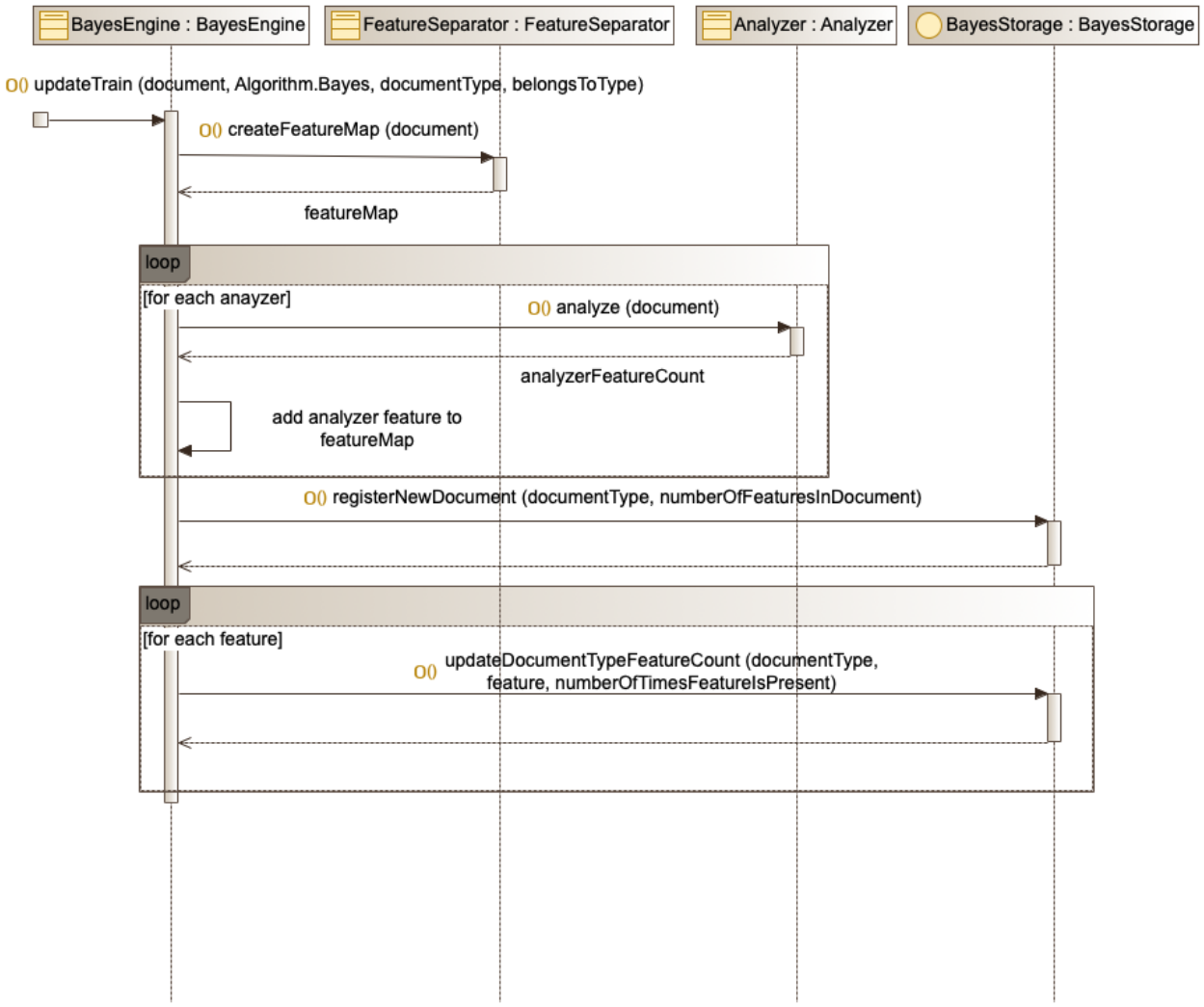
Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání	Podpis studenta

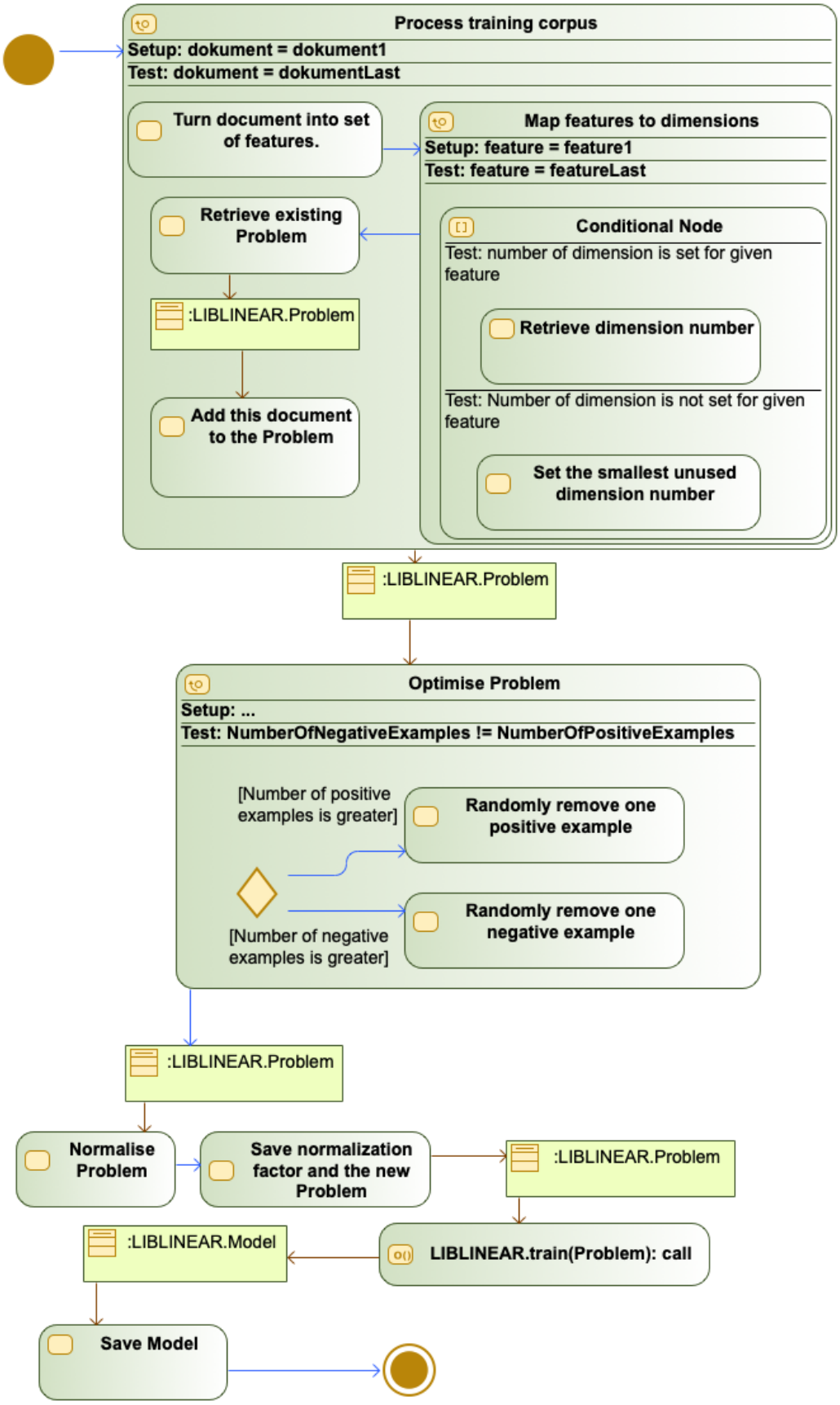
Obrázek A.2. Zadanie práce (2/2).

Příloha B
Diagramy

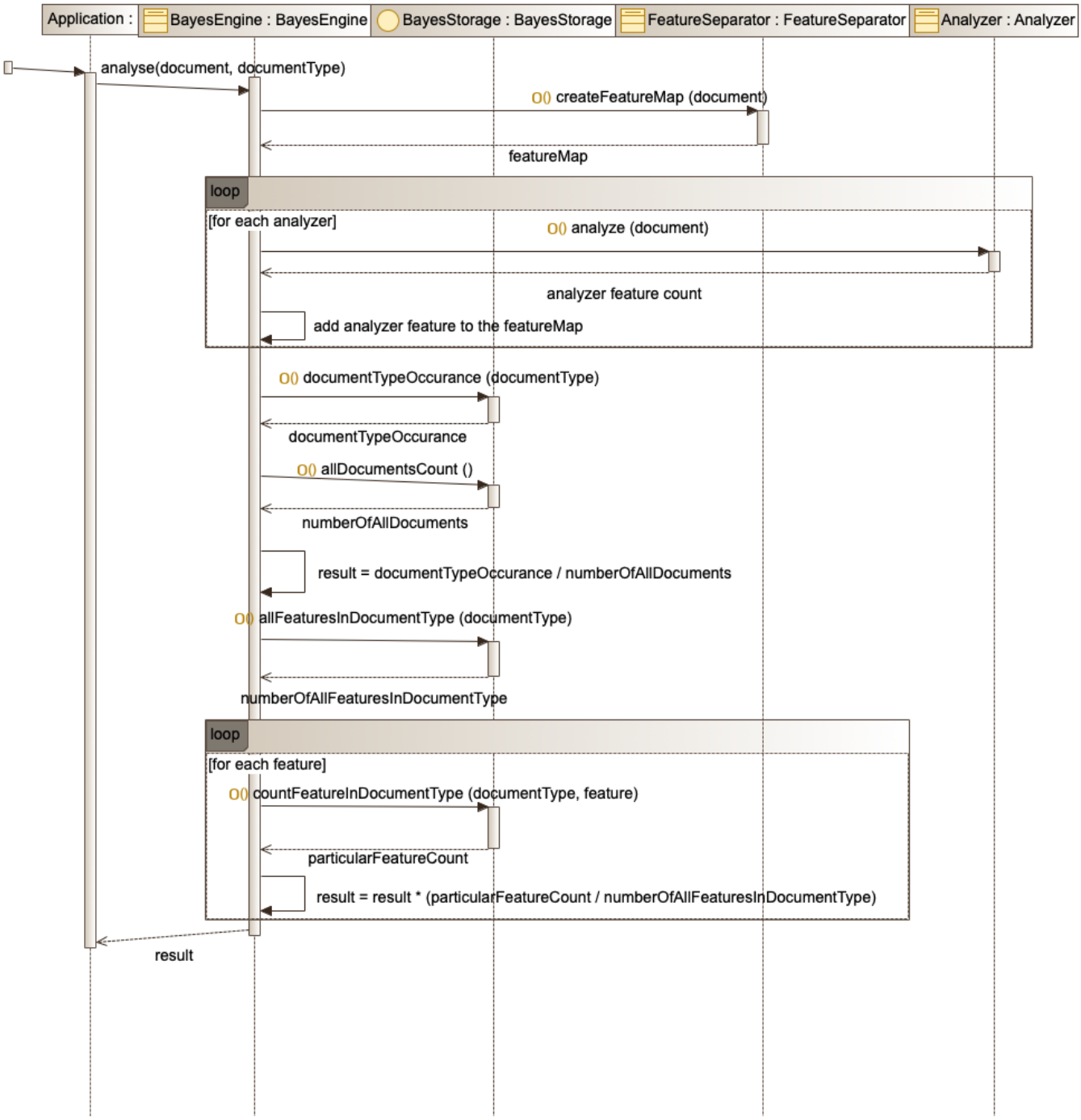
Táto príloha obsahuje UML diagramy, ktoré boli príliš veľké na zahrnutie v texte.



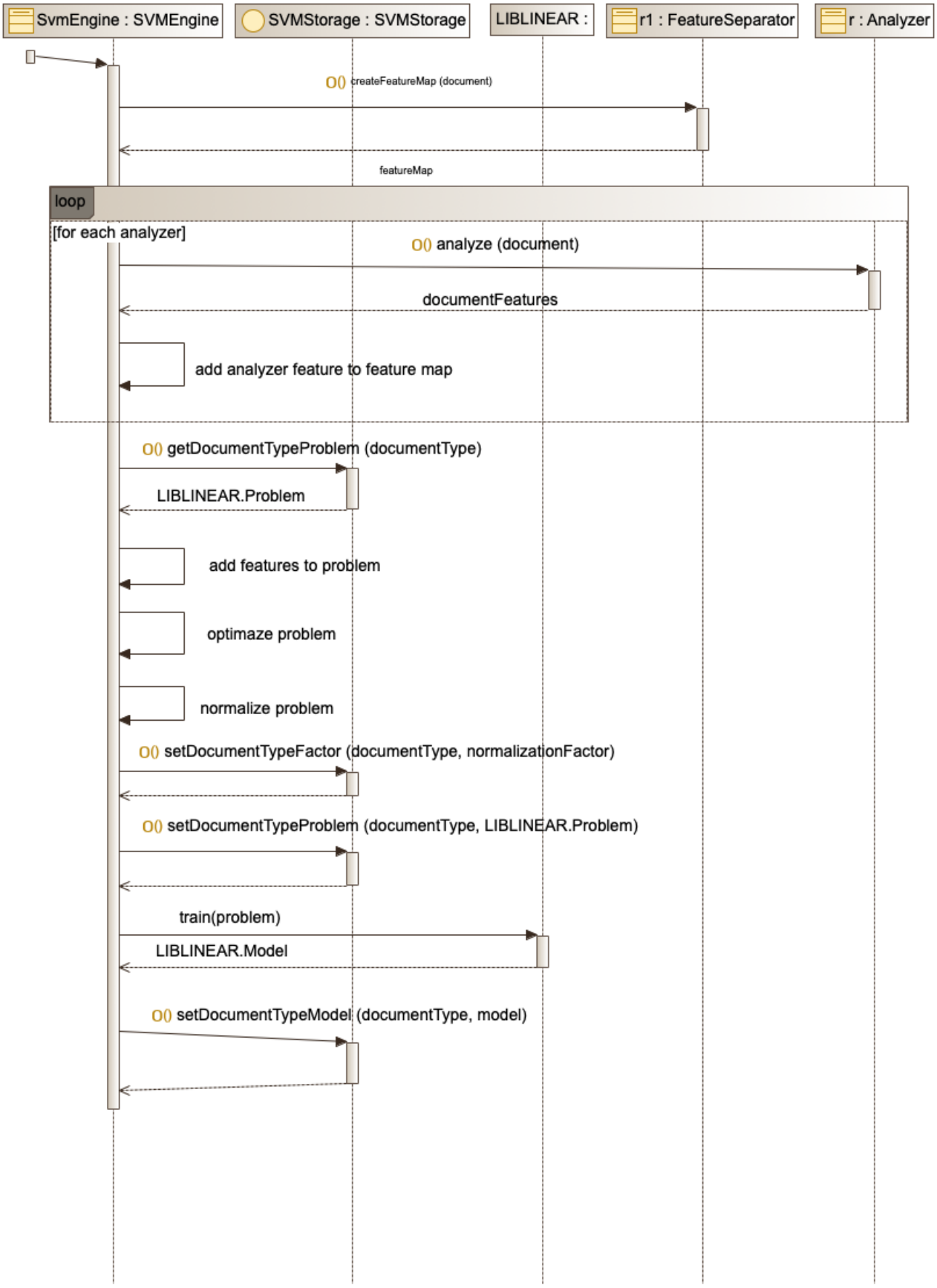
Obrázek B.4. Sekvenčný diagram trénovania bayesovského klasifikátoru.



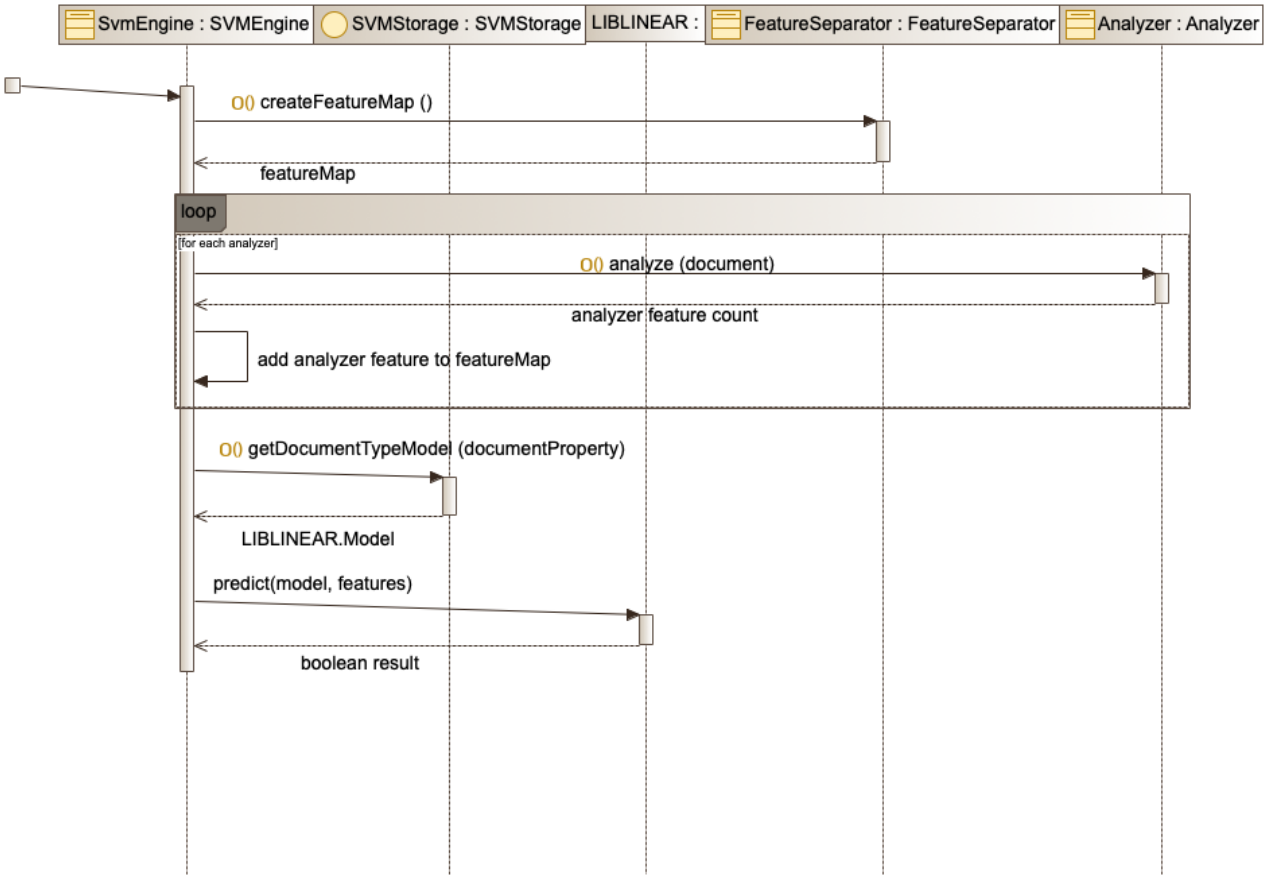
Obrázek B.3. Trénovanie SVM klasifikátoru.



Obrázek B.5. Sekvenčný diagram klasifikovania bayesovským klasifikátorom.



Obrázek B.6. Sekvenčný diagram trénovania SVM klasifikátoru.



Obrázek B.7. Sekvenčný diagram klasifikovania SVM klasifikátorom.

Příloha C
Tabuľky testovania aplikácie

Názov položky	Váha	Počet krokov ku položke	Percento celkovej váhy	Potrebný čas [s]
Add link	50	1	~2,2%	3,7
Append selected messages	50	1	~2,2%	3,7
Close	50	1	~2,2%	3,7
Complete	50	1	~2,2%	3,7
Copy	50	1	~2,2%	3,7
Delete	50	1	~2,2%	3,7
Find	50	1	~2,2%	3,7
Find next	50	1	~2,2%	3,7
Mark as unread	50	1	~2,2%	3,7
Move to junk	50	1	~2,2%	3,7
New message	50	1	~2,2%	3,7
Open message	50	1	~2,2%	3,7
Paste	50	1	~2,2%	3,7
Paste as quotation	50	1	~2,2%	3,7
Redirect	50	1	~2,2%	3,7
Redo	50	1	~2,2%	3,7
Reply	50	1	~2,2%	3,7
Reply all	50	1	~2,2%	3,7
Save	50	1	~2,2%	3,7
Save as	50	1	~2,2%	3,7
Find previous	50	2	~2,2%	4,85
Forward	50	2	~2,2%	4,85
Forward as attachment	50	2	~2,2%	4,85
Jump to sleection	50	2	~2,2%	4,85
Mailbox search	50	2	~2,2%	4,85
Select all	50	2	~2,2%	4,85
Sen again	50	2	~2,2%	4,85
Set flag blue	50	2	~2,2%	4,85
Set flag green	50	2	~2,2%	4,85

Obrázek C.8. Výsledky testovania aplikácie pred adaptáciou. (1/2)

Názov položky	Váha	Počet krokov ku položke	Percento celkovej váhy	Potrebný čas [s]
Set flag orange	50	2	~2,2%	4,85
Set flag pink	50	2	~2,2%	4,85
Set flag purple	50	2	~2,2%	4,85
Set flag red	50	2	~2,2%	4,85
Set flag yellow	50	2	~2,2%	4,85
Set high priority	50	2	~2,2%	4,85
Set low priority	50	2	~2,2%	4,85
Set medium priority	50	2	~2,2%	4,85
Show substitution	50	2	~2,2%	4,85
Smart copy/paste	50	2	~2,2%	4,85
Smart dashes	50	2	~2,2%	4,85
Smart links	50	2	~2,2%	4,85
Smart quotes	50	2	~2,2%	4,85
Text replacement	50	2	~2,2%	4,85
Undo	50	2	~2,2%	4,85
Use selection for find	50	2	~2,2%	4,85

Obrázek C.9. Výsledky testovania aplikácie pred adaptáciou. (2/2)

Názov položky	Váha	Počet krokov ku položke	Počet krokov ku položke pred adptáciou	Zlepšenie množstva krokov	Percento celkovej váhy	Potrebný čas [s]	Potrebný čas pred adpatáciou [s]	Zlepšenie potrebného času [%]
Set flag blue	135	0	2	2	5,45	2,55	4,85	47,42
Mark as unread	90	1	1	0	3,63	3,7	3,7	0
Select all	90	1	2	1	3,63	3,7	4,85	23,71
New message	78	1	1	0	3,15	3,7	3,7	0
Set flag red	67	1	2	1	2,7	3,7	4,85	23,71
Set high priority	62	1	2	1	2,5	3,7	4,85	23,71
Reply	56	1	1	0	2,26	3,7	3,7	0
Add link	50	1	1	0	2,02	3,7	3,7	0
Find	50	1	1	0	2,02	3,7	3,7	0
Find next	50	1	1	0	2,02	3,7	3,7	0
Find previous	50	1	2	1	2,02	3,7	4,85	23,71
Forward	50	1	2	1	2,02	3,7	4,85	23,71
Forward as attachment	50	1	2	1	2,02	3,7	4,85	23,71
Jump to sleection	50	1	2	1	2,02	3,7	4,85	23,71
Mailbox search	50	1	2	1	2,02	3,7	4,85	23,71
Move to junk	50	1	1	0	2,02	3,7	3,7	0
Append selected messages	50	2	1	-1	2,02	4,85	3,7	-31,08
Close	50	2	1	-1	2,02	4,85	3,7	-31,08
Complete	50	2	1	-1	2,02	4,85	3,7	-31,08

Obrázek C.10. Výsledky testovania aplikácie po adaptácií pre časť 1. (1/3)

Názov položky	Váha	Počet krokov ku položke	Počet krokov ku položke pred adptáciou	Zlepšenie množstva krokov	Percento celkovej váhy	Potrebný čas [s]	Potrebný čas pred adpatáciou [s]	Zlepšenie potrebného času [%]
Copy	50	2	1	-1	2,02	4,85	3,7	-31,08
Delete	50	2	1	-1	2,02	4,85	3,7	-31,08
Open message	50	2	1	-1	2,02	4,85	3,7	-31,08
Paste	50	2	1	-1	2,02	4,85	3,7	-31,08
Paste as quotation	50	2	1	-1	2,02	4,85	3,7	-31,08
Redirect	50	2	1	-1	2,02	4,85	3,7	-31,08
Redo	50	2	1	-1	2,02	4,85	3,7	-31,08
Reply all	50	2	1	-1	2,02	4,85	3,7	-31,08
Save	50	2	1	-1	2,02	4,85	3,7	-31,08
Sen again	50	2	2	0	2,02	4,85	4,85	0
Set flag green	50	2	2	0	2,02	4,85	4,85	0
Set flag orange	50	2	2	0	2,02	4,85	4,85	0
Set flag pink	50	2	2	0	2,02	4,85	4,85	0
Set flag purple	50	2	2	0	2,02	4,85	4,85	0
Set flag yellow	50	2	2	0	2,02	4,85	4,85	0
Set low priority	50	2	2	0	2,02	4,85	4,85	0
Set medium priority	50	2	2	0	2,02	4,85	4,85	0
Show substitution	50	2	2	0	2,02	4,85	4,85	0
Smart copy/ paste	50	2	2	0	2,02	4,85	4,85	0
Smart dashes	50	2	2	0	2,02	4,85	4,85	0
Smart links	50	2	2	0	2,02	4,85	4,85	0
Save as	50	3	1	-2	2,02	6	3,7	-62,16

Obrázek C.11. Výsledky testovania aplikácie po adaptácií pre časť 1. (2/3)

Názov položky	Váha	Počet krokov ku položke	Počet krokov ku položke pred adptáciou	Zlepšenie množstva krokov	Percento celkovej váhy	Potrebný čas [s]	Potrebný čas pred adpatáciou [s]	Zlepšenie potrebného času [%]
Smart quotes	50	3	2	-1	2,02	6	4,85	-23,71
Text replacement	50	3	2	-1	2,02	6	4,85	-23,71
Undo	50	3	2	-1	2,02	6	4,85	-23,71
Use selection for find	50	3	2	-1	2,02	6	4,85	-23,71

Obrázek C.12. Výsledky testovania aplikácie po adaptácií pre časť 1. (3/3)

Názov položky	Váha	Počet krokov ku položke	Počet krokov ku položke pred adptáciou	Zlepšenie množstva krokov	Percento celkovej váhy	Potrebný čas [s]	Potrebný čas pred adpatáciou [s]	Zlepšenie potrebného času [%]
Redirect	75	0	1	1	3,2	2,55	3,7	31,08
Set flag green	75	0	2	2	3,2	2,55	4,85	47,42
Forward	65	0	2	2	2,77	2,55	4,85	47,42
Forward as attachment	60	1	2	1	2,56	3,7	4,85	23,71
Find next	58	1	1	0	2,47	3,7	3,7	0
Find previous	58	1	2	1	2,47	3,7	4,85	23,71
Find	54	1	1	0	2,3	3,7	3,7	0
Add link	50	1	1	0	2,13	3,7	3,7	0
Append selected messages	50	2	1	-1	2,13	4,85	3,7	-31,08
Close	50	2	1	-1	2,13	4,85	3,7	-31,08
Complete	50	2	1	-1	2,13	4,85	3,7	-31,08
Copy	50	2	1	-1	2,13	4,85	3,7	-31,08
Delete	50	2	1	-1	2,13	4,85	3,7	-31,08
Jump to sleection	50	2	2	0	2,13	4,85	4,85	0
Mailbox search	50	2	2	0	2,13	4,85	4,85	0
Mark as unread	50	2	1	-1	2,13	4,85	3,7	-31,08
Redo	50	2	1	-1	2,13	4,85	3,7	-31,08
Reply	50	2	1	-1	2,13	4,85	3,7	-31,08
Reply all	50	2	1	-1	2,13	4,85	3,7	-31,08
Save	50	2	1	-1	2,13	4,85	3,7	-31,08
Save as	50	2	1	-1	2,13	4,85	3,7	-31,08
Select all	50	2	2	0	2,13	4,85	4,85	0
Sen again	50	2	2	0	2,13	4,85	4,85	0

Obrázek C.13. Výsledky testovania aplikácie po adaptácií pre časť 2. (1/3)

Názov položky	Váha	Počet krokov ku položke	Počet krokov ku položke pred adptáciou	Zlepšenie množstva krokov	Percento celkovej váhy	Potrebný čas [s]	Potrebný čas pred adpatáciou [s]	Zlepšenie potrebného času [%]
Set flag blue	50	2	2	0	2,13	4,85	4,85	0
Set flag orange	50	2	2	0	2,13	4,85	4,85	0
Set flag pink	50	2	2	0	2,13	4,85	4,85	0
Set flag purple	50	2	2	0	2,13	4,85	4,85	0
Set flag red	50	2	2	0	2,13	4,85	4,85	0
Move to junk	50	3	1	-2	2,13	6	3,7	-62,16
New message	50	3	1	-2	2,13	6	3,7	-62,16
Open message	50	4	1	-3	2,13	7,15	3,7	-93,24
Paste	50	4	1	-3	2,13	7,15	3,7	-93,24
Paste as quotation	50	4	1	-3	2,13	7,15	3,7	-93,24
Set flag yellow	50	4	2	-2	2,13	7,15	4,85	-47,42
Set high priority	50	4	2	-2	2,13	7,15	4,85	-47,42
Set low priority	50	4	2	-2	2,13	7,15	4,85	-47,42
Set medium priority	50	4	2	-2	2,13	7,15	4,85	-47,42
Show substitution	50	4	2	-2	2,13	7,15	4,85	-47,42
Smart copy/ paste	50	4	2	-2	2,13	7,15	4,85	-47,42
Smart dashes	50	4	2	-2	2,13	7,15	4,85	-47,42
Smart links	50	4	2	-2	2,13	7,15	4,85	-47,42
Smart quotes	50	4	2	-2	2,13	7,15	4,85	-47,42

Obrázek C.14. Výsledky testovania aplikácie po adaptácií pre časť 2. (2/3)

Názov položky	Váha	Počet krokov ku položke	Počet krokov ku položke pred adptáciou	Zlepšenie množstva krokov	Percento celkovej váhy	Potrebný čas [s]	Potrebný čas pred adpatáciou [s]	Zlepšenie potrebného času [%]
Text replacement	50	4	2	-2	2,13	7,15	4,85	-47,42
Undo	50	4	2	-2	2,13	7,15	4,85	-47,42
Use selection for find	50	4	2	-2	2,13	7,15	4,85	-47,42

Obrázek C.15. Výsledky testovania aplikácie po adaptácií pre časť 2. (3/3)

Příloha D

Zoznam vzorcov