Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Klasifikace uživatelů přistupujících k webovým službám

Plzeň 2017 Jindřich Pouba

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 17. května 2017

Jindřich Pouba

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce, panu Ing. Kamilu Ekšteinovi, Ph.D., za vstřícnost, upřímnost a dobré rady během zpracovávání této práce.

Abstract

This thesis explores methods of classification of web users depending on their behaviour. It emphasizes classification of robots and human web users. There are multiple explored methods, from simple classification based on times of requests to classification based on Markov models. Another part of this thesis is an application that uses and tests these methods. All methods were tested and measured on multiple datasets and their results compared.

Abstrakt

Tato práce prozkoumává metody klasifikaci uživatelů webových služeb podle jejich chování. Důraz je kladen zejména na klasifikaci lidských uživatelů a robotů. Zkoumaných metod je několik, od jednoduchého porovnávání časů přístupů na web ke klasifikátoru na bázi Markovova modelu. Součástí práce je také aplikace, která tyto metody využívá a testuje. Všechny metody v aplikaci byly vyzkoušeny, jejich úspěšnost změřena na několika datasetech a výsledky porovnány mezi sebou.

Obsah

1 Úvod

Cílem této práce je prozkoumat metody klasifikace uživatelů na základě jejich chování na webových stránkách a napsat program, který tyto metody bude využívat k rozdělování uživatelů do tříd. Vytvořený program bude vhodně otestován a úspěšnost jednotlivých metod porovnána mezi sebou.

Každý návštěvník si webové stránky prohlíží jiným způsobem, otevírá různé stránky v různém pořadí a tráví na nich různě dlouhou dobu. Přesto lze návštěvníky shlukovat do skupin s podobným chováním. Například návštěvníci zpravodajského serveru s podobnými zájmy budou nejspíše číst podobné rubriky. Provozovatel internetového obchodu může rozlišovat návštěvníky na ty, co si zboží spíše prohlížejí, a na ty, kteří neváhají s nakupováním.

Další nezanedbatelnou část návštěv webů tvoří roboti, neboli počítačové programy naprogramované za účelem procházení webových stránek a získávání informací v nich obsažených. Tyto roboty můžeme rozdělit na „hodné“ a „zlé“.

Hodní roboti jsou například internetové vyhledávače, které na našem webu zpravidla chceme a které pomáhají náš web zviditelnit. Mezi zlé roboty můžeme zařadit ty, kteří prochází web za účelem získávání informací, které sice jsou veřejné, ale autor je nedává volně k dispozici a má z nich prospěch pouze v případě, že jsou navštíveny na jeho serveru. Jako příklad můžeme uvést robota, který kopíruje novinové články z různých serverů a umist'uje je na svůj vlastní. Autor původního obsahu je tak připraven o zisky například z reklamy na svém serveru. Další možností je procházení obsahu sociálních sítí a ukládání osobních informací o uživatelích. Nehledě na to, že uživatelé sociálních sítí nechtějí, aby se jejich informace dostávaly do rukou cizím lidem, tak pro samotného provozovatele sociální sítě představují takoví roboti pouze zátěž pro servery, ze které nemají žádný užitek.

Důvody, proč se zabývat klasifikací uživatelů, se různí podle prostředí, ve kterém je klasifikátor nasazen. Jeden z důvodů může být například, že majitele webu zajímá jaké druhy uživatelů navštěvují jeho web, aby na základě těchto informací upravil například marketingovou strategii nebo strukturu webu. Dalším důvodem může být bezpečnost, kdy chceme odříznout případné hackerské útoky na web a/nebo „zlé“ roboty, kteří takové útoky mohou provádět. Klasifikaci lidských uživatelů může využít například internetová reklama, která je přesně cílená na různé skupiny uživatelů. Uživateli

se poté zobrazuje reklama na základě skupiny, do které je zařazen, a u které se předpokládá, že má na členy dané skupiny největší vliv (tj. že členové dané skupiny budou nakupovat zboží a služby, které reklama inzeruje).

V této práci se podíváme na možnosti klasifikace uživatelů podle různých kritérií a metod. Kapitola 2 obsahuje definici základních pojmů a přehled prostředí Webu, ve kterém se klasifikace odehrává. Kapitola 3 se zabývá metodami klasifikace a jejich výhodami a nevýhodami v tomto prostředí.

Metody používané v aplikaci jsou jedna po druhé rozebrány v následujících kapitolách. Kapitola 5, se zabývá používanou metodou klasifikace podle typu souborů a koncových bodů na které uživatelé přistupují. Kapitola 6 popisuje pokus o rozdělení návštěvníků jednoduchým způsobem na základě prodlev mezi jednotlivými požadavky na server. Tato metoda rozdělování podle časů je dále rozpracována v kapitole 7, kde jsou použity statistické metody pro určení třídy uživatele.

Kapitola 8 ukazuje možnost použití Markovova modelu pro modelování webového serveru, vytvoření sítě koncových bodů a klasifikace uživatelů podle toho, jak se po dané síti pohybují, tj. v jakém pořadí a na které koncové body přistupují. Tento postup je rozvinut v kapitole 9, kde se ke klasickému Markovovu modelu přidávají údaje o uživateli a jeho historii procházení webu.

Všechny metody jsou otestovány v kapitole 11, ale nejdříve je v kapitole 10 popsán způsob předzpracování testovacích dat a představeny různé datasety, na kterých je testování provedeno.

Doplňková kapitola 12 popisuje metodu pro odhadování a trénování parametrů jednotlivých klasifikátorů.

2 Základní pojmy

2.1 Web

Přeloženo z [2]: „World Wide Web“ (WWW, nebo jednoduše Web) je informační prostor, ve kterém jsou předměty zájmu, nazývané zdroje, identifikovány globálními identifikátory nazývanými *Uniform Rescource Identifier* (zkráceně URI). Za zdroje se nejčastěji považují webové stránky napsané v jazyce *HTML* a protokol pro jejich přenos v počítačové síti je *HTTP* [2].

2.1.1 HTML

HTML je značkovací jazyk pro tvorbu webových stránek (viz dále). Pro účely této práce je nepodstatné rozebírat celý tento jazyk, ovšem je dobré říci, že HTML stránka mimo textu obsahuje ještě *tagy*, jinak značky, které určují formátování textu a jeho pozici nebo do stránky přidávají multimediální obsah (obrázky, hudbu, videa) nebo interaktivní skripty v různých jazycích, z nichž nejrozšířenejší (a jediný podporovaný ve všech nejpoužívanějších prohlížečích [5]) je JavaScript. Velmi důležitým pojmem a značkou na webové stránce je odkaz. Odkazy jsou většinou vizuálně zvýrazněny na stránce a říkají, že když na ně uživatel klikne, dostane se na jinou stránku [3, 4].

Tagy jsou vždy uzavřeny mezi znaky „<“ a „>“ a dělí se na párové a nepárové. Párové tagy musí být ukončeny stejným tagem, ovšem uvozeným lomítkem, např. `<p> text </p>`, kde vidíme, že tag `<p>` je ukončen pomocí tagu `</p>`. Tagy nepárové se neukončují, např. tag `<br>`. Funkce tagu je daná především jeho názvem, ale může být doplněna pomocí atributů. Jako příklad uvedeme tag `<img src='obrazek.jpg'>`, kde atribut `src` uvádí umístění obrázku, který má být zobrazen.

2.1.2 Webová stránka

Pojem webová stránka můžeme chápat více způsoby, například jako jednu stránku napsanou v HTML, ale pro účely této práce budeme jako webovou stránku označovat celý soubor zdrojů (HTML, multimédia, skripty, atd.),

které prohlížeč stáhne po otevření jedné URL adresy (viz dále) [4].

2.1.3 URL a URI

Výrazy *Uniform Resource Identifier* (URI) a *Uniform Resource Locator* (URL) se často zaměňují. V zásadě URI je řetězec znaků, používaný k identifikaci nějakého zdroje dat. URL je podmnožina URI, ve které je přidána podmínka, že v řetězci musí být kromě identifikace zdroje také obsažen mechanismus pro jeho získání. Tím je nejčastěji myšlen protokol, přes který je zdroj dostupný [6].

Takže například `www.stranka.cz/index.html` tvoří URI, ovšem ne URL. Aby vzniklo URL, je nutné přidat jméno protokolu, přes který je možné stránku stáhnout, např. `http://www.stranka.cz/index.html`. Pro účely této práce je dobré poznamenat, že jediný protokol, který je pro nás důležitý je protokol HTTP (rozebrán dále), protože se jedná o základní a nejpoužívanější metodu získávání dat na Webu [2].

2.2 HTTP

HyperText Transfer Protocol, zkráceně HTTP, je protokol aplikační vrstvy podle modelu ISO/OSI pro výměnu informací v distribuované a prolinkované síti. Tedy v síti, kde stránky jsou uloženy na různých fyzicky na sobě nezávislých místech a obsahují v sobě odkazy na další stránky. Je to základní stavební kámen komunikace na Webu [2].

V současné době existují čtyři verze HTTP protokolu označované za lomítkem v názvu protokolu: HTTP/0.9, HTTP/1.0, HTTP/1.1 a HTTP/2 [9, 10]. V této práci se budeme soustředit na verzi HTTP/1.1, jelikož je v současné době nejpoužívanější [10, 11].

HTTP/1.1 je protokol typu požadavek–odpověď pro síťová prostředí typu klient–server. Pro člověka přistupujícího na webovou stránku je klientem webový prohlížeč a serverem se myslí aplikace na cílovém počítači zodpovědná za předkládání webových stránek. Výraz *server* může v případě Webu mít dva významy, jednak jako počítač, na kterém webová aplikace běží, nebo může označovat samotnou aplikaci. V této práci budeme jako server

označovat aplikaci zodpovědnou za odpovědi na požadavky. Celá komunikace je bezestavová, tedy server si neuchovává žádnou informaci o klientech v rámci jednoho spojení. Veškeré údaje potřebné k vyřízení požadavku musí být obsaženy v daném požadavku [7, 8].

Protokol je textový, celá komunikace si vystačí s obsahem ASCII tabulky s výjimkou samotných přenášených binárních souborů. HTTP požadavek (také se používá anglický tvar *request*) má formát:

```
<metoda> <cesta ke zdroji> <http verze>CRLF
<hlavička1>: <hodnota1>CRLF
<hlavička2>: <hodnota2>CRLF
<hlavička3>: <hodnota3>CRLF
.
.
CRLF
<volitelné tělo požadavku>
```

Odpověď na požadavek (ang. *response*) má formát:

```
<http verze> <kód odpovědi> <zpráva odpovědi>CRLF
<hlavička1>: <hodnota1>CRLF
<hlavička2>: <hodnota2>CRLF
<hlavička3>: <hodnota3>CRLF
.
.
CRLF
<volitelné tělo odpovědi>
```

Metody HTTP požadavku jsou:

- `GET` – Používaný pro prosté získání zdroje identifikovaného cestou.
- `POST` – Obsahuje data, která má server zpracovat v kontextu zdroje, identifikovaném cestou. Může jít například o přidání zprávy na nástěnku nebo do diskusního fóra. V některých aplikacích se i POST používá k získávání dat, ovšem zásadní rozdíl mezi POST a GET je ten, že GET by nikdy neměl měnit stav serveru, zatímco POST může.

- `HEAD` – Odpověď na tento požadavek by měla být identická s odpovědí na ten samý GET požadavek, ovšem HEAD vrací pouze hlavičky a nikdy tělo odpovědi.
- `OPTIONS` – Vrací podporované typy HTTP požadavků na daný zdroj. Slouží k ověřování funkcionality serveru nebo například k ověřování přístupových práv na zdroj, pokud request přichází z jiného serveru.
- `PUT` – Požadavek má za cíl buď vytvořit nový zdroj daný cestou nebo ho aktualizovat daty v požadavku.
- `DELETE` – Smaže zdroj definovaný cestou.
- `TRACE` – Odešle zpět požadavek v takovém formátu v jakém byl přijat. Slouží klientovi k porovnání odeslaného a přijatého požadavku a ke sledování změn, které na něm mohly vzniknout po cestě.
- `CONNECT` – Vytváří TCP/IP tunel používaný hlavně v případě zašifrovaného spojení přes nezašifrovanou proxy (viz dále).

Hlavičky požadavku i odpovědi slouží k přidávání dalších informací o požadavku nebo odpovědi. V případě požadavku může jít například o typ klienta, který data požaduje, seznam přijatelných kódování, jazyk a prostředí klienta, atd. Hlavičky požadavku jsou volitelné s jedinou výjimkou, kterou je hlavička `Host`. Host určuje, na který hostovaný web na serveru je požadavek směřován. Cesta v hlavičce požadavku je většinou relativní a tento další prvek je tedy pro server nutný ke správné identifikaci zdroje [7, 8].

Hlavičky odpovědi určují například délku odpovědi, její typ a kódování, typ serveru, který data odeslal nebo čas odeslání. V odpovědi nejsou přímo povinné žádné hlavičky, ovšem ke správnému fungování webového serveru je nutné v odpovědi na požadavky, které vracejí nějaká data, sdělit, kde požadovaná data končí. Tedy specifikovat jejich délku (hlavička `Content-Length`), jejich kódování při přenosu (hlavička `Transport-Encoding`) nebo specifikovat, že data končí v okamžiku ukončení TCP spojení (hlavička `Connection` s hodnotou `close`) [7, 8].

Cesta ze zdroji většinou odkazuje na jeden soubor, ovšem může obsahovat další informace, nazývané *parametry*, které ho specifikují. Parametry jsou uvedeny v cestě za znakem „`?`“ a ve formátu `klíč=hodnota`. Jednotlivé parametry se oddělují znaky „`&`“ nebo „`;`“. Například v požadavku

```
GET /stranka?jazyk=en&styl=tisk
```

uživatel specifikuje zdroj `/stranka`, ovšem přidává další parametry, jako anglický jazyk a styl stránky ve verzi pro tisk. Zda při klasifikaci parametry v cestě používat nebo je vynechávat řeší kapitola 10. Pro větší přehlednost budou ve většině uvedených příkladů parametry vynechávány.

Typický požadavek na webovou i s odpovědí stránku má tedy podobu (znaky `CR` a `LF` na koncích řádků byly tentokrát vynechány):

```
GET index.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.stranka.cz
```

Odpověď na tento požadavek může vypadat:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html

<html>
   <body>

   <h1>Hello, World!</h1>

   </body>
</html>
```

Kódy odpovědí doplněné zprávami udávají, jak byl požadavek vyřízen. V případě procházení webu je nejčastější odpovědí `200 OK`, což udává kladné vyřízení požadavku následované požadovaným zdrojem. Dalším kódem, který můžeme vidět v odpovědi na požadavek, je kód `404` a zpráva `Not found`. Kód říká, že požadovaný zdroj nebyl na serveru nalezen. Kód `401 Unauthorized` indikuje, že k požadovanému zdroji nemáme povolen přístup. Jako poslední příklad uvedeme kód `500 Internal Server Error`, který uvádí, že server narazil na neznámý problém nebo chybu, která mu zabránila ve vyprodukování odpovědi na požadavek.

Kódů je celá řada, ovšem pro účely této práce není třeba je uvádět ani dále rozebírat, vypovídají většinou o chování serveru a metodách, jak zpracovává požadavky. V této práci nás ale více zajímá chování uživatelů, a tedy v případě komunikace se serverem se zaměříme více na požadavky než na odpovědi na ně. Seznam kódů proto není úplným výčtem, ale spíše základním přehledem nejčastějších případů [12].

Pojem „koncový bod“

Koncový bod je takový zápis cíle požadavku, kterým je přesně identifikovatelný bod v programu nebo podprogram, který přímo řeší takový požadavek. Většinou neznáme přesné chování serveru, ale budeme předpokládat, že různé typy požadavků mají různé chování na serveru a tedy jsou obsluhovány různými podprogramy. Pro účely této práce definujeme koncový bod jako typ požadavku společně s cestou a verzí HTTP protokolu. Tedy celou první řádku HTTP požadavku.

2.2.1 HTTPS

HTTPS, jinak nazývané *HTTP over TLS*, *HTTP over SSL* nebo *HTTP Secure* je protokol pro zabezpečení HTTP komunikací. V zásadě se jedná o protokol HTTP zabalený do šifrování SSL nebo TLS (viz dále). Pokud klient navazuje HTTPS spojení, nejdříve naváže šifrované SSL/TLS spojení a poté tímto šifrovaným tunelem komunikuje přes HTTP [14].

Transport Layer Security (TLS) a jeho předchozí verze *Secure Sockets Layer (SSL)* jsou kryptografické protokoly sloužící k zabezpečení spojení v počítačové síti. Takto šifrovaná spojení mají zajištěné:

- *soukromí* – takže ani pozorovatel stojící mezi klientem a serverem nezjistí obsah komunikace,
- *integritu dat* – kdy útočník není schopný modifikovat obsah zpráv tak, aby to klient ani server nebyli schopni rozpoznat
- a nejčastěji v případe webových služeb je také zajištěna *identita serveru* – takže klient má jistotu, že komunikuje s požadovanou aplikací [17].

Popis fungování těchto protokolů není pro účely této práce důležitý a jelikož na straně serveru (kde se námi zkoumaná klasfikace odehrává) musí dojít k rozšifrování spojení a zpracování požadavku v protokolu HTTP, budeme i v případě šifrovaného spojení vždy v této práci za spojení považovat vždy pouze HTTP spojení po rozšifrování.

2.3 Proxy server

Proxy server je takový server, který jedná jako prostředník mezi klientem a dalším serverem. Klient posílá své požadavky na proxy server, který je vyřídí a odpověď vrací opět klientovi. Proxy může fungovat pouze jako prostředník, který požadavky neupravuje a například jen zajišťuje spojení z intranetu do Internetu. Popřípadě funguje jako *cache*, která časté požadavky ukládá do svojí paměti a na opakující se požadavek už nemusí kontaktovat cílový server. Mezi další funkce proxy serverů může patřit úprava požadavků a odpovědí, sloužící například k anonymizaci uživatelů nebo blokování reklamy na Webu [13].

2.3.1 Reverzní proxy

Speciální příklad proxy serveru je *reverzní proxy*, která narozdíl od ostatních typů proxy serverů nezastupuje klientskou stranu, ale místo toho se nachází těsně před cílovým serverem. Klient tak ani neví, že přes nějakou proxy vůbec prochází. Takto umístěná proxy nejčastěji poskytuje úlevu od zátěže cílovému serveru, například tím, že ukládá odpovědi do své cache, dekomprimuje požadavky a naopak komprimuje odpovědi nebo se stará o navazování zabezpečených TLS spojení.

Další možnou a pro účely této práce nejzajímavější úlohou reverzní proxy je zabezpečení samotných aplikačních serverů. Proxy může vyhodnocovat požadavky a v případě podezření na hackerský nebo DoS (*Denial of Service*, nejčastěji jde o zahlcení serveru falešnými požadavky, takže nezbyde výpočetní výkon na zpracovávání reálných požadavků a server se zvenčí jeví jako nefunkční) útok požadavky zablokovat nebo detailněji logovat, což usnadní pozdější analýzu.

2.4 Uživatel

Za uživatele budeme považovat buď člověka používajícího webový prohlížeč k přistupování na stránky nebo robota, který stránky prochází, stahuje a nějak zpracovává.

U lidských uživatelů nás nejvíce budou zajímat normální uživatelé, přistupující z jednoho počítače (z jedné IP adresy) používající normální a neupravený prohlížeč. Slovem neupravený zde máme na mysli, že nepoužívají externí programy, metody nebo pluginy do prohlížeče, které slouží k anonymizaci uživatele a/nebo zmatení serveru. Mazi takové postupy řadíme použití *Virtual Private Network (VPN)* nebo systému *The Onion Router (Tor)*, které v tomto případě slouží ke skrytí identity uživatele. Jak tyto programy a postupy ztěžují klasifikaci uživatelů je více rozepsáno v sekci 2.5 – Návstěva webu.

Co se týče robotů, máme na mysli programy, které stahují webové stránky a zpracovávají z nich data. Ovšem přidáme další podmínku, že takové programy se na stránce umějí orientovat, a to zejména v tom smyslu, že dokáží z webové stránky získat odkazy na další stránky a ty poté opět stáhnout a zpracovat. Za roboty budeme také považovat specializované programy napsané za účelem získávání dat z nějaké stránky, například příspěvků z diskusních fór. Takový robot nemusí umět ze stránky rozpoznávat hypertextové odkazy, ale přesto je napsán za účelem získávání dat z jednoho serveru a dokáže správně číst a analyzovat obdržená data.

Aplikace, která je součástí této práce, se neomezuje na klasifikaci pouze těchto nejčastějších případů a v případě použití správných trénovacích dat by se dala použít i například na detekci uživatelů používajících specifické anonymizační nástroje nebo robotů, kteří jenom ověřují stav serveru a získaná data je v podstatě nezajímají. Tato práce se ovšem zaměřuje na nejčastější případy klasifikace uživatelů, tedy rozdělování lidských uživatelů do skupin a separaci robotů od lidí. Ostatní okrajové případy nebyly testovány také díky absenci označkovaných testovacích dat.

2.5 Návstěva webu

Interakce uživatele s webovou stránkou se většinou neomezuje pouze na jeden HTTP požadavek a jednu odpověď. V duchu toho, jak byl Internet a hypertext navržen, si uživatel prohlíží stránku plnou odkazů na další stránky, na nějaký klikne, což vede k dalšímu požadavku na server a další stránce. Tímto způsobem uživatel pokračuje, dokud nenajde informaci, kterou hledá nebo jiným způsobem nesplní to, co na webový server přišel udělat.

Dalším způsobem, jak vznikají nové požadavky na server, jsou požadavky od samotné stránky. V prvních letech a prvních verzích webu se opravdu jedna stránka rovnala jednomu požadavku, ovšem dnes jsou stránky složeny z několika dalších zdrojů, z nichž nejčastější jsou multimédia, formátovací kaskádové styly a skripty. Ačkoliv je možné tyto další zdroje vložit přímo do HTML stránky, z důvodu přehlednosti a znovupoužitelnosti se ukládají jako samostatné soubory a do stránek vkládají pomocí tagů a odkazů v nich obsažených.

Například tag `<img ...>` udává, že na tomto místě se na stránce má nacházet obrázek, většinou určený atributem `src`. Internetový prohlížeč si tedy o obrázek musí požádat dalším HTTP požadavkem a může ho zobrazit až poté, co je požadavek kladně vyřízen. Stejný proces se opakuje pro kaskádové formátovací styly, skripty a všechny další zdroje, které prohlížeč potřebuje ke správnému zobrazení stránky.

Dalším způsobem, kterým prohlížeč generuje požadavky na server bez přičinění uživatele, jsou samotné skripty. JavaScript, který je většinou zodpovědný za interaktivitu stránky může například po kliknutí na tlačítko přes protokol HTTP odeslat data z formuláře bez načtení nové stránky. Popřípadě může každých pár sekund kontrolovat změny na stránce (například nové příspěvky na sociální síti) a zobrazovat je bez nutnosti načtení celé nové stránky.

Tento postup je dnes hojně využívaný, zejména z důvodu rychlosti (není potřeba načítat pokaždé celou stránku), uživatelské přívětivosti (uživatel chce vidět okamžitou reakci na svůj podnět a nechce čekat na načtení nové stránky) a odlehčení nároků na samotný server (který nemusí počítat a vracet všechna data, ale pouze ta, která klient přímo požaduje).

2.5.1 Definice návštěvy webu

Pojem *návštěva webu* budeme chápat jako všechny požadavky od jednoho uživatele během jednoho sezení (anglicky *session*), tj. od začátku do konce jeho interakce s webovým serverem. Určit začátek sezení je snadné, jelikož stačí určit čas prvního požadavku. Určit konec sezení je těžší, jelikož jako pozorovatel, který vidí pouze požadavky a ne reálného uživatele, nevíme zda po posledním requestu bude následovat další nebo zda uživatel s prohlížením už skončil. Stanovíme si tedy časový interval, který když uplyne od posledního požadavku, prohlásíme, že uživatel s prohlížením skončil. Jak uvádí [1, s. 575] vhodným intervalem se jeví 30 minut.

Uživatele můžeme rozpoznat pouze podle jeho IP adresy a proto si pojem *návštěva webu* definujeme jako *všechny HTTP požadavky z jedné IP adresy v jednom časovém intervalu, kde se mezi po sobě následujícími požadavky nevyskytuje mezera delší než 30 minut.*

Název této práce je „Klasifikace uživatelů“, ovšem ve skutečnosti se budeme více zabývat právě klasifikací návštěv. Vzhledem k tomu, že nemůžeme nijak ověřit, z kolika různých IP adres jeden uživatel k webu přistupuje nebo naopak kolik různých uživatelů přistupuje z jedné adresy, nemáme možnost návštěvy přiřazovat k uživatelům a budeme jednu návštěvu webu považovat za vytvořenou jedním unikátním uživatelem. Navíc nám to poskytne tu výhodu, že různé návštěvy, byť vytvořené jedním uživatelem, můžeme klasifikovat do různých skupin a pokud se později rozhodneme klasifikovat uživatele definovaného IP adresou na základě jeho návštěv, můžeme dostat přesnější informace – uživatel se může vyskytovat ve více skupinách s větší či menší pravděpodobností.

2.5.2 Možné problémy této definice návštěvy webu

Problémy s dělením podle IP adresy

Jedním z možných míst, kde tato definice může dělat problémy je rozpoznávání uživatele pouze podle IP adresy. Zejména v těch případech, kdy je více uživatelů připojeno k Internetu z jedné adresy použitím *Network Address Translation (NAT)*, což je metoda, kterou router umožňuje spojení z intranetu do Internetu více počítačům za použití pouze jedné IP adresy na výstupu. Další možností, kterou lze dosáhnout stejného efektu je VPN nebo

proxy server.

V případě, že více uživatelů přistupuje ze stejné adresy, ale mezi jejich návštěvami budou potřebné 30-minutové mezery, budou klasifikováni v pořádku a každá navštěva zvlášť. Pokud budou navštěvovat web ve stejném okamžiku, bude více návštěv chybně spojeno do jedné. Řešením tohoto problému by mohlo být rozdělení návštěv nejen podle adresy, ale také podle hodnot hlaviček v požadavcích, zejména hodnot *cookies*, což jsou řetezce, které server pošle klientovi k uložení a kterými se poté klient prezentuje serveru. Vzhledem k tomu, že tyto informace se nelogují z bezpečnostních důvodů, v této práci tento postup nemohl být použit.

Opačný problém, tedy že jeden uživatel přistupuje na web z více adres, je mnohem méně závažný. Běžný uživatel většinou nemá více adres, ze kterých by mohl vybírat (uvažujeme protokol IPv4), a i kdyby měl, tak nemá důvod to dělat během jedné návštěvy webu. Pokud jeden člověk používá k návštěvě webu více zařízení ve stejný čas, je jeho chování na každém zařízení považováno za samostatnou návštěvu. I roboti často používají k procházení jednoho webu z důvodu omezení nadbytečného síťového provozu jeden počítač s jednou adresou [22].

Použití anonymizačních nástrojů, například systému Tor, také může vyvolat zdání, že požadavky přicházejí z různých adres a různých zdrojů, protože pro každé spojení se vytváří nová cesta, která může mít jiný výstupní bod (tj. poslední počítač v řadě, ze kterého je požadavek směrován na cílový server). Tyto nástroje používají jak roboti, tak lidští uživatelé a vzhledem k tomu, že byly navrženy právě za účelen anonymizace uživatelů a zabránění jejich klasifikace, je jejich odhalování jiná a mnohem těžší úloha než je téma této práce. Problém to není pro účely této práce tak závažný, jelikož tyto systémy nejsou tolik rozšířeny a používány ve velkém měřítku [15].

Problémy s volbou časového intervalu

Další problém může vzniknout z důvodu zvoleného časového intervalu. Uživatel nemusí během 30 minut vygenerovat žádný požadavek, když si například čte dlouhý článek. Ovšem další procházení webu by bylo lepší zařadit do stejné návštěvy. Opakem je uživatel, který se vrací k webu např. po dvaceti minutách, ale s jinou činností, kterou by bylo lepší z hlediska klasifikace zařadit do nové návštěvy.

Zvolený časový interval je také závislý na serveru, kterého se týká. Například uživatelé serveru vědeckého časopisu stráví více času čtením jednotlivých článků, což se projeví většími pauzami v návštěvách a interval by bylo možné zvětšit. Oproti tomu chatovací místnost nebo online hra na Internetu přepokládá skoro nepřetržitou aktivitu uživatele a návštěvu je možné ukončit už např. po pěti minutách nečinnosti.

V této práci bylo zvoleno 30 minut na základě [1].

3 Klasifikace

3.1 Definice

Jak uvádí [16], klasifikaci můžeme definovat jako systematické zařazování prvků do disjuktních množin (nazývaných *třídy*) na základě zvolených kritérií, většinou na základě společných vlastností. V této práci budeme klasifikovat návštěvy uživatelů do tříd podle jejich chování na webu. V ideálním případě budou třídy definované podle cíle, který návštěvník na webu má. Takže například u e-shopu budeme rozlišovat lidské návštěvníky na ty, co si pouze prohlížejí zboží a na ty co nakupují. V případě sociální sítě na uživatele, kteří hrají hry, uživatele, kteří vkádají nové statusy a například uživatele, kteří nejvíce času stráví chatováním. Definice tříd bude rozdílná pro každý případ nasazení a pro každý web.

Další podstatnou věcí, která ovlivní volbu tříd bude existence trénovacích dat nebo možností jejich získávání. Častou věcí, ze které se dají trénovací data vytvářet, jsou logy webového serveru (více v kapitole 10). V nich můžeme najít, které návštěvy se týkají jakých stránek a z toho je možné odvodit chování uživatele. Například pokud se během návštěvy bude často vyskytovat POST request na zdroj `/chat/send`, můžeme vytušit, že uživatel posílá hodně zpráv v chatu.

Další možností, jak získat trénovací data, je po určitý čas logovat detailněji než obvykle a právě z těch detailních dat trénovat aplikaci. Můžeme například u každého uživatele zaznamenávat jeho unikántní ID pokud se přihlásí, nebo hodnoty jeho hlavičky *Cookie*. Nebo si k požadavkům doplnit další parametry, např. pokud nás zajímá jestli zákazníci platí kartou nebo převodem přes účet, ale všechny tyto parametry jsou v těle POST požadavku a tedy se v záznamech nevyskytují, můžeme je pro určitý časový okamžik přidat do cesty v requestu jako parametr např: `/obchod/platba?typ=kartou`. Možností je také logovat úplně celé požadavky i s odpovědmi, ale tento postup má jednu nevýhodu – velmi často se tyto informace nelogují z bezpečnostních důvodů, popřípadě je uchovávání osobních dat uživatelů (například rodná čísla, čísla kreditních karet) přímo nelegální.

Možností jak definovat třídy je i využít znalosti a zkušenosti majitelů a správců daného webu. Například pokud administrátor ví z jakých adres

typicky přichází pokusy o útok na stránky, může data rozdělit podle těchto zdrojových IP adres. Popřípadě pokud data ukazují, že většina plateb je provedena v pondělí mezi sedmou a osmou ráno, můžeme data rozdělit podle času a právě tyto návštěvy v jeden daný čas považovat za reprezentaci jedné třídy.

Všechny dosud zmíněné metody spoléhají na to, že data jsou nějakým způsobem předpřipravená a že je připravil někdo se znalostí prostředí a řešeného problému. Jedné se o strojové učení *s učitelem*, tedy že data jsou připravena a označnována. Druhou možností je učení *bez učitele*, kde si algoritmus hledá závislosti v datech sám. Ovšem z hlediska této práce je tento postup nevhodný, jelikož většinou řešíme problémy, kdy dopředu víme, jaké třídy nás zajímají, jak jsou definované a popřípadě jak změnit chování serveru pokud je návštěva klasifikována do dané třídy (například neposílat žádná data robotům). V případě učení bez učitele by následně vzniklé třídy musely projít důkladnou analýzou k určení co reprezentují a jak se k daným návštěvám chovat. A mohlo by se stát, že vzniklé třídy by sice měly významné parametry pro klasifikační algoritmus, ovšem tyto parametry by nedávaly smysl při reálné aplikaci. Pokud tedy v této práci budeme zmiňovat strojové učení, budeme vždy myslet učení *s učitelem*.

3.2 Offline klasifikace

Klasifikaci, která probíhá na už uzavřených návštěvách, a tedy její výsledky návštěvníka během interakce s webem nijak neovlivní, budeme označovat jako „offline“ klasifkaci.

Důvod takové klasifikace může být prostá zvědavost, jaké typy návštěvníků na webu převažují, popřípadě kolik procent celkového provozu tvoří roboti a podobné otázky. Odpovědi na tyto otázky doplněny statistikou mohou dále ovlivňovat rozhodnutí o budoucnosti a provozu webu. Pokud například zjistím, že většina zákazníků platí kartou, mohu tuto volbu zvolit jako výchozí. Pokud zjistím, že roboti tvoří odhadem procento provozu webu, můžu se rozhodnout, že vlastně ani není potřeba je blokovat. Oproti tomu, pokud budou zabírat podstatnou část výpočetního výkonu serveru, může to být impuls pro zavedení systému jejich blokování.

Jiným důvodem může být hledání dalších vztahů a pravidel pro klasifikované návštěvníky webu, tedy detailní analýzy nalezených tříd. Zde je možné

například zjistit, na jakou stránku se většina uživatelů podívá po přihlášení a popřípadě jí nastavit jako výchozí. z analýzy chování uživatelů můžeme také zjistit, že většina uživatelů svou návštěvu ukončí po zobrazení agresivní reklamy a to může vést k rozhodnutí tuto reklamu omezit.

Oba dva výše uvedené důvody klasifikace jsou závislé na řešeném problému a detailní znalosti webu, na kterém bude systém nasazen. Tato práce se ovšem zabývá obecným problémem klasifikace a ne jednotlivými problémy, které by mohla řešit. Proto v této práci budeme offline klasifikaci používat k vytváření trénovacích a testovacích dat a hlavně k ověřování funkcionality jednotlivých metod klasifikace.

3.3 Online klasifikace

Online klasifikace oproti tomu znamená, že klasifikace probíhá během procházení webových stránek a chování webu se mění na základě toho, do jaké třídy byl uživatel zařazen. Nejčastěji se tento způsob používá k zobrazování cílené reklamy a k blokování robotů.

Klasifikátor může být přímo součástí navštěvovaného webu nebo před ním jako bezpečnostní opatření. K tomuto účelu, kdy je web oddělen od samotné klasifikace a dalších bezpečnostních filtrů se nejvíce hodí právě reverzní proxy server. Tato práce se zabývá obecným problémem klasifikace a použití proxy serveru je zde vhodnější než klasifikátor zabudovaný do webové služby, ovšem přímo na úlohu klasifikace to nemá vliv.

4 Používané metody k detekci robotů

Tato práce se z velké části zabývá separací robotů od lidských uživatelů, v této kapitole proto uvedeme přehled možných a používaných technik k jejich detekci. Jak uvádí [18], můžeme je rozdělit do 4 kategorií z nichž každá bude popsána dále v samostatné sekci.

4.1 Syntaktická analýza logů

Jedná se o jednoduchou metodu hledání klíčových slov v souboru s logy. Mezi tato klíčová slova patří v největší míře hodnoty hlavičky `User-Agent` nebo přímo IP adresy známých robotů. Největší nevýhoda tohoto postupu je, že dokáže odhalit pouze již známé roboty nebo „slušné“ roboty, kteří se „podepíší“ do hlavičky `User-Agent` [18].

4.2 Analýza vzorů procházení webu

Tato analýza se týká klasifikace návštěv místo samostatných požadavků na server. U každé zkoumané návštěvy se hodnotí například, jestli byl stažen soubor `robots.txt`, což je soubor, který udává, jak by se na stránce roboti měli chovat, tj. které zdroje mohou a které nesmí stahovat [19]. Spousta robotů tento soubor ignoruje a neřídí se jím, proto se ve většině případů používá pouze jako doplňující ukazatel [18].

Mezi další vzory a metriky, které se zde vyhodnocují může patřit poměr typů požadavků, jelikož roboti často pouze kontrolují stav webu a používají k tomu požadavek HEAD místo GET. Také poměr stažených stránek k počtu stažených obrázků popřípadě skriptů může napovídat, zda návštěvu vykonal robot nebo člověk. Tato metoda je podrobně rozebrána v kapitole 5 [18].

4.3 Metody strojového učení

Metody využívající strojové učení využívají stejné metriky jako metody z přechozí sekce, ovšem zatímco předchozí sekce měla parametry předem nastavené a dané znalostí administrátora, zde je použito strojové učení k získání optimálních parametrů. Metody strojového učení mají větší úspěšnost než analýza vzorů, ovšem jejich nevýhoda je v potřebě označkovaných trénovacích dat [18].

Je dobré poznamenat, že dvě nejúspěšnější metody rozebrané v této práci, tedy vylepšený Markovův model (kapitola 9) a klasifikace podle času na základě statistických metod (kapitola 7) používají právě metody strojového učení.

4.4 Turingův test

Klasifikace pomocí Turingova testu na webu většinou předloží uživateli úkol nebo dotaz, který ověřuje analytické myšlení člověka a rozlišuje uživatele na základě toho, zda test splní. Může se jednat o jednoduchý dotaz „Který den následuje po úterý?“, popřípadě rozpoznání kódu z obrázku. Tento systém je obvykle nazývaný *CAPTCHA* (Completely Automated Public Turing test to tell Computers and Humans Apart) a díky tomu, že vyžaduje přímou interakci s uživatelem, je použitelný pouze v online podobě [18, 20, 21].

Další metodou, jak může CAPTCHA rozpoznávat lidské uživatele od robotů, je použití JavaScriptu k zaznamenávání chování jedince na stránce (pohyby myší, stisky kláves) popřípadě ke zjištění, zda jsou skripty vůbec vykonávány. Tyto metody spadají na pomezí Turingova testu a strojového učení, kde se vzájemně doplňují. Například pokud metoda používající strojové učení dojde k závěru, že uživatel by mohl být robot, je mu zobrazen obrázek, ze kterého musí opsat kód a tím potvrdit, že jde o lidského uživatele [20, 21].

5 Klasifikace podle typu stahovaných zdrojů

5.1 Rozpoznávání robotů

Tento způsob klasifikace vychází z hypotézy, že robot nestahuje všechny zdroje příslušné k webové stránce, ale pouze ty, které obsahují hodně dat pro robota zajímavých. Některé zdroje na webových stránkách totiž nepřidávají žádnou informační hodnotu, ale pouze formátují text a upravují vzhled stránek. Obrázky na stránce sice mohou a většinou obsahují velké množství informací, ovšem rozpoznávat význam obrázků je značně těžší úloha než parsovat text. Proto si většina robotů vystačí se stahováním HTML stránek a ostatní zdroje (multimédia, formátovací styly, skripty, atd.) už by pro ně představovaly pouze zátěž navíc bez jakéhokoliv zisku.

Oproti tomu existují výjimky z pravidel, například roboti, naprogramovaní speciálně za účelem získávání fotografií např. ze sociálních sítí. Popřípadě stahování formátů souborů, které se dají snadno parsovat, například `pdf` a `txt`. Tyto formáty často stahují roboti více než lidé.

5.2 Klasifikace uživatelů

Tato metoda není omezena pouze na rozpoznávání robotů a se dá použít i ke klasifikaci lidských uživatelů do tříd. Vychází z toho, že i každá část webu má jiný poměr webových stránek a jiných typů souborů. Například uživatel sociálních sítí, který si prohlíží fotografie přátel bude stahovat více obrázků než uživatel, který svůj čas stráví chatováním.

Dalším kritériem, které v podstatné míře ovlivní typ stahovaných souborů, je cache, ať už přímo v prohlížeči nebo v proxy serveru. Ve většině případů se obrázky, styly a skripty nemění tolik jako samotný obsah stránek, a proto uživatel přistupující na stránku opakovaně využívá služby cache a negeneruje tolik požadavků na zdroje jako uživatel, který stránku navštěvuje poprvé. Na kritérium klasifikace uživatelů na základě používání cache není v této práci brán takový důraz jako na ostatní kritéria z toho důvodu,

že administrátor serveru nemůže poměr uživatelů využívající této funkce nijak ovlivnit a na rozhodování o budoucnosti serveru a případných opatřeních mají větší vliv ostatní kritéria klasifikace uživatelů.

Použití této metody klasifikace vyžaduje důkladnou znalost celého serveru v případě, že administrátor nastavuje klasifikátor ručně, popřípadě je možné využít automatické nastavení parametrů (více o této funkci aplikace v kapitole 12), kde je ale nutné opatřit si trénovací data.

5.3 Rozpoznání typu zdroje

Nejjistější metodou, jak zjistit typ zdroje vráceného na request od klienta by bylo logování na straně serveru, který má největší přehled o tom, jakého typu je odpověď na požadavek. Tento způsob ovšem by ovšem vyžadoval modifikaci serveru právě k logování vrácených typů souborů. Navíc je nevhodný i z toho důvodu, že nás více zajímá o jaké typy souborů si klient a potenciální robot říká, než jaké typy jsou mu skutečně vraceny.

Analýza typů zdrojů, které server vrací je také otázkou znalosti serveru a jeho fungování. V zásadě nic neomezuje server v odesílání dat v jakémkoliv formátu, ovšem pro správné fungování webu je nutné, aby klient obdrženým datům porozuměl, tj. aby je dostal v takovém formátu a takového typu, jaký očekává.

Pro účely této práce budeme předpokládat, že zdroj je soubor uložený na serveru a jeho typ je daný jeho příponou. Uvedeme příklad HTTP požadavku:

```
GET /index.html HTTP/1.1
```

Tento požadavek si žádá o zdroj identifikovaný cestou `/index.html` a budeme předpokládat, že tímto zdrojem bude soubor `index.html` uložený v kořenovém adresáři webového serveru. Přípona souboru udávající typ je `html`, což značí webovou stránku ve formátu HTML. V tomto případě tedy za požadovaný typ zdroje považujeme webovou stránku.

Ne vždy je ovšem typ zdroje daný příponou. Spousta dynamicky generovaných webových stránek příponu neuvádí vůbec a (správně) předpokládá, že prohlížeč stránku vykreslí jako HTML a její formát pozná podle samotného obsahu. Hlavička požadavku pak může vypadat takto:

```
GET /zpravy/domaci HTTP/1.1
```

Dalším případem, kdy přípona není uvedena, může být, že se jedná o dynamicky generovanou odpověď na požadavek vygenerovaný skriptem. Například HTML stránka obsahuje skript, který periodicky kontroluje nové příspěvky v diskusním fóru požadavkem na zdroj `/prispevky/nove` a očekává odpověď ve formátu JSON (*JavaScript Object Notation* – formát pro snadný zápis a přenost dat v JavaScriptu, příklad viz dále). Odpovědí může být:

```
{"prispevky": [
        {"uzivatel": "Jan", "text": "Ano, souhlas"},
        {"uzivatel": "Martin", "text": "To nikdy"},
        {"uzivatel": "Iveta", "text": ":D"},
  ]
}
```

Je vidět na první pohled, že tento text není HTML stránkou. Prohlížeč ho také nebude parsovat jako HTML, ale vrátí ho přímo JavaScriptu, který JSON očekává a umí ho číst. Jak je vidět, obě cesty v požadavcích nemají určenu příponu a vrací jiný typ dat. Rozdíl je v tom, jak požadavek vzniká a jak je na klientské straně zpracováván. Tento příklad vypovídá o tom, že jsou případy, kdy k přesnému určení typu požadavků nestačí logy, ale je vyžadována znalost serveru a jeho fungování.

Naštěstí většina zdrojů, které nás zajímají, tedy multimédia, skripty a formátovací styly není dynamicky generována a dá se snadno určit pomocí přípony v požadavku. Typ požadovaného zdroje tedy v práci definujeme jako *řetězec za poslední tečkou v cestě požadavku po odstranění všech parametrů v cestě obsažených*. Na příkladu požadavku s cestou `/server/obrazky/obr1.jpg?klic=hodnota` tedy nejdříve odstraníme všechny parametry z cesty na `/server/obrazky/obr1.jpg` a řetězec za poslední tečkou bude `jpg`. Pokud se v cestě žádná tečka nevyskytuje, považujeme za typ souboru prázdný řetězec.

5.4 Klasifikace

Před samotnou klasifikací musí administrátor určit, které typy souborů jsou znakem které třídy a v jaké míře. Pro každou třídu a každý typ určí, jakou

Tabulka 5.1: Váhy pro klasifikaci podle typu

Typ souboru	Váha pro lidského uživatele	Váha pro robota
`jpg`	5	1
`css`	5	1
`js`	5	1
`html`	2	3
`pdf`	1	4
`txt`	1	5

vahou se daný typ souboru podílí na celkovém zařazení. Pro základní odlišení lidských uživatelů od robotů může nastavení vypadat tak, jak je znázorněno v tabulce 5.1. Váhy v tabulce byly určeny náhodně, slouží pouze pro ilustraci v následujícím příkladě.

Samotná klasifikace návštěvy probíhá tak, že se nejdříve nastaví pro každou třídu její *celková váha*. Hodnoty se nemusí nastavovat na nulu, můžeme ze začátku předpokládat, že uživatel je člověk nastavením celkové váhy pro tuto třídu na vyšší číslo. V tomto případě můžeme nadefinovat že počáteční váha pro člověka bude 5 a pro roboty 0. Poté se návštěva rozdělí na jednotlivé požadavky, které se zpracují samostatně. Pro každý požadavek se určí jeho typ a pokud jsou definovány váhy pro daný typ, k celkové váze se připočítají.

Na konci návštěvy, po zpracování všech požadavků, určíme třídu jako tu, která má nejvyšší hodnotu celkové váhy. Pokud tedy budeme uvažovat stále stejné hodnoty vah, jak uvádí tabulka 5.1, a budeme zpracovávat návštěvu skládající se z požadavků:

```
GET /index.html HTTP/1.1
GET /styl.css HTTP/1.1
GET /script.js HTTP/1.1
GET /obrazek.png HTTP/1.1
```

Celková váha pro člověka se skládá z:

- 5 – počáteční váha,
- 2 – váha za první požadavek na `html` zdroj,

- 5 – váha za druhý požadavek na `css` zdroj,
- 5 – váha za třetí požadavek na `js` zdroj.

V součtu pro celkovou váhu vychází 17. Je vidět, že nijak nedefinovaný typ `png` se na celkové váze neprojevuje vůbec. Váhy pro robota budou:

- 0 – počáteční váha,
- 3 – váha za první požadavek na `html` zdroj,
- 1 – váha za druhý požadavek na `css` zdroj,
- 1 – váha za třetí požadavek na `js` zdroj,

tedy v součtu 4. Tedy tato návštěva bude klasifikována jako lidský uživatel. Jiná návštěva skládající se z

```
GET /robots.txt HTTP/1.1
GET /forum/prispevky.html?stranka=1 HTTP/1.1
GET /forum/prispevky.html?stranka=2 HTTP/1.1
GET /forum/prispevky.html?stranka=3 HTTP/1.1
```

ve výsledku bude mít celkovou váhu pro člověka 12 (5 + 1 + 2 + 2 + 2) a pro robota 14 (5 + 3 + 3 + 3), bude tedy klasifikována jako robot.

Klasifikace touto metodou může probíhat i online, kdy se váhy upravují za běhu podle nových příchozích požadavků a návštěva je zařazena do třídy podle toho, která celková váha je nejvyšší v okamžiku klasifikace. Během návštěvy se tedy zařazení do třídy může měnit.

6 Klasifikace podle časů požadavků

Tato metoda klasifikace vychází z předpokladu, že lidský návštěvník prochází webem celkem nahodile, prochází různé stránky, na kterých stráví různě dlohou dobu. Oproti tomu robot buď stahuje stránky tak rychle, jak mu rychlost internetového připojení nebo rychlost serveru dovolí nebo mezi jednotlivými požadavky určitou dobu čeká. Stahování maximální možnou rychlostí se příliš nepoužívá z toho důvodu, že je velice snadno odhalitelné a zablokovatelné konvenčními bezpečnostními metodami [22].

Většina robotů tedy mezi jednotlivými požadavky na server nějakou dobu čeká. Tato doba ovšem není narozdíl od lidských uživatelů závislá na množství a důležitosti informací na stránce, ale často bývá zvolena konstantní nebo je náhodně generovaná v určitém intervalu [22].

Tato jednoduchá metoda je založena na sledování dvou metrik – průměrnému časovému intervalu mezi jednotlivými požadavky a směrodatnou odchylkou na této množině intervalů. Základní hypotéza je taková, že člověk, který prochází web, stráví nad různými stránkami různou dobu, takže rozptyl intervalů mezi jednotlivými návštěvami bude větší než v případě robota, který může být naprogramován, aby mezi požadavky čekal přesně odměřenou dobu. Zároveň roboti většinou procházejí web rychleji než lidé a proto by i průměrný čas mezi požadavky měl být menší.

Parametry klasifikátoru jsou tedy pouze dvě výše zmíněné veličiny pro každou třídu, navíc doplněné jejich vahou. Průměr a směrodatnou odchylku může určit administrátor serveru ručně nebo se může vypočítat jako průměrná hodnota pro každou třídu z trénovacích dat. Právě metoda vypočtení těchto hodnot z dat byla v práci použita.

A samotná klasifikace probíhá tak, že u každé návštěvy se spočítají intervaly mezi jednotlivými požadavky, z této množiny intervalů se spočítá průměr a směrodatná odchylka a porovná se která třída je svým odhadem nejblíž těmto hodnotám po započtení vah. Váhy jsou pro každou třídu jiné, tím se mimo jiné dá ovlivnit poměr výsledků klasifikovaných tříd. Uvedeme na příkladu:

Z dat vyplynulo, že průměrný čas, který na jedné stránce člověk stráví (tj. interval mezi jednotlivými požadavky, budeme značit $\bar{i}_c$), je třicet sekund se směrodatnou odchylkou (σ_c) dvacet sekund. Zatímco u robota je průměrný

čas mezi požadavky ($\bar{i}_r$) pět sekund s odchylkou (σ_r) jedné sekundy.

Zároveň bylo určeno, že dvakrát větší váhu při rozhodování pro lidské uživatele bude mít směrodatná odchylka, tedy její váha ($w_{\sigma,c}$) bude dvě a váha průměru ($w_{i,c}$) jedna. Pro robotické návštěvníky budou váhy jiné, váha pro průměr ($w_{i,r}$) bude tři a pro směrodatnou odchylku ($w_{\sigma,r}$) jedna.

$$\begin{aligned} \bar{i}_c = 30; \qquad \sigma_c = 20; \qquad \bar{i}_r = 5; \qquad \sigma_r = 1; \\ w_{i,c} = 1; \qquad w_{\sigma,c} = 2; \\ w_{i,r} = 3; \qquad w_{\sigma,r} = 1; \end{aligned}$$

Budeme klasifikovat návštěvu skládající se z požadavků, které přišly v časech

`11:50:23, 11:50:24, 11:51:10, 11:52:01 a 11:52:15`

tedy množina intervalů pro nás je $i = \{1, 46, 51, 14\}$, její průměr $\bar{i} = 28$ a směrodatná odchylka $\sigma = \sqrt{(27^2 + 18^2 + 23^2 + 14^2)} \doteq 42,17$.

Odchylku od očekávaných hodnot pro lidského návštěvníka (označíme jako o_c) vypočteme jako rozdíl skutečných hodnot od očekávaných v absolutní hodnotě přenásobených váhou

$$\begin{aligned} o_c &= |\bar{i}_c - \bar{i}| \cdot w_{i,c} + |\sigma_c - \sigma| \cdot w_{\sigma,c} \\ o_c &= |30 - 28| \cdot 1 + |20 - 42,17| \cdot 2 \\ o_c &= 2 + 44,34 \\ o_c &= 46,34 \end{aligned} \tag{6.1}$$

Odchylku od hodnot pro robota (o_r) vypočteme stejným způsobem

$$\begin{aligned} o_r &= |\bar{i}_r - \bar{i}| \cdot w_{i,r} + |\sigma_r - \sigma| \cdot w_{\sigma,r} \\ o_r &= |5 - 28| \cdot 3 + |1 - 42,17| \cdot 1 \\ o_r &= 69 + 22,17 \\ o_r &= 91,17 \end{aligned} \tag{6.2}$$

Jak ukazují výsledky rovnic 6.1 a 6.2, odchylka od očekávaných hodnot pro lidského návštěvníka je menší a tato návštěva by tedy byla klasifikována jako vytvořená člověkem.

7 Klasifikace podle času s využitím statistických metod

Tato metoda je podobná té přechozí, ovšem ke klasifikaci používá celou množinu intervalů mezi požadavky a její rozložení. Vychází ze statistické metody zvané *test dobré shody* nebo *Pearsonův* χ^2 *test dobré shody* [23].

7.1 Hladina významnosti

Na základě omezené množiny pokusů nikdy nemůžeme žádnou hypotézu s jistotou potvrdit nebo vyvrátit a proto je u tohoto testu velmi důležitá hodnota *hladiny významnosti* testu. Hladina významnosti (také označována jako α *hodnota*) udává pravděpodobnost, že hypotézu nesprávně zamítneme, i když platí. α se určuje před provedením pokusu a nejčastěji používané hodnoty jsou 5% a 1%.

Pro testovanou hypotézu (a za předpokladu, že platí) tedy spočítáme pravděpodobnost, že statistická data budou stejná nebo ještě extrémnější než data naměřená na testovaném vzorku. Extrémnější v tomto případě znamená, že data budou odporovat testované hypotéze. Tato pravděpodobnost se nazývá *p-hodnota*, anglicky *p-value*. Pokud tedy p-hodnota bude menší než zvolená hladina významnosti testu, hypotézu se nepodařilo vyvrátit a tak dále předpokládáme, že platí. Pokud je p-hodnota větší než hladina významnosti testu, hypotézu se nám podařilo vyvrátit a většinou s tím potvrdit jinou, *alternativní* hypotézu, k ní opačnou [24].

Uvedeme na příkladě volně přeloženém z [25]:

Testujeme osoby, zda mají schopnost jasnovidectví. Testované osobě je ukázáno 25 karet lícem dolů a jejím cílem je určit, k jaké ze 4 barev jednotlivé karty náleží. Kolikrát se testovaná osoba trefí označíme jako X. Jako nulovou hypotézu (H_0) označíme stav, kdy osoba dar jasnovidectví nemá. Alternativní hypotéza (H_1) je k ní opačná, tedy, že osoba je jasnovidec.

Pokud H_0 platí, bude testovaná osoba pouze tipovat. Pro kteroukoliv kartu bude pravděpodobnost, že se osoba trefí $1/4$. Pokud bude alternativní hypotéza pravdivá, počet správně předpověděných barev karet bude větší.

Pravděpodobnost správné předpovědi označíme jako p a hypotézy můžeme definovat jako:

$$H_0 : p = \frac{1}{4}; \qquad H_1 : p > \frac{1}{4}$$

Pokud testovaná osoba správně uhodne všech 25 barev, budeme ho považovat za jasnovidce, tedy odmítneme H_0 a přijeme H_1. Pokud uhodne pouze 5 nebo 6, budeme to považovat za náhodu a H_0 neodmítáme. Otázkou zůstává, jak se zachovat v případě, že uhodne například 12 nebo 17 barev. Které číslo c, označující počet správně zařazených karet, budeme považovat za znak jasnovidectví? Je vidět, že čím větší číslo zvolíme, tím kritičtější budeme k případným nálezům. V případě, že zvolíme menší číslo, bude určitý počet lidí označen za jasnovidce, i když jejich správně uhodnuté karty byly pouze dílem náhody. Volíme tedy, jak často jsme ochotni akceptovat falešně pozitivní nález. Pro $c = 25$, pravděpodobnost takového nálezu bude:

$$P(X = 25 \mid p = \frac{1}{4}) = (\frac{1}{4})^{25} \approx 10^{-15}.$$

Tedy pravděpodonost falešně potitivního nálezu bude rovna pravděpodobnosti, že náhodně uhodneme všech 25 barev a bude velmi malá. V případě, že nebudeme požadovat všechny karty správně určeny, ale nastavíme $c = 10$, tato pravděpodobnost bude:

$$P(X \geq 10 \mid p = \frac{1}{4}) = \sum_{k=10}^{25} P(X = k \mid p = \frac{1}{4}) \approx 0,07.$$

Vidíme, že pro $c = 10$ máme mnohem větší šanci na falešně pozitivní nález.

Předtím, než provedeme samotný pokus, musíme tedy určit maximální pravděpodobnost falešně pozitivního nálezu (α), kterou jsme ochotni akceptovat. Jak je uvedeno výše, typicky se používají hodnoty 1% nebo 5% (pokud zvolíme nulu, bude k potvrzení naší hypotézy, že člověk je jasnovidec, potřeba nekonečný počet správných pokusů). V tomto příkladě zvolíme hladinu významnosti 1% a dopočítáme c ze vztahu:

$$P(X \geq c \mid p = \frac{1}{4}) = P(X = c \mid p = \frac{1}{4}) \leq 0,01.$$

Pro všechna c, která vztahu vyhovují vybereme to nejmenší, aby došlo k minimalizaci falešně negativních nálezů (případů, kdy nesprávně prohlásíme, že člověk s darem jasnovidectví, jasnovidcem není). Pro výše uvedený příklad zvolíme $c = 13$.

7.2 Test dobré shody

Pearsonův χ^2 test dobré shody testuje hypotézu, že rozdělení jevů pozorovaných na vzorku dat odpovídá definovanému teoretickému rozdělení. V tomto testu musí být pozorované jevy disjunktní, na sobě nezávislé a jejich celková pravděpodobnost se musí rovnat jedné. Test je založen na tom, že náhodnou veličinu s multinomickým rozdělením lze transformovat na veličinu s rozdělením *chí–kvadrát*. ke konstrukci testu v tomto případě budeme používat součet druhých mocnin odychlek (označeno χ^2), který porovnáme s tabulkovou hodnotou rozdělení pravděpodobnosti χ^2 s daným počtem stupňů volnosti a požadovanou hladinou významnosti [23].

Uvedeme na příkladu hrací kostky:

Předpokládáme, že šestistěnná hrací kostka je férová, a tedy že pravděpodobnost hodu kteréhokoliv čísla je $1/6$. To bude naše nulová hypotéza (H_0). Alternativní hypotéza (H_1) bude znít, že kostka férová není a že rozdělelní pravděpodobnosti není rovnoměrné. Hladinu významnosti pro tento test zvolíme 5%. Nyní provedeme samotný pokus – provedeme sto dvacet pokusů (hodů kostkou) a zaznamenáme četnosti. Naměřené četnosti (X) a očekávané četnosti (N) jsou shrnuty v tabulce 7.1. Nyní spočítáme odychlky očekávaných a naměřených hodnot podle vzorce:

$$O_i = \frac{(N_i - X_i)^2}{N_i}$$

Tabulka 7.1: Měřené a předpokládané hodnoty pro hody kostkou

Hozené číslo	Předpokládaná četnost (N)	Naměřená četnost (X)
1	20	15
2	20	21
3	20	25
4	20	14
5	20	19
6	20	26
$\sum$	120	120

Součet těchto odchylek dává požadovnou hodnotu *chí–kvadrát*:

$$\chi^2 = \sum_{i=1}^{k} O_i$$

Tabulka 7.2: Odchylky od předpokládaných hodnot pro hody kostkou

Hozené číslo	Předp. četnost (N)	Naměřená četnost (X)	Odchylka (O)
1	20	15	1,25
2	20	21	0,05
3	20	25	1,25
4	20	14	1,80
5	20	19	0,05
6	20	26	1,80
$\sum$	120	120	**6,20**

Hodnoty odchylek i jejich součet uvádí tabulka 7.2. Hodnotu $\chi^2 = 6,2$ porovnáme s tabulkovou hodnotou *chí–kvadrát* rozdělení, kde počet stupňů volnosti je $k-1 = 5$. Jako hladinu významnosti jsme zvolili 5% a proto nás zajímá 0,95 kvantil. Tato tabulková hodnota je 11,06. Protože námi vypočtená hodnota 6,2 je menší než tabulková, nulovou hypotézu nezamítáme a kostku můžeme považovat za férovou.

7.3 Klasifikace návštěv

Pro klasifikaci návštěv webového serveru použijeme v zásadě stejnou metodu, ovšem s několika úpravami. Časy mezi jednotlivými požadavky rozdělíme do intervalů po několika sekundách a pro každý interval spočítáme počet časů v tomto intervalu umístěných. To samé provedeme s funkcí, se kterou budeme návštěvu srovnávat (viz dále). Obě tyto funkce normalizujeme, jelikož návštěva bude nejspíše mít jiný počet prvků (intervalů mezi požadavky) než předpokládaná funkce.

Dalším rozdílem zde bude, že hladinu významnosti nahradíme *prahem*, tedy hodnotou, po jejímž překročení budeme návštěvu klasifikovat jinak, než pokud překročena nebyla. Zde záleží na tom, jak si danou funkci a způsob klasifikace nastavíme. V případě testu dobré shody se hodnota porovnává s tabulkovou hodnotou χ^2 rozdělení pro daný počet stupňů volnosti. Jelikož zde hodnotu prahu určuje administrátor serveru, není potřeba určovat počet stupňů volnosti a zajímat se o χ^2 rozdělení.

7.4 Metody srovnávání

V této sekci se zaměříme na funkce, se kterými budeme jednotlivé návštěvy srovnávat. Většina dále rozebíraných metod klasifikuje pouze do dvou tříd – na návštěvy, které vyhovují definované funkci a všechny ostatní. U těchto metod je velmi důležitá hodnota prahu, která přímo určuje poměr rozdělení. Tuto hodnotu určuje administrátor serveru popřípadě se dá natrénovat na trénovacích datech. Práh určuje jak moc se návštěva může odlišovat od definované funkce, ale kdy bude ještě zařazena do stejné třídy.

7.4.1 Předem definovaná funkce

Myšlenka tohoto postupu spočívá v tom, že robot bude naprogramován aby mezi požadavky vkládal časové prodlevy. Rozdělení prodlev ovšem bude definováno pseudonáhodnou funkcí v programu robota a tato funkce bude nejspíše konvergovat k definovanému rozdělení. V nejčastějších případech k rovnoměrnému nebo normálnímu rozdělení, jelikož to jsou nejčastější metody generování pseudonáhodných čísel.

Budeme proto intervaly v návštěvě srovnávat s distribucemi těchto funkcí a pokud se budou podobat (vzhledem k definovanému prahu), prohlásíme návštěvu za robota.

Kromě prahu zde ještě musíme definovat parametry předpokládané funkce. V případě rovnoměrného rozdělení to bude délka intervalu, nebo jinak maximální možná doba, na kterou je robot nastaven čekat. U gaussovské funkce střední hodnotu a odchylku.

7.4.2 Největší shoda

Tato metoda se od ostatních liší tím, že může klasifikovat návštěvy do více než dvou tříd. Zároveň také nepotřebuje hodnotu prahu, jelikož pro každou návštěvu se určí, které předem definované funkci (a třídě) se podobá nejvíce. Opět je potřeba definovat funkce pro porovnání, poté už se jenom pro každou třídu vypočítá odchylka a vybere se třída s nejmenší hodnotou.

7.4.3 Průměr pro lidské návštěvníky proti zbytku

Tato metoda vychází se stejného základu jako klasifikace na základě shody s předem definovanou funkcí. Rozdíl zde je ten, že srovnávané rozdělení není definováno administrátorem na začátku, ale vypočte se z testovacích dat jako součet dat pro lidské návštěvníky (po normalizaci tedy dostaneme průměr). Předpokladáme, že panuje větší shoda ve stylu procházení webu mezi lidmi než mezi roboty. Zároveň také nechceme řešit typ nebo druh robota, který prochází naše stránky, ale stačí nám vědět, že se nejedná o člověka.

Tato metoda je jediná používaná při testování klasifikátoru s využitím statistických metod v této práci.

7.5 Nulové hodnoty

V případě předem definované spojité funkce, jako je konstanta pro rovnoměrné rozdělení nebo gaussovská funkce, se nikdy nestane, že by předpokládaná hodnota funkce vyšla nula. V případě, že rozdělení počítáme z testovacích dat, se toto stát může. Pak vidíme, že ve vzorci pro odchylku:

$$O_i = \frac{(N_i - X_i)^2}{N_i}$$

by došlo k dělení nulou. Daná odychlka by tedy byla nedefinovaná. Samotný test dobré shody toto nepřipouští, minimální předokládanou hodnotu pro interval udává jako 5 [23]. Pokud se stane, že hodnota pro interval je menší než 5, je třeba sloučit interval s jeho sousedem a tento postup opakovat tak dlouho, až všechny intervaly budou mít předpokládanou hodnotu větší nebo rovnu 5.

Z programátorského hlediska má tento postup tu nevýhodu, že vzniklé intervaly nejsou stejně velké. Proto v tomto případě použijeme jinou metodu pro odstranění nul v předpokládaných datech. Zavedeme, že všechny intervaly mají počáteční četnost 1, ještě před samotným přičítáním intervalů z testovacích dat. Tímto způsobem se zbavíme nul v předpokládaných četnostech. Ovšem za cenu snížené přesnosti vzorce.

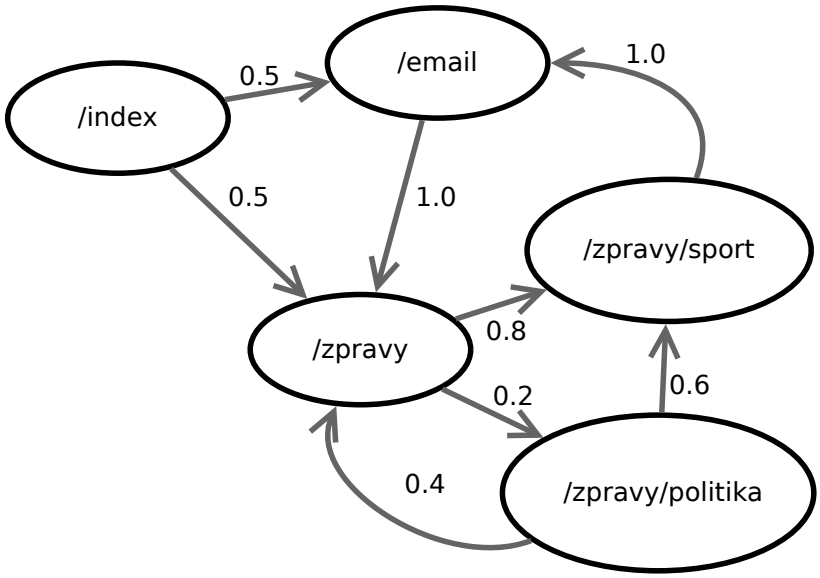
8 Markovův model

Tento postup klasifikace uživatelů je založen na pořadí, ve kterém si uživatelé prohlížejí webové stránky. Typické chování lidských uživatelů je, že člověk sleduje odkazy na stránce, kliká na ty, které ho zajímají a doplňují další informace, a tím se přesouvá ze stránky na jinou stránku. Zároveň také jeho prohlížeč stahuje zdroje v předpokládaném pořadí, tak, jak jsou na stránce uvedeny. Člověk také nezkouší měnit parametry v odkazech a pokud je server plně funkční, málokdy se dostane na neexistující stránku.

Oproti tomu robot nemusí stahovat zdroje v pořadí, v jakém jsou na stránce uvedeny, nemusí je ani stahovat vůbec. Také pořadí procházení stránek se u robotů mění, roboti mohou procházet web jako graf algoritmy procházení do šířky nebo do hloubky [22], což v případě procházení do hloubky je snadno predikovatelné, jelikož robot vždy jako další otevře odkaz, který je na stránce jako první. V případě procházení do šířky zase bude robot velmi často stahovat stránku, na kterou nevedl z aktuální stránky žádný odkaz, což lidský uživatel dělá jen málokdy.

8.1 Mapa webu jako graf

Pro lepší představu, jak se uživatelé pohybují na webovém serveru, si můžeme všechny stránky představit jako uzly v grafu. Tyto uzly jsou propojeny hranami, jejichž hodnoty v tomto případě budou určovat pravděpodobnost toho, že uživatel nacházející se v uzlu, ze kterého hrana vychází přejde do uzlu, do kterého hrana vede. V případě, že pravděpodobnost přechodu je nulová, hranu uvažujeme s hodnotou 0, ovšem do obrázků grafů takové hrany nebudeme zakreslovat.



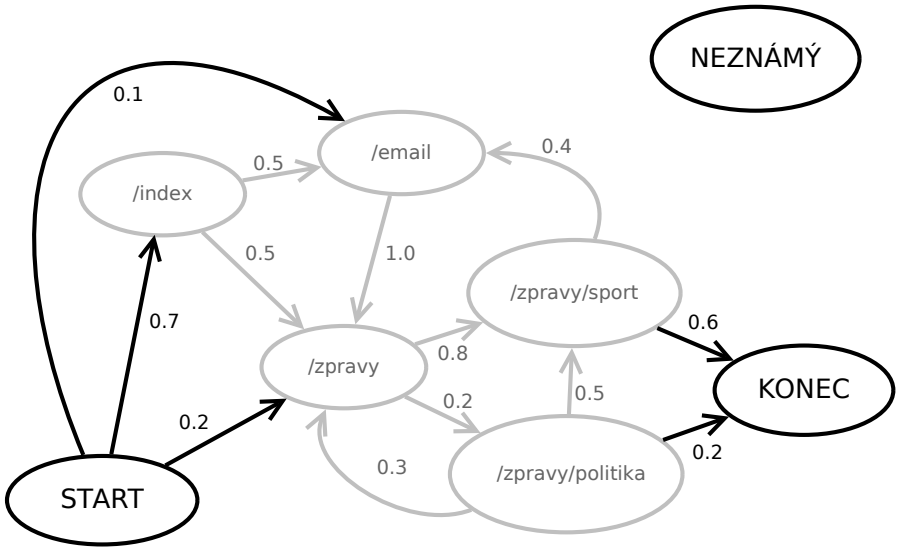
Obrázek 8.1: Stránky webu jako graf

Příklad vidíme na obrázku 8.1. Jedná se o velmi zjednodušenou verzi webového serveru s pouze pěti stránkami. Zatím nebudeme uvažovat, kde uživatel svoji návštěvu začne a kde skončí (viz dále). Na obrázku pouze vidíme, že pokud se uživatel nachází v uzlu `/index` (tj. jeho poslední požadavek vedl na tuto stránku), tak se s pravděpodobností 0,5 přesune do uzlu `/email` a opět s pravděpodobností 0,5 do uzlu `/zpravy`. Pokud poslední požadavek byl směřován na URL `/zpravy`, je pravděpodobnost přesunu na uzel `/zpravy/sport` 0,8, zatímco na uzel `/zpravy/politika` pouze 0,2.

8.1.1 Přidané uzly

Aby se model mohl označovat jako Markovův, je vyžadováno, aby součet všech hran vycházejících z jednoho uzlu byl 1. Jelikož ovšem nevíme kolik požadavků v návštěvě bude (v případě online klasifikace), a pro lepší porovnání výsledků klasifikace, zavedeme dva další speciální uzly – *START* a *KONEC*. Jak napovídá název, všechny návštěvy začínají v uzlu START a končí v uzlu KONEC.

V rámci offline klasifikace si můžeme všechny požadavky předzpracovat a uložit do grafu, takže se poté při samotné klasifikaci nemůže stát, že požadavek povede na uzel, který nemáme v grafu. V případě online klasifikace se to ovšem stát může. I když máme naprosto neměnný web a všechny přístupné stránky jsme uložili do grafu, nikdy si nemůžeme být jistí, že klient nepošle request na nikdy neviděnou stránku, byť taková stránka neexistuje a odpověď na tento požadavek skončí chybovým kódem 404. Přidáme tedy další uzel *NEZNÁMÝ*. Do tohoto uzlu budeme počítat všechny požadavky, které jsme nenašli ve zbytku grafu.



Obrázek 8.2: Přidané uzly

Přidané uzly jsou zobrazeny na obrázku 8.2. Je vidět, že uzel START obsahuje pouze výstupní hrany a uzel KONEC pouze hrany vstupní. Pravděpodobnosti všech hran, které z uzlu vystupují musí v součtu dát jedna, proto bylo nutné, zde nějaké pravděpodobnosti upravit. Uzel NEZNÁMÝ má v tomto případě všechny hrany s hodnotou nula, proto nejsou v obrázku zakresleny. Hodnoty hran tohoto uzlu jsou dále řešeny v sekci 8.4.

8.1.2 Parametry v URL a typy požadavků

Nyní je načase zvážit, jak parametry URL ovlivňují chování uživatele a rozhodnout, jestli více požadavků na stejnou stránku, ale s jinými parametry, patří do stejného uzlu nebo budeme URL brát jako celek a uzlů nám vznikne několik. Toto téma je rozebráno v kapitole 10.

Další věcí ke zvážení je rozdělování nebo naopak slučování požadavků na stejnou adresu, ale s jiným typem požadavku. Například zda požadavek

```
GET /chat/zprava HTTP/1.1
```

sloučit s požadavkem

```
POST /chat/zprava HTTP/1.1
```

Zde samozřejmě záleží na tom, jak chceme uživatele klasifikovat, na nastavení serveru a jeho funkcionalitě. Pro tento příklad budeme předpokládat, že typem požadavku `GET` čteme zprávy a typem `POST` je posíláme. Pokud chceme uživatele rozdělit podle toho, jestli na webu chatují nebo si například prohlížejí zprávy, a je nám jedno jestli zprávy v chatu spíš čtou nebo spíše posílají, můžeme tyto požadavky sloučit. V opačném případě, kdy klasifikujeme podle toho, zda uživatelé v chatu spíše posílají zprávy nebo je čtou, musíme tyto požadavky zpracovávat odděleně jako dva různé uzly v grafu.

8.1.3 Úpravy uzlů

Všechny další úpravy uzlů se týkají požadavků specifických pro každý server. Jednak může jít o úzce specializovanou předpřípravu dat na základě zdrojové adresy, cookies, typu dat, které uživatel v requestu žádá nebo dokonce údajů, které doplňuje až server. Například chceme z klasifikace úplně vyřadit

uživatele, kteří přicházejí z vnitřní sítě, popřípadě uzly, které patří do administrace stránek. Další možností je rozdělit některé uzly podle toho ze kterého zařízení se uživatelé připojují, jelikož se na serveru stejně rozdělují a vrácené stránky vypadají jinak.

Další úpravy uzlů se můžou týkat slučování uzlů, které odkazují na stejný typ dat. Uvedeme na příkladu sociální sítě, kde se každý nový příspěvěk uloží pod unikátním ID a je dostupný na stránce `/status/<id>`, například `/status/45635`. Další status bude dostupný na jiném ID, např. `/status/45636`. Ovšem z dlouhodobého hlediska by bylo lepší tyto uzly a všechny další, které odkazují na jiné příspěvky, sloučit do jednoho, nového uzlu, označeného pouze jako *STATUS*. Důvodem je, že pokud klasifikátor natrénujeme na starších datech, nebude pak schopný správně třídit nové statusy, jelikož ty se nikdy v trénovacích datech nevyskytovaly a budou tedy všechny zařazeny do uzlu *NEZNÁMÝ*. Samozřejmě opět záleží na typu klasifikace a typu serveru. V některých případech, např. kdy klasifikujeme na které příspěvky se kteří uživatelé dívají, si nemůžeme dovolit tuto informaci ztratit v předpřípravě.

Tyto úpravy grafu musí definovat administrátor serveru a v této práci jsou pouze zmíněny, ale dále nerozebírány ani netestovány, jelikož jsou úzce specialzované pro jednotlivé servery a dostupná testovací data tyto údaje neobsahují.

8.2 Návštěva jako sled v grafu

Každá návštěva se pak dá popsat jako sled v grafu, začínající v uzlu START a končící v uzlu KONEC. Zároveň má tento sled celkovou pravděpodobnost, kterou získáme vynásobením všech hodnot hran ve sledu. Pravděpodobnosti přechodu ze stavu do stavu jsou dané pouze stavem, ve kterém se uživatel nachází v současnosti. Pokud časy požadavků nahradíme jejich pořadím, tedy nebude záležet na tom, jak dlouho se návštěvník na stránce zdržel, dostaneme diskrétní hodnotu pro čas, začínající nulou a končící délkou návštěvy.

Stochastický proces, který je definován nad konečnou množinou stavů (v našem případě uzlů) a diskrétní množinou času a který splňuje podmínku, že pravděpodobnost přechodu ze stavu do stavu je dána pouze aktuálním stavem a nikoli historií, splňuje definici *Markovova procesu*. Jelikož v našem případě je stav plně pozorovatelný a jako server nemáme nad chováním uživatele žádnou kontrolu, jedná se o *Markovův řetězec*.

Pro jeden graf, sestrojený pro typického uživatele (viz sekce 8.3), a pro jednu návštěvu dostaneme jednu pravděpodobnost, udávající, jak je pravděpodobné, že návštěvník bude web procházet právě daným sledem požadavků. Tyto údaje nám neumožňují klasifikaci, ale jsou využitelné v dalších úlohách, jako je předpřipravování odpovědí a úpravy webu (více v sekci 8.5).

Pro klasifikaci návštěvníků tedy potřebujeme více hodnot pravděpodobností, pro každou třídu jednu. Ty získáme tak, že si vytvoříme pro každou třídu jeden graf nad stejnými uzly, ovšem s jinými pravděpodobnostmi, definovanými pro každou třídu. Tyto pravděpodobnosti můžeme nastavit ručně, ovšem vzhledem k velikosti webů je lepší je trénovat na základě dostupných dat, jak je ukázáno v následující sekci.

8.3 Trénování klasifikátoru

Ke trénování klasifikátoru potřebujeme trénovací data, tedy sady návštěv, předem označkovaných, tj. rozdělelených do tříd. Nejdříve projdeme všechny koncové body, které se vyskytují v datech a vytvoříme z nich uzly grafu, buď přímo nebo aplikujeme úpravy probrané v podsekci 8.1.3.

Z vytvořeného seznamu uzlů začneme tvořit grafy, pro každou třídu jeden. Nejdříve přidáme uzly START a KONEC. Poté přidáme všechny hrany, tj. vytvoříme úplný graf. Na začátku budou hodnoty všech hran nulové. Dále zavedeme i hodnoty pro uzly, na začátku také nulové. Hodnoty hran v tomto okamžiku nebudou označovat pravděpodobost, ale počet návštěv, které danou hranu obsahují. Hodnoty uzlů budou označovat počet požadavků na daný uzel.

Poté už pro každou třídu napočítáme hodnoty hran a uzlů tak, že pro každou návštěvu provedeme operace předepsané pseudokódem, popsaném v Algoritmu 1.

Vstup: seznam požadavků S, graf G

```
1  seřadit S podle času;
2  P_a := START ;                          // aktuální požadavek
3  while zbývají požadavky v S do
4  |  P_n := první požadavek z S ;         // následující požadavek
5  |  inkrementuj hodnotu uzlu P_a v G;
6  |  inkrementuj hodnotu hrany P_a → P_n z G;
7  |  P_a := P_n;
8  end
9  inkrementuj hodnotu uzlu P_a v G;
10 inkrementuj hodnotu hrany P_a → KONEC v G;
```

Algoritmus 1: Trénování klasifikátoru

Po načtení všech návštěv vypočteme finální pravděpodobnosti prostým vydělením hodnot hran hodnotami uzlů, ze kterých vycházejí. Hodnota uzlu A udává jeho celkovou návštěvnost, hodnota hrany $A \rightarrow B$ udává, kolik uživatelů se po navštívení A přesunulo na B. Tento údaj budeme považovat za pravděpodobnost takového přesunu.

Uvedeme na příkladu, kde budeme klasifikovat uživatele lidské od robotů. Jako trénovací data máme 6 návštěv, 3 lidských uživatelů a 3 robotů.

Návštěvy lidských uživatelů:

```
15:21:10 GET /index  HTTP/1.1
15:22:10 GET /styl.css  HTTP/1.1
15:22:15 GET /script.js  HTTP/1.1
15:22:30 GET /obrazek.png  HTTP/1.1
15:21:10 GET /chat/zprava  HTTP/1.1

17:05:05 GET /index  HTTP/1.1
17:05:06 GET /chat/zprava  HTTP/1.1
17:05:20 POST /chat/zprava  HTTP/1.1
17:05:35 POST /chat/zprava  HTTP/1.1
17:06:02 GET /chat/zprava  HTTP/1.1
17:07:20 POST /chat/zprava  HTTP/1.1
17:07:38 GET /chat/zprava  HTTP/1.1

18:55:38 GET /index  HTTP/1.1
18:56:06 GET /chat/zprava  HTTP/1.1
```

```
18:59:20 POST /chat/zprava  HTTP/1.1
19:02:30 GET /obrazek.png  HTTP/1.1
19:02:49 POST /chat/zprava  HTTP/1.1
19:05:01 GET /chat/zprava  HTTP/1.1
```

Návštěvy robotů:

```
13:02:45 GET /robots.txt  HTTP/1.1
13:03:55 GET /index  HTTP/1.1
13:04:15 GET /obrazek.png  HTTP/1.1
13:05:57 GET /index  HTTP/1.1
13:07:30 GET /obrazek.png  HTTP/1.1
```

```
14:30:55 GET /index  HTTP/1.1
14:31:08 GET /chat/zprava  HTTP/1.1
14:35:45 GET /chat/zprava  HTTP/1.1
14:35:49 GET /chat/zprava  HTTP/1.1
14:35:51 GET /chat/zprava  HTTP/1.1
14:35:55 GET /chat/zprava  HTTP/1.1
```

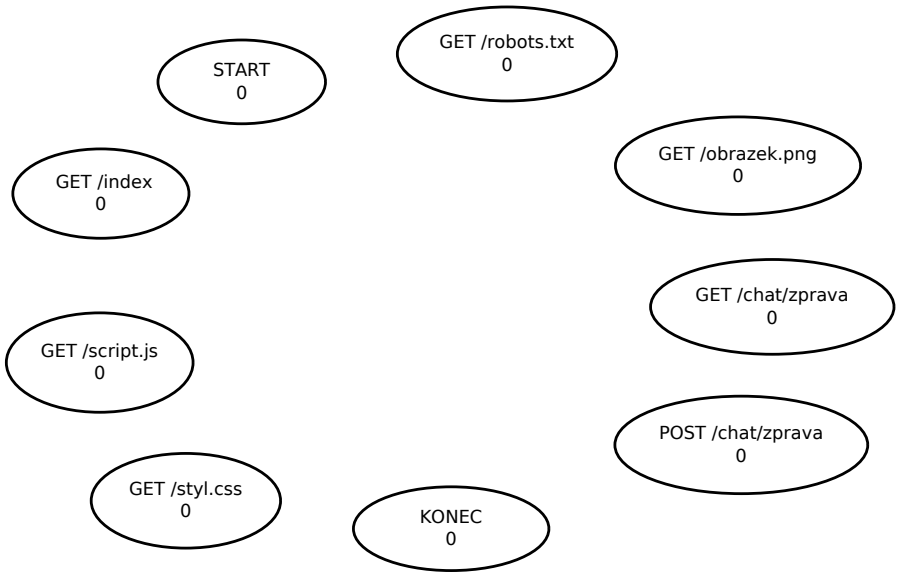
```
14:30:55 GET /robots.txt  HTTP/1.1
14:31:08 GET /index  HTTP/1.1
14:35:45 GET /obrazek.png  HTTP/1.1
```

Sestavíme seznam uzlů a přidáme uzly START a KONEC. Seznam uzlů tvoří:

- `GET /index`,
- `GET /styl.css`,
- `GET /script.js`,
- `GET /obrazek.png`,
- `GET /chat/zprava`,
- `POST /chat/zprava`,
- `GET /robots.txt`,

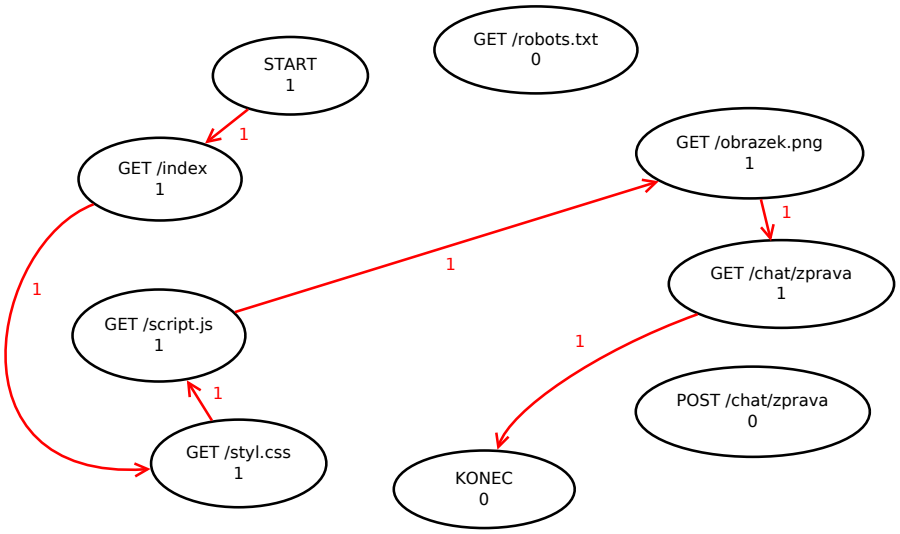
- START,
- KONEC.

Z tohoto seznamu vytvoříme graf, znázorněný na obrázku 8.3. Všechny hodnoty uzlů i hran jsou zatím rovny nule, a proto nejsou v obrázku žádné hrany zakresleny.



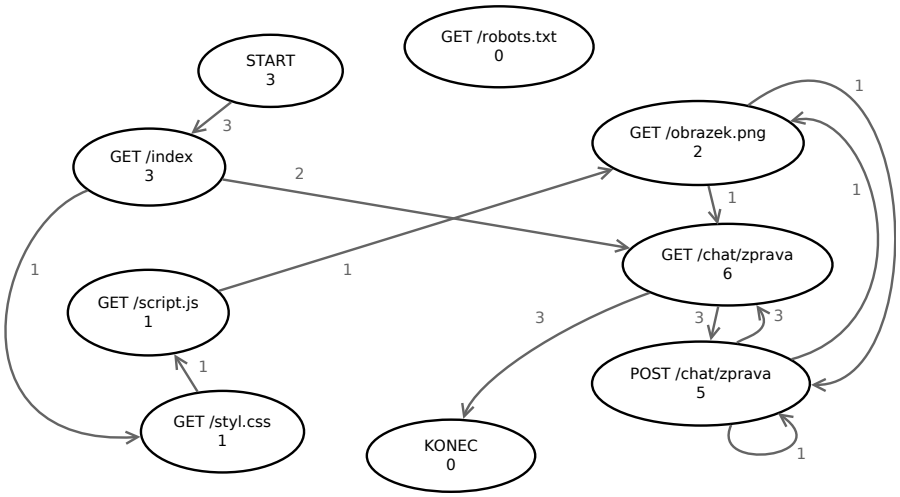
Obrázek 8.3: Vytvořený graf

První lidská návštěva začíná požadavkem na `/index`. Zvýšíme hodnotu uzlu START na 1 a hodnotu hrany START → `/index` také na 1. Další požadavek je na `/styl.css`, inkrementujeme hodnotu uzlu `/index` a hodnotu hrany `/index` → `/styl.css`. Takto pokračujeme až na konec návštěvy, kde hodnoty grafu jsou znázorněny na obrázku 8.4.



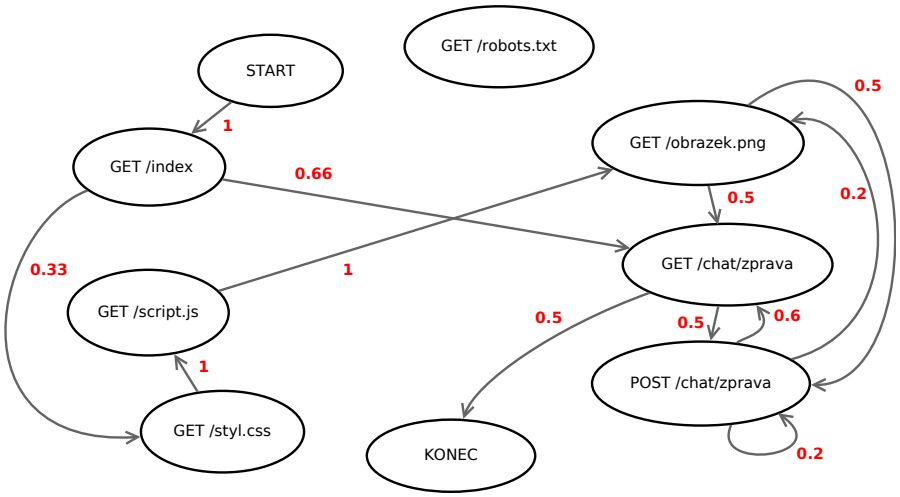
Obrázek 8.4: Hodnoty hran po zpracování první návštěvy

Takto pokračujeme i pro další dvě návštěvy. Výsledný graf po trénování třídy pro lidské uživatele je na obrázku 8.5.



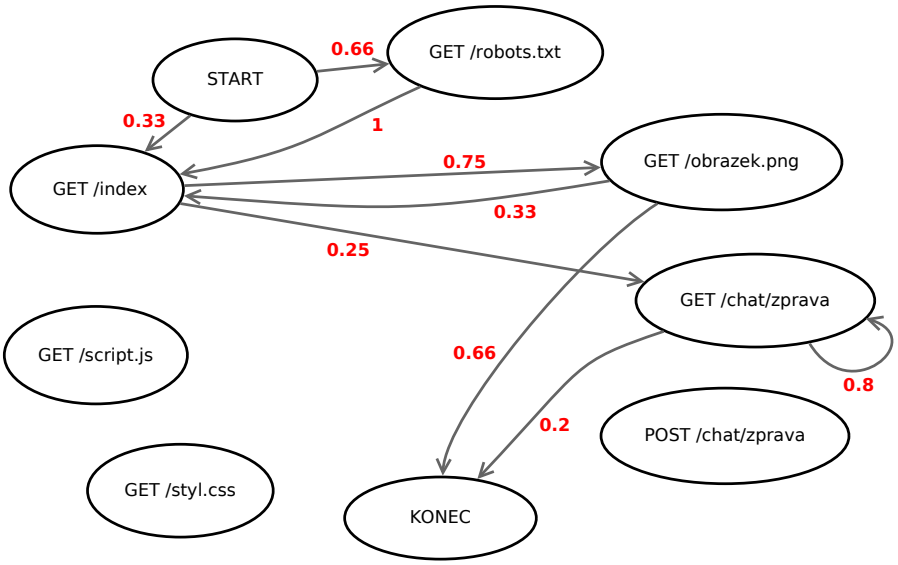
Obrázek 8.5: Hodnoty hran po zpracování třídy pro lidské uživatele

Po vydělení hodnot hran hodnotami počátečních uzlů dostaneme hodnoty pravděpodobnosti a finální podobu grafu (viz obrázek 8.6).



Obrázek 8.6: Pravděpodobnosti přechodů pro lidské uživatele

Stejným způsobem vytvoříme graf pro robotické návštěvníky (obrázek 8.7).



Obrázek 8.7: Pravděpodobnosti přechodů pro roboty

8.4 Klasifikace

Po natrénování na trénovacích datech, tj. vytvoření grafů pravděpodobností pro všechny třídy, můžeme začít s klasifikací. Nejdříve musíme vytvořené grafy mírně upravit přidáním uzlu NEZNÁMÝ. Všechny hrany vedoucí do tohoto uzlu budou mít nulovou pravděpodobnost, ovšem součet hodnot odchozích hran jednoho uzlu musí dávat pravděodobnost rovnou jedné. Neznámé údaje, které se nevyskytovaly v trénovacích datech, můžeme pouze odhadovat, a tak zavedeme, že pravděpodobnost přechodu z uzlu NEZNÁMÝ do kteréhokoliv jiného uzlu (kromě START) bude stejná a rovná *1 / počet uzlů*. Vzhledem k tomu, že pravděpodobnosti se mezi sebou násobí a přechodem přes uzel NEZNÁMÝ, kde pravděpodobnost všech hran vedoucích do něj je nulová, nezáleží na zvoleném rozdělení v podstatě vůbec, jelikož celková pravděpodobonost bude stejně nula.

Nyní použijeme podobný algoritmus jako u trénování, pouze s tím rozdílem, že budeme kontrolovat existenci uzlu a místo úprav grafu budeme násobit pravděpodonosti všech hran, přes které projdeme. Znázorněno pseudokódem v Algoritmu 2.

Vstup: seznam požadavků S, graf G
Výstup: celková pravděpodonost návštěvy pro danou třídu

```
1  seřadit S podle času;
2  P_a := START ;                          // aktuální požadavek
3  ppst_p := 1 ;                           // pravděpodobnost
4  while zbývají požadavky v S do
5  |  P_n := první požadavek z S ;         // následující požadavek
6  |  if P_n se nevyskytuje v G then
7  |  |  P_n := NEZNÁMÝ;
8  |  end
9  |  ppst_p := ppst_p × hodnota hrany P_a → P_n z G;
10 |  P_a := P_n;
11 end
12 ppst_p := ppst_p × hodnota hrany P_a → KONEC z G;
```

Algoritmus 2: Klasifikace

Tento postup zopakujeme s klasifikovanou návštěvou pro každou třídu a z výsledné množiny pravděpodobností vybereme tu nejvyšší. Na grafech natrénovaných v minulé sekci budeme klasifikovat krátkou návštěvu skládající se z:

```
17:21:10 GET /index  HTTP/1.1
17:28:58 GET /chat/zprava  HTTP/1.1
```

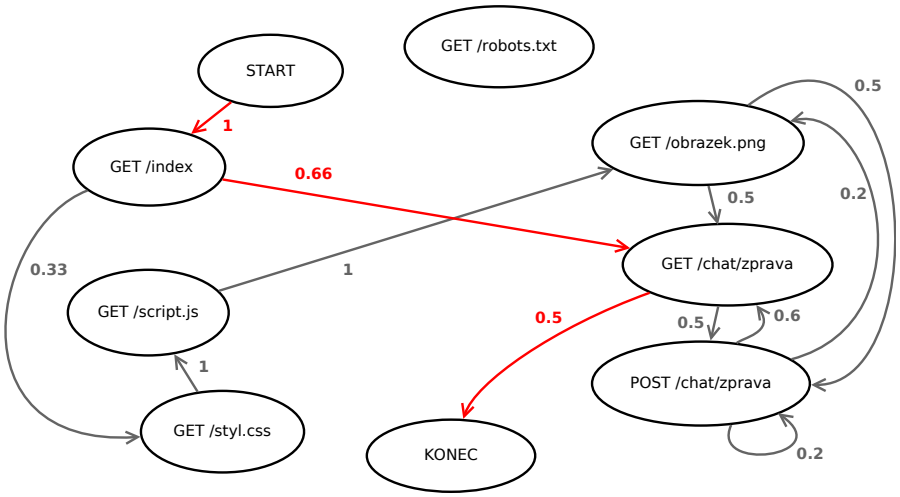
Projdeme grafem pro lidské uživatele, jak je znázorněno na obrázku 8.8, a po vynásobení pravděpodobností dostaneme

$$ppst_{clovek} = 1 \times 0,66 \times 0,5 = 0,33$$

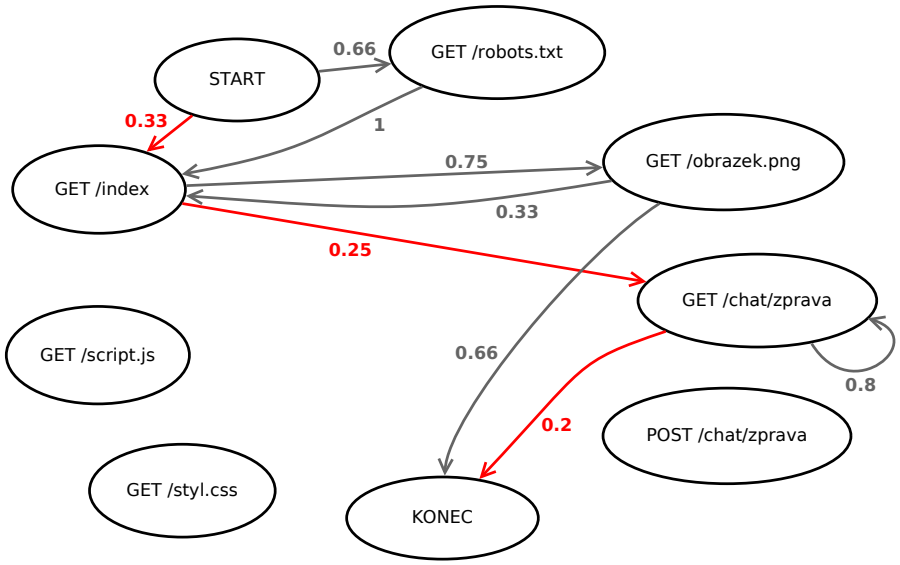
Obrázek 8.9 ukazuje průchod grafem pro roboty. Pravděpodobnost pro robota vyjde

$$ppst_{robot} = 0,33 \times 0,25 \times 0,2 = 0,0165$$

Budeme tedy klasifikovat jako lidského uživatele.



Obrázek 8.8: Průchod grafem pro lidské uživatele



Obrázek 8.9: Průchod grafem pro roboty

Oproti tomu pro návštěvu skládající se z požadavků:

```
23:45:55 GET /robots.txt  HTTP/1.1
23:46:15 GET /index  HTTP/1.1
23:47:28 GET /chat/zprava  HTTP/1.1
23:47:35 GET /chat/zprava  HTTP/1.1
23:47:48 GET /chat/zprava  HTTP/1.1
```

dostaneme celkové pravděpodobnosti:

$$ppst_{clovek} = 0 \times 0 \times 0,66 \times 0 \times 0 \times 0,5 = 0$$

$$ppst_{robot} = 0,66 \times 1 \times 0,25 \times 0,8 \times 0,8 \times 0,2 = 0,02112$$

Tuto návštěvu bychom tedy klasifikovali jako robota. Tento příklad ilustruje problém, kdy je k vytvoření grafu málo dat a tedy mnoho hran má nulovou hodnotu, jak zde vidíme pro lidské uživatele. Tento problém řeší vylepšení Markovova modelu v kapitole 9.

8.5 Další využití hodnot z grafu

Hodnoty pravděpodobností přechodů z grafu můžeme využít i jinak než jenom ke klasifikaci. Můžeme algoritmus pro trénování vyzkoušet i pro všechny logovací záznamy ze serveru, tj. jen pro jednu třídu, obsahující všechny návštěvy. z takového grafu se dají vyčíst vzorce chování průměrného uživatele.

Například pomocí hran z uzlu START se dá zjistit, kterou stránkou většina uživatelů svou návštěvu začne. Pomocí hran vedoucích do uzlu KONEC zase vidíme, na kterých stránkách většina uživatelů svou návštěvu končí.

Důležité hrany také mohou být ty s vysokou pravděpodobností, například vyšší než 90%. Ty mohou jednak značit stránky, které většina uživatelů navštíví hned po sobě a mohlo by být výhodné se zaměřit na průzkum důvodu tohoto postupu. Naříklad pokud mám na stránkách předpověď počasí na týden dopředu, ale zjistím, že 95% uživatelů klikne na podrobnou předpověď pro další den na speciální stránce, mohlo by být výhodnější zobrazit tuto denní předpověď jako první a teprve po vyžádání uživatele mu předložit předpověď týdenní.

Také jsou stránky, které ke svému fungování potřebují množinu skriptů a stylů. Pokud tyto skripty nebo styly jsou použity na více stránkách a vidíme, že existuje skoro 100% pravděpodobnost, že dané skripty nebo kaskádové styly jsou stahovány hned po sobě, mohlo by být výhodnější je sloučit do jednoho souboru (jednoho pro styly a jednoho pro skripty) a tím ušetřit HTTP požadavky a zátěž serveru.

Opačný případ, tedy když některé hrany mají velmi malé nebo nulové hodnoty, může administrátorovi pomoci s úpravou serveru. Pokud například v menu jsou položky, na které nikdo nikdy nekliknul, může to být důvod k jejich odstranění.

9 Vylepšený Markovův model

Základní Markovův model má několik nevýhod. Jedna z nevýhod může být, že se nedá nijak zvolit, v jakém poměru se uživatelé mají klasifikovat. Dále se nedá specifikovat výchozí třída (viz dále), kterou předpokládáme pro nového návštěvníka a „náročnost“ klasifikace, tj. jak moc se musí návštěva lišit, aby byla klasifikována jako jiná třída.

Uvedeme na příkladu koncového bodu URL `/robots.txt`. Tuto stránku navštěvují hlavně roboti a může se stát, že v testovacích datech pro lidské uživatele se nevyskytne ani jeden uživatel, který svoji návštěvu začne právě tímto URL. Pokud bude klasifikován uživatel, který se ze zvědavosti rozhodne na tuto stránku podívat, automaticky bude zařazen k robotům, i kdyby celý zbytek jeho návštěvy měl vyšší pravděpodobnost pro lidské návštěvníky. Ovšem díky nulové pravděpodobnosti pro hranu START → `/robots.txt`, pravděpodobnost pro člověka bude vždy nulová.

Další možností, kde klasifikace selže, bude návštěva uzlu NEZNÁMÝ. Tento uzel se v trénovacích datech nevyskytoval, tedy všechny hrany do něj vedoucí mají hodnotu nula, a proto celkové pravděpodobosti pro všechny třídy vyjdou opět nulové. Tento stav je možný a velmi pravděpodobný v případě, že se web změnil a některé stránky v něm jsou nové, tj. nebyly v trénovacích datech.

Stejný případ může nastat pokud návštěvník přistoupí na stránku v takovém pořadí, které se nevyskytlo v datech ani pro jednu třídu. Samotné koncové body se v grafu vyskytovat budou, ovšem tato jedna hrana může mít pro všechny třídy hodnotu nula.

V případě online klasifikace se může administrátor rozhodnout při klasifikaci uživatele jako robota s ním okamžitě ukončit spojení. Tento postup má tu nevýhodu, že během prvních pár požadavků může být větší pravděpodobnost pro robota, ovšem tato pravděpodobnost by se srovnala během dalších požadavků a uživatel by již byl správně klasifikován jako člověk. Jako příklad uvedeme sociální síť, kde většina lidí začíná na úvodní stránce přihlášením, ovšem roboti stahují údaje o uživatelích přes přímé odkazy na jejich profily. Když ovšem lidský uživatel dostane odkaz na nějaký profil jiného uživatele, na který by se chtěl podívat, začne svou návštěvu právě tímto prvním požadavkem na specifický profil. Tím pádem by hned u prvního požadavku byl klasifikován jako robot a na zbytek webu už by se nepodíval.

Požadovaných chováním by zde mohlo být to, aby se na několik prvních požadavků se uživatel vždy považoval za člověka a až poté co, bude nasbírán dostatek dat a klasifikátor bude rozhodovat s větší mírou jistoty, by se uživateli případně odepřel přístup.

9.1 Skóre

Základní Markovův model popsaný v minulé kapitole vylepšíme nahrazením celkové pravděpodobnosti novou hodnotou, nazvanou *skóre*. Skóre se narozdíl od pravděpodobnosti nebude násobit přes všechny hrany, ale sčítat. Tím se eliminuje problém s hranami s hodnotou nula, které nyní pouze nebudou přispívat ke klasifikaci k dané třídě, ale zároveň jí uplně nevyloučí.

Navíc zavedeme základní hodnotu pro skóre pro každou třídu, čímž určíme náš apriorní předpoklad o novém návštěvníkovi. Například základní hodnota skóre pro lidské návštěvníky bude 5, pro roboty 0 a zároveň budeme předpokládat hodnoty hran v intervalu od 0 do 1. Pokud tedy návštěvník začne hned po příchodu na web procházet web ve stylu robota, tj. přechody s hodnotami ≥ 0.9, bude klasifikován jako robot až po šesti takových požadavcích. Oproti tomu člověk, který začne procházet web neočekávaným způsobem, ale rychle se vrátí do stylu typického pro lidské uživatele, bude po celou dobu klasifikován jako člověk.

Tento postup se dá použít i pro případy klasifikace lidských uživatelů do tříd, například pro přístup do administrace nebo nějaké zabezpečené části webu, kde nejdříve předpokládáme, že uživatel není administrátor a až poté co projde správnou sekvencí požadavků (např. přihlášení) ho budeme klasifikovat jako administrátora. Tato metoda ovšem nezaručuje, že se jinému uživateli nepodaří napodobit styl procházení, a tedy bych tuto metodu nedoporučil jako jediný způsob zabezpečení citlivé části webu, ale vždy v kombinaci s dalšími metodami jako jsou hesla nebo jiné metody autorizace.

Úpravy algoritmu a metoda počítání skóre namísto pravděpodobnosti je znázorněna v Algoritmu 3. Samotná klasifikace zůstává stejná, pouze se místo pravděpodobností porovnávají hodnoty skóre a opět se vybírá nejvyšší hodnota.

Vstup: seznam požadavků S, graf G, počáteční skóre třídy s_p
Výstup: celková skóre návštěvy pro danou třídu

```
1  seřadit S podle času;
2  P_a := START ;                          // aktuální požadavek
3  skore := s_p;
4  while zbývají požadavky v S do
5  |  P_n := první požadavek z S ;         // následující požadavek
6  |  if P_n se nevyskytuje v G then
7  |  |  P_n := NEZNÁMÝ;
8  |  end
9  |  skore := skore + hodnota hrany P_a → P_n z G;
10 |  P_a := P_n;
11 end
12 skore := skore + hodnota hrany P_a → KONEC z G;
```

Algoritmus 3: Klasifikace pomocí skóre

9.2 Úpravy hodnot grafu

Hodnoty hran z grafu ukazují pravděpodobnosti přechodů, ovšem vzhledem k tomu, že počítáme skóre místo celkové pravděpodobnosti, můžeme si dovolit odstoupit od Markovova modelu a porušit pravidlo, že součet všech hodnot hran z jednoho uzlu se musí rovnat jedné. Hodnoty skóre tak nemusíme upravovat, normalizovat a hlavně omezovat na interval nula až jedna.

Jako administrátor také můžeme určit, které hrany jsou důležité pro danou třídu a jejich hodnoty upravit. Popřípadě rovnou přidat nebo odebrat nějaké uzly. Například se objeví nový typ útoku na webové servery, který se projeví vícenásobnými přístupy na koncový bod `/administrace`. Pro hranu START → `/administrace` a pro hranu `/administrace` → `/administrace` tak můžeme přidat skóre pro roboty na vysokou hodnotu (např. ≥ 10) a takový útok bude s největší pravděpodobností rozpoznán během prvních pár požadavků.

Jako jiný příklad můžeme uvést přidání celé nové sekce webu. V takovémto případě by bylo nutné znovu natrénovat klasifikátor, tj. opatřit nová testovací data a znovu vytvořit všechny grafy. Pokud ovšem nepotřebujeme klasifikovat nové třídy podle nové sekce na webu, ale stačilo by nám, aby např. nová sekce měla svou vlastní třídu a hlavně aby přístupy na ní

neovlivňovaly klasifikaci původních tříd, můžeme pro tuto novou sekci vytvořit jeden nový uzel, ve kterém bude hodnota hran nulová pro všechny stávající třídy a nenulová pro novou třídu. Tímto jednoduchým způsobem by se nám mělo podařit klasifikovat uživatele navštěvující novou sekci a zároveň výrazněji neovlivňovat původní klasifikaci.

Jako další příklad můžeme uvést změnu hodnot pro hrany vedoucí do uzlu NEZNÁMÝ. Budeme předpokládat, že roboti častěji navštěvují stránky, které se na webu nevyskytují. To je většinou pravda pro roboty, snažící se napadnout redakční systémy a administrátorské stránky tím, že se připojují na jejich obvyklé URL na webu. Jedná se například o koncové body `/administrator` nebo `/phpmyadmin`. Za tohoto předpokladu tedy zvýšíme skóre pro hrany vedoucí do uzlu NEZNÁMÝ pro třídu reprezentující roboty.

10 Testovací data

Jako testovací data byly použity logy z webového serveru Západočeské univerzity. Obsahují všechny požadavky od 17. prosince 2014 do 17. listopadu 2015, celkem je to 3 484 179 požadavků.

Logy jsou ze serveru Apache ve formátu *Combined Log Format*. Tento i další používané formáty serveru Apache jsou popsané podrobně v [26]. Ostatní formáty nebudou dále v práci rozebírány, ale představíme si Combined Log Format. V dalších sekcích probreme metody dolování návštěv z parsovaných logů a jejich ukládání. Dále probereme metody rozdělění návštěv do tříd a na trénovací a testovací data.

10.1 Combined Log Format

Tento formát logů zapisuje základní údaje o požadavku i odpovědi v jedné řádce do souboru s logy. Tato řádka je zapsána ve tvaru [26]:

```
%h %l %u %t "%r" %>s %b "%{Referer}i" "%{User-agent}i"
```

Ukládané informace jsou:

- `%h` – vzdálený host, tedy adresa připojeného klienta,
- `%l` – identifikace klienta podle programu `identd`, tato možnost je vysoce nespolehlivá a neměla by být používána kromě vysoce sledovaných interních sítí [26],
- `%u` – uživatelské jméno podle HTTP autentizace,
- `%t` – čas přijetí požadavku,
- `%r` – koncový bod požadavku (např. `"GET /index HTTP/1.1"`),
- `%>s` – kód HTTP odpovědi (200, 303, 404, ...),
- `%b` – velikost odpovědi v bytech,

- `%{Referer}i` – hodnota HTTP hlavičky `Referer` požadavku, tato hlavička má indikovat z jaké stránky byl požadavek generován (tj. nejčastěji předchozí navštívená stránka),
- `%{User-agent}i` – hodnota HTTP hlavičky `User-Agent` požadavku, tedy identifikaci prohlížeče nebo klienta.

Jedna řádka logu tak může vypadat například takto [26]:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] \
  "GET /apache_pb.gif HTTP/1.0" 200 2326 \
  "http://www.example.com/start.html" \
  "Mozilla/4.08 [en] (Win98; i ;Nav)"
```

Tato řádka říká, že 10. října 2000 v čase 13:55:36 (více o formátu zápisu času dále) poslal uživatel `frank` z adresy `127.0.0.1` požadavek `GET` na zdroj `/apache_pb.gif`. Jeho klient (prohlížeč) se identifikoval jako `"Mozilla/4.08 [en] (Win98; i ;Nav)"` a ohlásil, že požadavek pochází se stránky `"http://www.example.com/start.html"`. Odpovědí byl kód `200` a 2326 bytů dat.

10.1.1 Formátování času

Čas přijetí požadavku (`%t`) je formátován [26]:

```
[day/month/year:hour:minute:second zone]
```

kde jednotlivé položky mají formát:

- `day` – dvoumístné číslo, označující den v měsíci,
- `month` – třímístný řetězec označující měsíc,
- `year` – čtyřmístné číslo označující rok,
- `hour` – dvoumístné číslo označující hodinu,
- `minute` – dvoumístné číslo označující minutu,
- `second` – dvoumístné číslo označující sekundu,

- `zone` – „+“ nebo „-“ následované čtyřmístným ozačením časové zóny.

Časová zóna je označena rozdílem od Greenwich Mean Time (GMT), první dvě číslice označují rozdíl v hodinách, druhé dvě v minutách. Znak „+“ nebo „-“ označuje, zda se má rozdíl odečíst nebo přičíst.

Důležitá poznámka o ukládání požadavků a jejich časů v případě serveru Apache je, že požadavky jsou logovány až po odeslání odpovědí, ale logované časy jsou časy přijetí požadavků. To má význam zejména při jejich dalším zpracovávání (viz dále), kdy je nemůžeme pokládat za chronologicky seřazené a musíme si je před dalším zpracováváním seřadit.

10.2 Parsování logů a slovník koncových bodů

V záznamech se nachází informace, které ke klasifikaci aplikací nepotřebujeme a také hodnoty, které souvisí s odpovědí (její kód a velikost). Informace související s odpovědí by se nejspíš také daly využít ke klasifikaci, ovšem v případě online klasifikace je neznáme. Navíc všechny metody v této práci probírané s nimi nepočítají a zaměřují se pouze na chování klienta a ne na odpovědi serveru.

Klasifikace podle typu potřebuje z požadavku znát pouze cestu ke zdroji, která určuje jeho typ. Klasifikace podle času a klasifikace podle času s využitím statistických metod potřebuje k fungování pouze časy jednotlivých požadavků. Základní i vylepšený Markovův model vyžaduje typ požadavku a cestu ke zdroji. Vzhledem k tomu, že první řádek HTTP požadavku obsahuje přesně toto a navíc ještě verzi HTTP protokolu, budeme kvůli zjednodušení parsování a zlepšení klasifikace považovat požadavky se stejným typem, stejnou cestou ale jinou verzí HTTP protokolu za různé (tj. různé uzly v grafu webu, viz Markovův model v kapitole 8).

Pro všechny metody klasifikace si tedy vystačíme se třemi hodnotami z logů. Potřebujeme čas požadavku, celou první řádku požadavku (tj. koncový bod, např. `"GET /index HTTP/1.1"`) a IP adresu. IP adresu společně s časem potřebujeme kvůli dělení požadavků do návštěv. Pro uložení samotné návštěvy již nemusíme řešit IP adresu, ale stačí uložit seznam požadavků s korespondujícími časy.

10.2.1 Ukládání času

Nejčastější operace s časem, které se v práci provádí, jsou odečítání a porovnávání. Formát času (viz výše) popsaný textem se k tomuto účelu nehodí z důvodu nároků na parsování a zápisu času. Budeme proto čas převádět a ukládat ve unixovém formátu zpřesněném na milisekundy, tj. budeme čas zapisovat jako jedno číslo typu `long`, udávající počet milisekund od 1.1.1970 00:00:00.

Další věcí, kterou musíme při ukládání času řešit, jsou situace, kdy v logovacím souboru jsou zapsány požadavky se stejným časem. Jelikož Apache ukládá čas s přesností na sekundy, tento jev není neobvyklý. Víme, že požadavky jsou seřazeny v souboru s logy podle času vyřízení a pokud je čas obdržení požadavku v logu stejný, nevíme, který požadavek přišel první. Pokud bychom ukládali požadavky s časem, který je v logu uvedený, požadavky se stejným časem by mohly změnit pořadí v závislosti na tom, jestli řadící algoritmus je *stabilní*, tedy že u elementů se stejnou hodnotu dodržuje pořadí před a po seřazení. Další nevýhodou je to, že při ukládání požadavků do hashovacích tabulek a podobných struktur, které jsou závislé na unikátní hodnotě klíče, bychom mohli o tyto požadavky přijít tím, že by došlo k jejich přepsání dalším požadavkem se stejnou hodnotou klíče.

Budeme ale předpokládat, že čas vyřízení požadavků se tolik nemění a že pořadí v jakém jsou požadavky uloženy v souboru tedy má určitou vypovídající hodnotu. V případě, že požadavky zalogované za sebou mají stejný čas, budeme zachovávat jejich uspořádání tím, že ke každému dalšímu času požadavku přičteme milisekundu. Uvedeme na příkladu, kdy požadavky jsou zalogovány v časech:

- požadavek a – `15:25:30 0000`,
- požadavek B – `15:25:30 0000`,
- požadavek C – `15:25:30 0000`.

Tyto časy budou převedeny na časy:

- požadavek a – `15:25:30 0000`,
- požadavek B – `15:25:30 0001`,

- požadavek C – `15:25:30 0002`.

Jsou dva případy, kdy toto přičítání milisekund selže, a to v případě, kdy je v jednu sekundu zalogováno více než tisíc požadavků a v případě, kdy řadu požadavků se stejným časem přeruší požadavek s jiným časem (zalogovaný se zpožděním). Oba tyto případy jsou okrajové a nejsou v práci řešeny.

10.2.2 Slovník koncových bodů

Stejně jako zápis času, tak textový zápis koncových bodů (typ požadavku, cesta ke zdroji a verze HTTP protokolu, tedy celý první řádek HTTP požadavku) je nevyhovující. Zejména kvůli výpočetní náročnosti porovnávání řetězců a vyšším nárokům na paměť při jejich ukládání. Budeme tedy nahrazovat řetězce popisující koncové body čísly. Začneme od 0 a každému novému koncovému bodu přiřadíme další číslo v pořadí.

Kvůli klasifikaci pomocí typu souborů a také kvůli pozdější analýze výsledků klasifikace ovšem tento seznam čísel a k nim odpovídajících koncových bodů nemůžeme smazat, ale musíme ho uchovávat a používat při klasifikaci. Tento vygenerovaný seznam čísel a koncových bodů budeme nazývat slovník. Slovník vznikne hned při parsování logů a pro všechny další operace bude neměnný. Všechny úpravy koncových bodů (viz další sekce) musí proběhnout ještě před vytvořením slovníku a zapsány do něj musí být už upravené koncové body. V aplikaci je formát slovníku následující:

číslo koncového bodu␣*textová reprezentace koncového bodu*

Tedy například:

```
125 GET /favicon.ico HTTP/1.1
```

Každý koncový bod bude zapsán na nové řádce, počet řádek souboru slovníku tedy udává jeho velikost.

V případě úprav koncových bodů, jako například odstranění parametrů v URL musí tyto úpravy proběhnout před zápisem do slovníku, tedy požadavky ve tvaru:

`GET /forum.html?stranka=10 HTTP/1.1` a

```
GET /forum.html?stranka=12 HTTP/1.1
```

budou do slovníku zapsány jako jeden koncový bod s jedním číslem, například:

```
21 GET /forum.html HTTP/1.1
```

10.2.3 Použité úpravy koncových bodů

Úpravy koncových bodů jsou většinou závislé na daném serveru a měl by je určovat administrátor nebo programátor daného webu. V této práci je použita pouze jedna úprava dat z logů a to již zmíněné odstranění parametrů z cesty. Testovány jsou oba případy, s odstraněním i bez něj, a výsledky porovnány i mezi sebou.

10.3 Dolování návštěv z logů

Návštěvu máme definovanou jako všechny pořadavky z jedné IP adresy, ve kterých interval mezi požadavky nepřesáhne 30 minut. Budeme tedy log číst řádek po řádku a v prvním kroku rozdělovat podle zdrojové IP adresy. Zavedeme nový pojem *aktivní návštěva*, který bude označovat započatou, ale ještě neukončenou návštěvu a bude vázán k jedné IP adrese. Aktivních návštěv budeme uchovávat celou mapu (M_{active}), kde klíčem bude IP adresa, a postupně do nich přidávat požadavky. Pokud by časový interval mezi nově přidávaným požadavkem a posledním požadavkem z dané aktivní návštěvy přesáhl 30 minut, aktivní návštěvu ukončíme, zapíšeme do souboru, smažeme a nově přidávaný požadavek vytvoří první požadavek nové aktivní návštěvy. Postup je znázorněn v Algoritmu 4.

Vstup: seznam požadavků S
Výstup: návštěvy vydolované z logů ve výstupním souboru

```
1  M_active := prázdná mapa;
2  while zbývají požadavky v S do
3  |  P := načti požadavek z S;
4  |  Addr := adresa požadavku P;
5  |  if Addr se vyskytuje v M_active then
6  |  |  Visit := seznam požadavků z M_active pro Addr;
7  |  |  P_latest := poslední požadavek z Visit;
8  |  |  if rozdíl časů P_latest a P ≤ 30 minut then
9  |  |  |  přidej P do Visit;
10 |  |  else
11 |  |  |  zapiš Visit do výstupního souboru;
12 |  |  |  vymaž všechny požadavky z Visit;
13 |  |  |  přidej P do Visit;
14 |  |  end
15 |  else
16 |  |  vytvoř nový seznam požadavků Visit;
17 |  |  přidej P do Visit;
18 |  |  přidej Visit do M_active s adresou Addr jako klíčem;
19 |  end
20 end
21 zapiš všechny návštěvy z M_active do výstupního souboru;
```

Algoritmus 4: Dolování návštěv z logů

Jak bylo zmíněno výše, logy ovšem nemusí být v souboru chronologicky seřazené podle času požadavku. Řešením by mohlo být všechny požadavky z S nejdříve seřadit a rozdělovat do návštěv až poté. Vzhledem k velikosti dat by tato operace mohla trvat dlouho a mít vysoké nároky na výpočetní výkon a paměť.

Ovšem můžeme využít faktu, že S není úplně nahodilý seznam a časy zapsání požadavků do logu a jejich časy požadavků nebudou mít vysoké rozdíly. Nemusíme tedy procházet celý soubor s logy, ale stačí se podívat pouze několik záznamů dopředu. Zavedeme tedy nový seznam požadavků $S_{lookahead}$, který bude obsahovat např. 5000 požadavků právě načtených a zpracovávaných. Načtený požadavek z S se nejdříve uloží do $S_{lookahead}$, který se následně seřadí a k dalšímu zpracování jde první požadavek z $S_{lookahead}$. Tedy požadavky se zpracují ve správném pořadí podle času, pokud nejsou v logu od sebe vzdáleny více než pět tisíc řádek. Pomocí testovacích výpisů se podařilo ově-

řit, že v testovacích datech se nikdy nevyskytla situace, kdy by tento limit 5000 řádek nebyl dostačující.

Další vylepšení, které v aplikaci používáme, je, že za návštěvu se považuje pouze taková návštěva, která obsahuje alespoň pět požadavků. Metody podle času i Markovovy modely potřebují ke správnému fungování větší množství dat a hodnota pět byla zvolena jako hraniční. Algoritmus tedy upravíme na podobu znározněnou v Algoritmu 5.

Vstup: seznam požadavků S
Výstup: návštěvy vydolované z logů ve výstupním souboru

```
1  M_active := prázdná mapa;
2  S_lookahead := prázdný seznam;
3  while zbývají požadavky v S a velikost S_lookahead < 5000 do
4  |  P := načti požadavek z S;
5  |  přidej P do S_lookahead;
6  end
7  seřaď S_lookahead;
8  while zbývají požadavky v S_lookahead do
9  |  if zbývají požadavky v S then
10 |  |  P := načti požadavek z S;
11 |  |  přidej P do S_lookahead;
12 |  |  seřaď S_lookahead;
13 |  end
14 |  P := načti první požadavek z S_lookahead;
15 |  Addr := adresa požadavku P;
16 |  if Addr se vyskytuje v M_active then
17 |  |  Visit := seznam požadavků z M_active pro Addr;
18 |  |  P_latest := poslední požadavek z Visit;
19 |  |  if rozdíl časů P_latest a P ≤ 30 minut then
20 |  |  |  přidej P do Visit;
21 |  |  else
22 |  |  |  if počet požadavků ve Visit ≥ 5 then
23 |  |  |  |  zapiš Visit do výstupního souboru;
24 |  |  |  end
25 |  |  |  vymaž všechny požadavky z Visit;
26 |  |  |  přidej P do Visit;
27 |  |  end
28 |  else
29 |  |  vytvoř nový seznam požadavků Visit;
30 |  |  přidej P do Visit;
31 |  |  přidej Visit do M_active s adresou Addr jako klíčem;
32 |  end
33 end
34 zapiš všechny návštěvy z M_active do výstupního souboru;
```

Algoritmus 5: Dolování návštěv z logů s použitím *lookahead*

10.3.1 Formát ukládání návštěv

Každá návštěva je seřazený seznam požadavků a z každého požadavku ukládáme pouze dvě čísla – číslo koncového bodu ze slovníku a čas požadavku. Byl zvolen jednoduchý formát zápisu návštěv do souboru, kde každá návštěva je zapsána na jeden řádek jako seznam požadavků oddělených čárkami. Každý požadavek je zapsán jako číslo koncového bodu následované dvojtečkou a číslem označujícím čas. Zápis jedné návštěvy tak může vypadat:

```
19:1418838151000,874:1418838236000,875:1418838246000, \
     881:1418838347000,892:1418838722000, \
     19:1418839225000,905:1418839405000,908:1418839500000, \
     912:1418839543000,913:1418839573000,917:1418839678000,
```

10.4 Dělení návštěv na třídy

Jsou použity celkem tři metody rozdělení návštěv na třídy, které jsou využity k trénování a testování klasifikátorů. První metodou je rozdělení na návštěvy „slušných“ robotů, tj. návštěvy, které obsahují jeden nebo více požadavků na koncový bod `"GET /robots.txt HTTP/1.1"`.

V případě Markovova modelu by z této definice třídy všechny hodnoty hran v grafu z uzlu `/robots.txt` byly nulové pro všechny třídy kromě třídy pro roboty. To by mohlo značně zvýhodnit metodu Markovova modelu před ostatními klasifikátory a také ovlivnit její funkcionalitu, která by se tak potenciálně mohla omezit pouze na klasifikaci podle tohoto jednoho uzlu a hran do něj vedoucích. Pro lepší porovnání je tedy vytovřen další dataset, který je opět rozdělelen podle toho, zda návštěva obsahuje požadavek na `"GET /robots.txt HTTP/1.1"`, ovšem z takové návštěvy jsou všechny tyto požadavky vymazány. Klasifikátory tedy musí klasifikovat podle ostatních požadavků a dalších parametrů.

Jako další metoda klasifikace bylo zvoleno rozdělení uživatelů na ty, kteří navštíví jakoukoliv URL, která obsahuje text `"/services/phorum/read.php"`. Jedná se o internetové fórum a tato metoda rozdělení byla zvolena ze dvou důvodů. Jednak předpokládáme, že uživatelé na fóru se chovají jinak než při pasivním čtení stránek na webu a jedná se tak o dobrého kandidáta na klasifikaci na základě času pomocí statistických metod a pomocí Markovových modelů. Druhým důvodem je, že většina ostatních stránek byla navstšívena pouze ma-

Tabulka 10.1: Počty návštěv v jednotlivých třídách

	Hodnota
Velikost slovníku	183 964
Velikost slovníku po úpravě	64 243
Celkový počet návštěv	105 237
Třída „roboti“	7 133
Třída „roboti“ po odtstranění koncového bodu	4 303
Třída „lidé“	98 104
Třída „fórum“	19 390
Třída „ostatní“	85 847

lým okruhem uživatelů a klasifikátory by neměly dostatek dat pro správné určení parametrů třídy. Rozdělení tímto způsobem rozdělí návštěvy (na neupraveném slovníku) na 19 390 návštěv na fórum a 85 847 návštěv, které na fórum nepřistupují, což je pro oba případy dostatečný počet.

Zároveň také většina koncových bodů na fórum obsahuje parametry v URL a na tomto příkladu je vidět vliv jejich odmazání nebo neopak ponechání při vytváření slovníku.

Celkem máme dvě verze slovníku, jeden s nezměněnými koncovými body a druhý upravený tak, že jsou z URL požadavků vymazány všechny parametry. Pro oba tyto případy jsou výše popsanými metodami vygenerovány tři množiny dat. Celkové množství datasetů je tedy šest. Velikost slovníku a počty návštěv v jednotlivých třídách jsou zapsány v tabulce 10.1. Důvod, proč je počet návštěv ve třídě „roboti“ menší po odstranění koncového bodu, je zmenšení velikosti návštěvy pod hranici minimálně pěti požadavků a celá návštěva tedy byla odstraněna z dat.

10.5 Dělení návštěv na trénovací a testovací

Data jsou dále rozdělena na trénovací a testovací v přibližném poměru 10:1. Toto rozdělení je ovlivněno zvolením počátečního parametru pseudonáhodného generátoru (*seedu*). Pro účely testování, pokud není specifikováno jinak, je tato hodnota vždy stejná, takže všechny klasifikátory pro všechny metody mají přesně stejná data, což vede ke snadné reprodukci testů. Výsledky testů uvedených v kapitole 11 byly vypočítány při seedu nastaveném na hodnotu `10`.

11 Testy

11.1 Měřené hodnoty

Ve všech případech se jedná o binární klasifikaci (tj. rozdělujeme návštěvy do dvou tříd) a ve všech případech je jako první třída uvedena „hledaná“ třída (roboti nebo návštěvníci fóra) proti zbytku. Měřené hodnoty jsou:

- *True positive (TP)* – počet správně označených prvků hledané třídy,
- *True negative (TN)* – počet správně označených prvků druhé třídy,
- *False positive (FP)* – počet nesprávně označených prvků druhé třídy,
- *False negative (FN)* – počet nesprávně označených prvků hledané třídy,
- *Precision (P)* – poměr nalezených skutečně hledaných prvků k celkovému počtu nalezených prvků, vypočteno jako $TP/(TP + FP)$,
- *Recall (R)* – poměr nalezených skutečně hledaných prvků k celkovému počtu hledaných prvků, vypočteno jako $TP/(TP + FN)$,
- *F1 measure (F1)* – harmonický průměr *Precision* a *Recall* (viz dále),
- *Matthews correlation coefficient (MCC)* – používaný koeficient pro binární klasifikaci vhodný i pro množiny s velice odlišnou velikostí (viz dále).

11.1.1 F1 measure

Většina klasifikátorů má parametry, kterými lze ovlivnit poměry prvků v obou množinách, tedy jak moc je klasifikace „náročná“ – jak moc se musí prvek lišit aby byl zařazen do druhé třídy. Změna těchto parametrů většinou ovlivňuje obě hodnoty *Precision* i *Recall*, a to v tom smyslu, že jednu zlepšuje na úkor druhé. Pro co nejlepší učení vyrovnaného skóre se používá jejich harmonický průměr, také označován jako *F1 measure*. Jeho výpočet je následující:

$$F1 = 2 \times \frac{P \times R}{P + R}$$

Hodnoty F1 skóre jsou v rozmezí nuly až jedné, kde nula je nejhorší a jedna nejlepší klasifikátor (tedy klasifikátor, který určil všechny prvky do správných tříd) [27].

11.1.2 Matthews correlation coefficient

Jak je vidět z výpočtů *Precision*, *Recall* a *F1 measure*, tyto metody neberou v potaz hodnotu *true negative*. Dále jsou negativně ovlivněny v případě, že velikosti obou tříd jsou (řádově) rozdílné. V tom případě úspěšnost klasifikace lépe vyjadřuje *Matthews correlation coefficient* [27], vypočtený následovně:

$$N = TP + TN + FP + FN$$

$$S = \frac{TP + FN}{N}$$

$$P = \frac{TP + FP}{N}$$

$$MCC = \frac{TP/N - S \cdot P}{\sqrt{P \cdot S \cdot (1 - S) \cdot (1 - P)}}$$

Hodnoty MCC jsou v rozsahu od -1 do 1, kde 1 znamená přesnou klasifikaci, 0 stejné skóre jako kdyby klasifikátor rozhodoval náhodně a -1 je hodnota pro přesnou neshodu v klasifikaci, tedy, že všechny prvky byly klasifikovány do nesprávné třídy. V případě, že všechny prvky byly klasifikovány do první třídy a tedy $P = N/N = 1$ a $(1-P) = 0$ by došlo k dělení nulou a v takových případech budeme koeficient do tabulek psát jako *NaN (Not a Number)*. V případě řazení (při trénování parametrů, viz kapitola 12) ho nahradíme hodnotou -2.

11.2 Nezměněný slovník

Hodnoty pro všechny klasifikátory na datasetu s nezměněným slovníkem a klasifikací robotů od lidí ukazuje tabulka 11.1.

Hodnoty pro klasifikaci na stejném, neupraveném slovníku, stejném problému klasifikace robotů a lidí, ale v případě, kdy byly vymazány koncové body `GET /robots.txt HTTP/1.1`, ukazuje tabulka 11.2.

Tabulka 11.3 ukazuje hodnoty pro klasifikaci návštěv uživatelů fóra a zbytku uživatelů opět na nezměněném slovníku.

Všechny hodnoty v tabulkách zmíněných výše byly testovány na datech rozdělených na testovací a trénovací pomocí pseudonáhodného generátoru s rovnoměrnou distribuční funkcí nastaveného na seed `10` (viz sekce 10.5). K ověření, že nedošlo k *overfittingu* na testovacích datech, byly klasifikátory se stejnými parametry otestovány ještě dvakrát na stejných datech, ovšem jinak rozdělených na trénovací a testovací.

Overfitting (česky *přeučení*) znamená, že klasifikátor je příliš přizpůsobený testovacím datům a malé a z celkového pohledu nedůležité rozdíly v testovacích datech jsou používány ke klasifikaci namísto hlavních vlastností klasifikovaných objektů. Takový klasifikátor dosahuje dobrých výsledků na jednom setu testovacích dat, ovšem selže při generalizaci (tj. zobecnění a nasazení na dalších testovaných případech).

Výsledky těchto testů nejsou pro přehlednost uvedeny zde, ale v příloze A. Testovány byly dvě další hodnoty *seedu* a to `11` a `12`. z tabulek A.1, A.2, A.3, A.4, A.5 a A.6 jasně vyplývá, že problémem overfittingu značně trpí jednoduchý klasifikátor na základě časů požadavků. Ostatní metody klasifikace jsou v tomto ohledu konzistentní.

Tabulka 11.1: Výsledky klasifikace na neupraveném slovníku a datasetu klasifikace robotů od lidí

	Metoda klasifikace				
	Podle typu	Podle času	Statistické metody	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	91,20%	39,74%	59,38%	100%	99,71%
Úspěšnost na druhé třídě	60,21%	74,99%	69,36%	0%	99,73%
True positive	622	271	405	682	680
True negative	6 026	7 504	6 941	0	9 980
False positive	60	411	277	0	2
False negative	3 981	2 503	3 066	10 007	27
Precision	0,91	0,40	0,59	1,00	0,99
Recall	0,14	0,10	0,11	0,06	0,96
F1 measure	0,24	0,16	0,20	0,12	0,98
Matthews correlation c.	0,25	0,08	0,15	*NaN*	0,98

Tabulka 11.2: Výsledky klasifikace na neupraveném slovníku a datasetu klasifikace robotů od lidí s vymazáním URL pro `robots.txt`

	Metoda klasifikace				
	Podle typu	Podle času	Statistické metody	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	70,16%	14,69%	69,00%	100%	70,40%
Úspěšnost na druhé třídě	57,18%	88,57%	69,38%	0%	72,76%
True positive	301	63	296	429	302
True negative	5 696	8 823	6 912	0	7 248
False positive	128	366	133	0	127
False negative	4 266	1 139	3 050	9 962	2 714
Precision	0,70	0,15	0,69	1,00	0,70
Recall	0,07	0,05	0,09	0,04	0,10
F1 measure	0,12	0,08	0,16	0,08	0,18
Matthews correlation c.	0,11	0,02	0,16	*NaN*	0,19

Tabulka 11.3: Výsledky klasifikace na neupraveném slovníku a datasetu klasifikace uživatelů fóra od zbytku

	Metoda klasifikace				
	Podle typu	Podle času	Statistické metody	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	73,58%	69,53%	77,79%	100%	83,21%
Úspěšnost na druhé třídě	56,13%	74,04%	75,57%	0%	76,37%
True positive	1 398	1 321	1 478	1 900	1 581
True negative	4 934	6 507	6 642	0	6 712
False positive	502	579	422	0	319
False negative	3 855	2 282	2147	8 789	2 077
Precision	0,74	0,70	0,78	1,00	0,83
Recall	0,27	0,37	0,41	0,18	0,43
F1 measure	0,39	0,48	0,53	0,30	0,57
Matthews correlation c.	0,23	0,35	0,43	*NaN*	0,48

11.3 Vymazané parametry z URL

V případě použité úpravy slovníku odmazáním všech parametrů z URL v požadavcích není potřeba testovat všechny klasifikátory znovu. Na klasifikátory pracující podle časových intervalů mezi požadavky, tj. jednoduchý klasifikátor podle času a klasifikátor rozdělující pomocí statistických metod, nemají úpravy slovníku žádný vliv. Časové prodlevy mezi požadavky zůstávají stejné.

Na klasifikátor dělící návštěvy podle typů stažených souborů také tato úprava nemá žádný vliv, jelikož daný klasifikátor stejně parametry odstraňuje před samotnou klasifikací.

V tabulkách 11.4, 11.5 a 11.6 jsou uvedeny hodnoty pro klasifikátory na principu Markovova modelu a vylepšeného Markovova modelu.

Tabulka 11.4: Výsledky klasifikace na upraveném slovníku a datasetu klasifikace robotů od lidí

	Metoda klasifikace	
	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	100%	97,95%
Úspěšnost na druhé třídě	0%	98,35%
True positive	682	668
True negative	0	9 842
False positive	0	14
False negative	10 007	165
Precision	1,00	0,98
Recall	0,06	0,80
F1 measure	0,12	0,88
Matthews correlation c.	*NaN*	0,88

Tabulka 11.5: Výsledky klasifikace na upraveném slovníku a datasetu klasifikace robotů od lidí s vymazáním URL pro `robots.txt`

	Metoda klasifikace	
	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	100%	74,59%
Úspěšnost na druhé třídě	0%	74,58%
True positive	429	320
True negative	0	7 430
False positive	0	109
False negative	9 962	2 532
Precision	1,00	0,75
Recall	0,04	0,11
F1 measure	0,08	0,20
Matthews correlation c.	*NaN*	0,22

Tabulka 11.6: Výsledky klasifikace na upraveném slovníku a datasetu klasifikace uživatelů fóra od zbytku

	Metoda klasifikace	
	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	100%	94,00%
Úspěšnost na druhé třídě	0%	94,21%
True positive	1 900	1 786
True negative	0	8 280
False positive	0	114
False negative	8 789	509
Precision	1,00	0,94
Recall	0,18	0,78
F1 measure	0,30	0,85
Matthews correlation c.	*NaN*	0,82

12 Trénování parametrů klasifikátorů

Většina zde uvedených klasifikátorů má nastavitelné parametry. V případě klasifikace podle typu stahovaných souborů jsou to váhy pro jednotlivé přípony a základní hodnota pro třídu. Jednoduchý klasifikátor na základě intervalů v časech požadavků potřebuje definovat váhy pro střední hodnoty a směrodatné odchylky pro každou třídu. Pro klasifikátor používajícího statistické metody je potřeba nastavit očekávanou funkci nebo (v případě použití průměru pro lidské návštěvníky) hodnotu prahu. Klasický Markovův model parametry nemá, ovšem pro vylepšený Markovův model můžeme nastavit počáteční skóre pro každou třídu a skóre pro hrany vedoucí do uzlu NEZNÁMÝ.

Všechny tyto hodnoty se dají odvodit z typického chování robotů a/nebo uživatelů daného webu, ze zkušeností a znalostí administrátora nebo dalších studií. Máme-li k dispozici trénovací označkovaná data, můžeme také využít možnosti strojového učení a parametry klasifikátoru natrénovat.

12.1 Použitá trénovací metoda

Metoda použitá v této práci k trénování parametrů vychází z náhodného prohledávání stavového prostoru, ovšem postupně se zmenšuje prohledávaný prostor na základě nalezených řešení a jejich výsledků.

Metoda začíná náhodným zvolením např. deseti možných řešení, vypočtením jejich cenové funkce (viz dále) a seřazením. Spodních pět hodnot je zahozeno a z pěti nejlepších řešení se vytvoří po dvou kopiích. Takto získaných nových deset řešení se náhodně posune ve stavovém prostoru. Směr posunu je náhodný ovšem velikost posunu je daná hodnotou α. Celý proces se opakuje s tím rozdílem, že hodnota α se v každé iteraci snižuje. Tím je zajištěno, že na začátku prohledávání se s velkou pravděpodobností vyzkouší všechny části daného prostoru řešení, ale jak výpočet pokračuje, už se pouze zpřesňují dříve nalezená řešení (tj. prohledávají se malé oblasti okolo nich). Použitý algoritmus hodnotu α v každé iteraci půlí.

Výpočet končí po daném množství iterací, které je navázané na α, tj.

pokud by hodnota α byla tak malá, že už se řešení nijak výrazně neposunují, algoritmus končí. Algoritmus si pamatuje nejlepší řešení z celého průběhu, které je na konci vráceno jako nejlepší nalezené.

Pokud mluvíme o posunutí řešení, máme na mysli úpravu všech parametrů o nějakou hodnotu. Tato hodnota je závislá na rozsahu hodnot daného parametru, parametru α a pseudonáhodné funkci. Jako taková funkce byla zvolená gaussovská funkce, kvůli svému rozložení. Jako střední hodnota se zvolí aktuální hodnota upravovaného parametru a směrodatná odchylka je závislá na α. Jak již bylo řečeno, tím je zajištěno, že na začátku algoritmu (při velké hodnotě α) se s velkou pravděpodobností řešení posune dál v prohledávaném prostoru. Na konci algoritmu je díky malé odchylce pravděpodobnost velkého posunu malá a je prohledáván menší prostor okolo upravovaného řešení.

12.2 Použité cenové funkce

12.2.1 Vynásobení úspěšností

Jako cenová funkce je použit výsledek testování klasifikátoru na testovacích datech, a to úspěšnosti pro obě třídy. Vrácené hodnoty jsou tedy dvě – úspěšnost rozpoznávání první a druhé třídy. Cenová funkce se vypočte jejich vynásobením. Tím je zajištěno, že pokud nějaká úspěšnost vyjde nulová bude i výsledná cenová funkce nulová a řešení bude s nějvětší pravděpodobností zahozeno. Nejvyšší dosažitelná úspěšnost je tedy jedna, tj. stoprocentní určení lidí i robotů bez jediné chyby.

12.2.2 F measure a Matthews correlation coefficient

Obě tyto metriky jsou popsány v kapitole 11. Parametry klasifikátorů uvedených v této kapitole a používaných při měření v kapitole 11 byly trénovány a upravovány k maximalizaci právě *Matthews correlation coefficient*. Zároveň se jedná o výchozí používanou metodu v aplikaci, všechny ostatní vyžadují změnu ve zdrojovém kódu.

12.3 Trénované parametry

Všechny následující parametry klasifikátorů byly zvoleny kombinací ručního testování parametrů, odhadování na základě znalosti chování robotů a popsaného algoritmu k trénování.

Nastavení klasifikátorů je předmětem preferencí administrátora a typu serveru. Například pro sociální síť může být lepší povolit přístup několika robotům, než nastavit klasifikátor přísnější a riskovat, že na stránky nebudou vpuštěni lidští uživatelé. Naopak web banky nebude riskovat povolení přístupu pro rizikové uživatele i za cenu toho, že oprávněný uživatel nebude občas chybně vpuštěn.

Zde zvolené parametry se snaží maximalizovat Matthews correlation coefficient, tedy nerozlišovat mezi třídami a hledat takové nastavení, které klasifikuje obě třídy stejně úspěšně.

12.3.1 Klasifikátor na základě typů souborů

V tomto klasifikátoru bylo zvoleno pět typů souborů, u kterých se předpokládá, že mají největší vliv na klasifikaci a to `txt`, `html`, `pdf`, `css` a `js`. Pro každý tento typ se trénují hodnoty vah pro obě třídy stejně jako výchozí hodnoty pro obě třídy. Nejlepší nalezené hodnoty jsou zobrazeny v tabulkách 12.1, 12.2 a 12.3.

Tabulka 12.1: Váhy pro klasifikaci podle typu, dataset klasifikace lidí od robotů

Typ souboru	Váha pro lidského uživatele	Váha pro robota
`html`	9,5	9,1
`pdf`	4,0	5,0
`js`	10,0	0,0
`css`	10,0	0,0
`txt`	1,8	10,0

Tabulka 12.2: Váhy pro klasifikaci podle typu, dataset klasifikace lidí od robotů s vymazáním koncového bodu pro `robots.txt`

Typ souboru	Váha pro lidského uživatele	Váha pro robota
`html`	8,4	9,0
`pdf`	1,9	0,0
`js`	8,0	0,0
`css`	9,0	0,0
`txt`	1,8	5,0

Tabulka 12.3: Váhy pro klasifikaci podle typu, dataset klasifikace návštěvníků fóra od zbytku

Typ souboru	Váha pro návštěvníka fóra	Váha pro ostatní návštěvníky
`html`	3,4	2,9
`pdf`	0,0	9,0
`js`	2,1	10,0
`css`	0,9	0,6
`txt`	7,1	0,0

12.3.2 Jednoduchý klasifikátor na základě časů

Samotné hodnoty průměru a směrodatné odchylky pro každou třídu se trénují přímo v klasifikátoru, a to určením průměru pro všechny návštěvy dané třídy na trénovacích datech.

Nastavují se zde tedy pouze dvě hodnoty pro každou třídu, a to váha pro rozdíl naměřeného a očekávného průměru časových intervalů a váha pro rozdíl směrodatných odchylek. Hodnoty parametrů pro všechny případy testovacích dat jsou v tabulkách 12.4, 12.5 a 12.6.

Tabulka 12.4: Váhy pro jednoduchou klasifikaci podle časů, dataset klasifikace lidí od robotů

	Váha rozdílu průměrů	Váha rozdílu směr. odchylek
Lidský návštěvník	0,21	0,98
Robot	0,5265	0,5225

Tabulka 12.5: Váhy pro jednoduchou klasifikaci podle časů, dataset klasifikace lidí od robotů s vymazáním koncového bodu pro `robots.txt`

	Váha rozdílu průměrů	Váha rozdílu směr. odchylek
Lidský návštěvník	0,137	0,587
Robot	0,3483	0,397

Tabulka 12.6: Váhy pro jednoduchou klasifikaci podle časů, dataset klasifikace návštěvníků fóra od zbytku

	Váha rozdílu průměrů	Váha rozdílu směr. odchylek
Návštěvník fóra	0,428	0,97
Ostatní návštěvníci	0,51	0,3213

12.3.3 Klasifikátor využívající statistických metod

Je využit algoritmus, který rozděluje uživatele na základě průměru pro druhou (většinovou, tj. lidské uživatele a uživatele nenavštěvující fórum) třídu oproti všem ostatním. Zde se trénuje pouze hodnota prahu. Nejlepší hodnota pro dataset klasifikace robotů od lidí byla natrénována jako `500`. Pro dataset, kde byl odtraněn koncový bod pro `robots.txt` tato hodnota vyšla `600`. A pro klasifikaci uživatelů fóra od ostatních opět `600`.

12.3.4 Vylepšený Markovův klasifikátor

Hodnoty v grafu, kromě hran vstupujících do uzlu NEZNÁMÝ (viz dále), se trénují samotným algoritmem, takže těch se trénování parametrů nijak netýká. Parametry, které se trénují postupem zapsaným výše, jsou obě základní hodnoty pro roboty i lidské uživatele a dále hodnota pro všechny hrany vstupující do uzlu NEZNÁMÝ. Nejlepší nalezené hodnoty uvádí tabulky 12.7, 12.8 a 12.9.

Parametry klasifikátoru pro případ, kdy bylo použito odstranění parametrů URL jako úprava slovníku, ukazují tabulky 12.10, 12.11 a 12.12.

Tabulka 12.7: Váhy pro Vylepšený Markovův klasifikátor, dataset klasifikace lidí od robotů

	Hodnota hrany pro uzel NEZNÁMÝ	Základní skóre
Lidský návštěvník	0,0	0,001
Robot	0,0	0,0

Tabulka 12.8: Váhy pro Vylepšený Markovův klasifikátor, dataset klasifikace lidí od robotů s vymazáním koncového bodu pro `robots.txt`

	Hodnota hrany pro uzel NEZNÁMÝ	Základní skóre
Lidský návštěvník	0,0	0,005
Robot	0,001	0,0

Tabulka 12.9: Váhy pro Vylepšený Markovův klasifikátor, dataset klasifikace návštěvníků fóra od zbytku

	Hodnota hrany pro uzel NEZNÁMÝ	Základní skóre
Návštěvník fóra	0,0	0,0
Ostatní návštěvníci	0,0	0,01

Tabulka 12.10: Váhy pro Vylepšený Markovův klasifikátor, dataset klasifikace lidí od robotů, upravený slovník

	Hodnota hrany pro uzel NEZNÁMÝ	Základní skóre
Lidský návštěvník	0,0	0,225
Robot	0,0	0,0

Tabulka 12.11: Váhy pro Vylepšený Markovův klasifikátor, dataset klasifikace lidí od robotů s vymazáním koncového bodu pro `robots.txt`, upravený slovník

	Hodnota hrany pro uzel NEZNÁMÝ	Základní skóre
Lidský návštěvník	0,01	0,00
Robot	0,0	0,0

Tabulka 12.12: Váhy pro Vylepšený Markovův klasifikátor, dataset klasifikace návštěvníků fóra od zbytku, upravený slovník

	Hodnota hrany pro uzel NEZNÁMÝ	Základní skóre
Návštěvník fóra	0,0	0,1
Ostatní návštěvníci	0,0	0,0

12.4 Možné vylepšení trénovací metody

Zde uvedená trénovací metoda pro parametry byla zvolena díky své jednoduché implementaci a rychlosti. Dala by se nahradit gradientním výstupem, simulovaným žíháním, genetickým algoritmem, popřípadě kteroukoliv jinou metodou hledání maxima funkce.

Zároveň nebyla provedena optimalizace, která by zejména u některých klasifikátorů výpočet značně zrychlila. Například u vylepšeného Markovova modelu se netrénují hodnoty hran v grafu, takže by bylo možné je zachovávat přes iterace algoritmu a upravovat pouze parametry klasifikátoru, oproti stávající implementaci, kdy se hodnoty grafu přepočítávají (tj. trénují pokaždé znovu) v každé iteraci.

Na trénování parametrů klasifikátorů ovšem tato práce není zaměřena a nebyl jí proto věnován takový prostor jako samotným klasifikátorům. Uvedená metoda byla implementována díky své jednoduchosti na pochopení a implementaci.

13 Závěr

Z výsledků testování jasně vyplývá, že klasifikátor používající jednoduchý Markovův model nemá vůbec žádnou předpovídající schopnost. Ve všech případech byly všechny prvky klasifikovány do první třídy. Možné důvody, proč tomu tak je jsou popsány v kapitole 9 a jsou odstraněny vylepšením Makovova modelu.

Klasifikátor na základě jednoduchého rozdělování podle času dosahuje dobrých výsledků při správném nastavení, ovšem při porovnání s výsledky z přílohy A, je vidět, že zde došlo k overffitingu a tento algoritmus nemá dostatečnou schopnost generalizace.

Všechny ostatní klasifikátory dosahovaly konzistentních hodnot i při použití na datech, jinak rozdělených na trénovací a testovací, tedy vykazovaly dobrou schopnost generalizace. Hodnoty Matthewsova korelačního koeficientu i úspěšností pro jednotlivé třídy ukazují dobrou předpovídací schopnost a tyto klasifikátory by tedy poskytovaly přidanou hodnotu při nasazení v reálném prostředí.

Nejlepších hodnot dosahoval vylepšený Markovův klasifikátor, a to zejména na datasetu klasifikace lidí a robotů bez odstranění koncového bodu (MCC = 0,98) a na datasetu uživatelů fóra od zbytku při upraveném slovníku (MCC = 0,82). Je tedy vidět, že tento klasifikátor je nejvíce náchylný na změny trénovacích dat i úprav slovníku, ale při správném nastavení dosahuje nejlepších hodnot ze všech testovaných metod.

A Hodnoty klasifikátorů pro jiné rozdělení trénovacích a testovacích dat

Tabulky A.1, A.2, A.3, A.4, A.5 a A.6 ukazují data naměřená stejně jako v kapitole 11, ovšem data byla jinak rozdělena na trénovací a testovací. Všechny ostatní parametry klasifikátorů zůstaly zachovány.

Tabulky A.7, A.8, A.9, A.10, A.11 a A.12 opět ukazují naměřená data při jiném rozdělení dat, ovšem jedná se o data v případě, že byl upraven slovník odstraněním parametrů z URL. Stejně jako v kapitole 11 nejsou měřeny všechny klasifikátory, ale pouze ty, kterých se změna týká, tedy Markovova klasifikátoru a jeho vylepšené verze.

Tabulka A.1: Výsledky klasifikace na neupraveném slovníku a datasetu klasifikace robotů od lidí pro náhodný seed `11`

	Metoda klasifikace				
	Podle typu	Podle času	Statistické metody	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	89,67%	93,74%	60,70%	100%	99,71%
Úspěšnost na druhé třídě	60,16%	5,89%	68,78%	0%	99,75%
True positive	616	644	417	687	685
True negative	5 928	579	6 777	0	9 828
False positive	71	43	270	0	2
False negative	3 925	9 274	3 076	9 853	25
Precision	0,90	0,94	0,61	1,00	0,99
Recall	0,14	0,06	0,12	0,07	0,96
F1 measure	0,24	0,12	0,20	0,12	0,98
Matthews correlation c.	0,25	-0,004	0,15	*NaN*	0,98

Tabulka A.2: Výsledky klasifikace na neupraveném slovníku a datasetu klasifikace robotů od lidí pro náhodný seed 12

	Metoda klasifikace				
	Podle typu	Podle času	Statistické metody	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	91,46%	0,00%	57,30%	100%	99,42%
Úspěšnost na druhé třídě	59,09%	99,98%	68,89%	0%	99,76%
True positive	632	0	396	691	687
True negative	5 806	9 823	6 768	0	9 801
False positive	59	691	295	0	4
False negative	4 019	2	3 057	9 825	24
Precision	0,91	0,00	0,57	1,00	0,99
Recall	0,14	0,00	0,11	0,07	0,97
F1 measure	0,24	NaN	0,19	0,12	0,98
Matthews correlation c.	0,25	-0,004	0,14	*NaN*	0,98

Tabulka A.3: Výsledky klasifikace na neupraveném slovníku a datasetu klasifikace robotů od lidí s vymazáním URL pro `robots.txt` pro náhodný seed `11`

	Metoda klasifikace				
	Podle typu	Podle času	Statistické metody	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	70,74%	93,04%	61,39%	100%	69,54%
Úspěšnost na druhé třídě	57,17%	6,16%	69,00%	0%	72,94%
True positive	295	388	256	417	290
True negative	5 627	606	6 792	0	7 179
False positive	122	29	161	0	127
False negative	4 216	9 237	3 051	9 843	2 664
Precision	0,71	0,93	0,61	1,00	0,70
Recall	0,07	0,04	0,08	0,04	0,10
F1 measure	0,12	0,08	0,14	0,08	0,17
Matthews correlation c.	0,11	-0,007	0,13	*NaN*	0,19

Tabulka A.4: Výsledky klasifikace na neupraveném slovníku a datasetu klasifikace robotů od lidí s vymazáním URL pro `robots.txt` pro náhodný seed `12`

	Metoda klasifikace				
	Podle typu	Podle času	Statistické metody	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	72,55%	93,08%	64,20%	100%	70,17%
Úspěšnost na druhé třídě	56,96%	6,07%	69,45%	0%	73,04%
True positive	304	390	269	419	294
True negative	5 593	596	6 820	0	7 173
False positive	115	29	150	0	125
False negative	4 227	9 224	3 000	9 820	2 647
Precision	0,73	0,93	0,64	1,00	0,70
Recall	0,07	0,04	0,08	0,04	0,10
F1 measure	0,12	0,08	0,15	0,08	0,18
Matthews correlation c.	0,12	-0,007	0,14	*NaN*	0,19

Tabulka A.5: Výsledky klasifikace na neupraveném slovníku a datasetu klasifikace uživatelů fóra od zbytku pro náhodný seed `11`

	Metoda klasifikace				
	Podle typu	Podle času	Statistické metody	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	72,01%	97,16%	78,98%	100%	81,55%
Úspěšnost na druhé třídě	56,07%	6,77%	75,31%	0%	76,68%
True positive	1 343	1 812	1 473	1 865	1 521
True negative	4 864	587	6 533	0	6 652
False positive	522	53	392	0	344
False negative	3 8 11	8 088	2 142	8 675	2 023
Precision	0,72	0,97	0,79	1,00	0,82
Recall	0,26	0,18	0,41	0,18	0,43
F1 measure	0,38	0,31	0,54	0,30	0,56
Matthews correlation c.	0,21	0,06	0,44	*NaN*	0,47

Tabulka A.6: Výsledky klasifikace na neupraveném slovníku a datasetu klasifikace uživatelů fóra od zbytku pro náhodný seed `12`

	Metoda klasifikace				
	Podle typu	Podle času	Statistické metody	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	70,96%	96,73%	79,46%	100%	82,20%
Úspěšnost na druhé třídě	56,65%	6,77%	71,58%	0%	77,28%
True positive	1 344	1 832	1 505	1 894	1 557
True negative	4 884	584	6 172	0	6 663
False positive	550	62	389	0	337
False negative	3 738	8 038	2 450	8 622	1 959
Precision	0,71	0,97	0,79	1,00	0,82
Recall	0,26	0,19	0,38	0,18	0,44
F1 measure	0,39	0,31	0,51	0,30	0,58
Matthews correlation c.	0,21	0,06	0,40	*NaN*	0,48

Tabulka A.7: Výsledky klasifikace na upraveném slovníku a datasetu klasifikace robotů od lidí pro náhodný seed `11`

	Metoda klasifikace	
	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	100%	97,82%
Úspěšnost na druhé třídě	0%	98,39%
True positive	687	672
True negative	0	9 694
False positive	0	15
False negative	9 853	159
Precision	1,00	0,98
Recall	0,07	0,81
F1 measure	0,12	0,89
Matthews correlation c.	*NaN*	0,88

Tabulka A.8: Výsledky klasifikace na upraveném slovníku a datasetu klasifikace robotů od lidí pro náhodný seed `12`

	Metoda klasifikace	
	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	100%	98,26%
Úspěšnost na druhé třídě	0%	98,36%
True positive	691	679
True negative	0	9 664
False positive	0	12
False negative	9 825	161
Precision	1,00	0,98
Recall	0,07	0,81
F1 measure	0,12	0,89
Matthews correlation c.	*NaN*	0,88

Tabulka A.9: Výsledky klasifikace na upraveném slovníku a datasetu klasifikace robotů od lidí s vymazáním URL pro `robots.txt` pro náhodný seed `11`

	Metoda klasifikace	
	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	100%	71,46%
Úspěšnost na druhé třídě	0%	74,41%
True positive	417	298
True negative	0	7 324
False positive	0	119
False negative	9 843	2 519
Precision	1,00	0,71
Recall	0,04	0,11
F1 measure	0,08	0,18
Matthews correlation c.	*NaN*	0,20

Tabulka A.10: Výsledky klasifikace na upraveném slovníku a datasetu klasifikace robotů od lidí s vymazáním URL pro `robots.txt` pro náhodný seed `12`

	Metoda klasifikace	
	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	100%	75,18%
Úspěšnost na druhé třídě	0%	74,34%
True positive	419	315
True negative	0	7 300
False positive	0	104
False negative	9 820	2 520
Precision	1,00	0,75
Recall	0,04	0,11
F1 measure	0,08	0,19
Matthews correlation c.	*NaN*	0,22

Tabulka A.11: Výsledky klasifikace na upraveném slovníku a datasetu klasifikace uživatelů fóra od zbytku pro náhodný seed `11`

	Metoda klasifikace	
	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	100%	93,78%
Úspěšnost na druhé třídě	0%	94,25%
True positive	1 865	1 749
True negative	0	8 176
False positive	0	116
False negative	8 675	499
Precision	1,00	0,94
Recall	0,18	0,78
F1 measure	0,30	0,85
Matthews correlation c.	*NaN*	0,82

Tabulka A.12: Výsledky klasifikace na upraveném slovníku a datasetu klasifikace uživatelů fóra od zbytku pro náhodný seed `12`

	Metoda klasifikace	
	Markovův model	Vylepšený Markovův m.
Úspěšnost na první třídě	100%	94,19%
Úspěšnost na druhé třídě	0%	94,37%
True positive	1 894	1 784
True negative	0	8 137
False positive	0	110
False negative	8 622	485
Precision	1,00	0,94
Recall	0,18	0,79
F1 measure	0,31	0,86
Matthews correlation c.	*NaN*	0,83

B Uživatelský manuál

B.1 Použití

Aplikace je napsaná v jazyce Java, běží na *Java Vitual Machine (JVM)* a pro správné fungování potřebuje JVM minimálně ve verzi 1.8 (otestováno na verzi 1.8.0_91). Všechny součásti aplikace jsou spouštěné z příkazové řádky a neobsahují žádné grafické uživatelské rozhraní. Aplikace je rozdělená na tři samostatně spustitelné části, z nichž každá je popsaná dále v samostatné sekci.

B.1.1 `preprocessing.jar`

`jar` soubor `preprocessing.jar` je část aplikace zodpovědná za parsování logů a jejich rozdělování do tříd k testování. Toto není samotná klasifikace, ale příprava dat k trénování a testování klasifikátorů. Obě dvě funkce modulu `preprocessing.jar`, tedy parsování logů na návštěvy a jejich pozdější rozdělování, jsou oddělené a určují se parametrem při spuštění aplikace.

Parsování logů

Funkce parsování logů na návštěvy se zapíná přepínačem `-p` následovaným parametry pro vstupní soubor s logy a dvěma parametry, které označují výstupní soubor pro slovník a výstupní soubor pro návštěvy. Aplikace je spustitelný `jar` soubor, spuštění se provádí příkazem:

```
java -jar preprocessing.jar -p <vstupní soubor s logy> \
   <výstupní soubor slovníku> <výstupní soubor návštěv> \
   <'true' nebo 'false' parametr určující použití úprav slovníku>
```

Pokud máme logy v souboru `apache.log`, výsledný slovník nechceme nijak upravovat, chceme ho uložit do souboru `dict.txt` a návštěvy chceme uložit

do souboru `visits.txt`, spustíme aplikaci takto:

```
java -jar preprocessing.jar -p apache.log dict.txt visits.txt false
```

Přepínač pro úpravy slovníku používá při nastavení na `true` základní úpravu a to odstranění parametrů z URL při parsování.

Rozdělování návštěv podle koncového bodu

Druhá funkce `preprocessing.jar` je z vytvořených návštěv utvořit dvě skupiny, rozdělené podle toho, zda návštěva obsahuje požadavek na nějaký koncový bod. Tato funkce se zapíná přepínačem `-s`, jako vstupní soubory teď figurují soubory slovníku a seznamu návštěv. Výstupní soubory jsou seznamy návštěv pro každou třídu. Poslední dva parametry určují podle kterého koncového bodu se má rozdělovat a zda tento bod z návštěv smazat. Typické spuštění tedy vypadá:

```
java -jar preprocessing.jar -s <vstupní soubor slovníku> \
    <vstupní soubor všech návštěv> \
    <výstupní soubor návštěv, kde text nebyl nalezen> \
    <výstupní soubor návštěv, kde text byl nalezen> \
    <'true' nebo 'false' parametr určující mazání požadavku> \
    <text, podle kterého se bude třídit>
```

Tedy pokud máme slovník v souboru `dict.txt`, seznam všech návštěv v souboru `visits.txt`, chceme rozdělovat podle koncového bodu, identifikovaného textem `GET /robots.txt HTTP/1.1`, který nebudeme vymazávat, návštěvy robotů ukládat do souboru `robots.txt` a zbytek návštěv do souboru `clean.txt`, bude spuštění vypadat takto:

```
java -jar preprocessing.jar -s dict.txt visits.txt clean.txt \
   robots.txt false "GET /robots.txt HTTP/1.1"
```

B.1.2 `testing.jar`

Pro samotné testování klasifikátorů slouží část aplikace ve spustitelném jar souboru `testing.jar`. Má pět parametrů – cestu k souboru slovníku, dvě

cesty k souborům s návštěvami tříd, náhodný seed pro rozdělelní dat na trénovací a testovací a cestu k souboru s parametry klasifikátoru. Formáty zápisu klasifikátorů jsou rozepsány v sekci B.2. Příklad spuštění:

```
java -jar testing.jar dict.txt robots.txt clean.txt 10 markov.txt
```

Aplikace má pouze textové výpisy výsledků a vypisuje všechny měřené údaje pro daná data a klasifikátor. Výstup může vypadat následovně:

```
Advanced Markov classifier
TP: 680.0, TN: 9980.0, FP: 1.9999999999999982, FN: 26.99999999999989
Precision: 0.9970674486803519, Recall: 0.9618104667609619, \
   F measure: 0.9791216702663788
Matthews correlation coefficient: 0.9778525471723265
'Class 1' correct: 99.70674486803519 %
'Class 2' correct: 99.73018886779255 %
```

Hodnoty výsledků TP, TN, FP a FN se nevypočítávají celočíselně, proto je nutné jejich hodnoty zaokrouhlit v případě, že nevyjdou jako celé číslo.

B.1.3 `training.jar`

Ke trénování parametrů klasifikátorů slouží `jar` soubor `training.jar`. Má čtyři parametry – cestu k souboru slovníku, dvě cesty k souborům s návštěvami tříd a náhodný seed pro rozdělelní dat na trénovací a testovací. V zásadě se spouští stejně jako `testing.jar`, pouze zde není uvedena cesta k souboru s parametry klasifikátoru.

`training.jar` trénuje všechny typy klasifikátorů postupně za sebou. Vypisuje mezivýsledky pro každý testovaný klasifikátor, jeho hodnotu úspěšnosti na každé třídě a hodnotu skóre, což je Matthewsův korelační koeficient. Na konci běhu programu jsou vypsány nejlepší nalezené klasifikátory i s jejich parametry a hodnotami skóre.

Této části programu nebyl věnován takový důraz na snadnou spustitelnost, jelikož u trénování parametrů se předpokládá, že uživatel je programátor se znalostí daných trénovacích dat a algorimus si přizpůsobí podle sebe přímo otevřením a změnou zdrojového kódu v IDE a rekompilací.

B.2 Formáty souborů parametrů klasifikátorů

Všechny zde probírané formáty souborů slouží k uložení parametrů klasifikátorů do souboru. Soubor bude poté načten aplikací `testing.jar`. Žádná část aplikace parametry neukládá, tyto soubory je nutné vytvořit a naplnit daty ručně. Jak bylo již zmíněno výše, v této části nebyl kladen takový důraz na uživatelskou přívětivost.

Každý soubor začíná řádkou textu, která určuje typ klasifikátoru. Možné hodnoty jsou:

- `BYTYPE` – určuje klasifikátor podle typů stahovaných souborů,
- `TIMING` – určuje jednoduchý klasifikátor podle časů požadavků,
- `ADVTIMING` – klasifikátor podle časů s využitím statistických metod,
- `MARKOV` – základní Markovův klasifikátor a
- `ADVMARKOV` – vylepšený Markovův klasifikátor.

Za touto první řádkou následují parametry, které jsou pro každý klasifikátor jiné (viz dále). Příklady souborů s parametry jsou jako elektronická příloha ve složce `Examples`.

B.2.1 Klasifikátor podle typů souborů

Parametry tohoto klasifikátoru jsou hodnoty vah pro všechny třídy a typy souborů. Každá řádka má tvar:

```
<typ požadavku>, <přípona souboru>, <váha pro třídu 1>, \
  <váha pro třídu 2>
```

Celý zápis parametrů může vypadat například:

```
BYTYPE
GET,html,10.5,2.3
GET,pdf,1.5,8.5
GET,js,0.0,10.0
GET,css,1.5,7.3
```

B.2.2 Jednoduchý klasifikátor podle časů

V tomto typu klasifikátoru je potřeba určit váhy rozdílů průměrů a směrodatných odchylek pro každou třídu. Třídy jsou dvě a každá je reprezentována jedním řádkem, na kterém jsou obě hodnoty odděleny čárkou:

```
<váha pro rozdíl průměrů>,<váha pro rozdíl směrodatných odchylek>
```

Soubor může mít podobu například:

```
TIMING
0.5,0.325
0.01,0.98
```

B.2.3 Klasifikátor využívající statistických metod

Zde se určují dvě hodnoty, jednak hodnota prahu ke klasifikaci a pak hodnota true/false, která říká, zda se výsledky klasifikace mají otočit. V klasifikátoru se totiž vždy trénuje jedna třída, se kterou se ostatní porovnávají. Ve většině případů, ale máme „hledanou“ třídu na prvním místě a ostatní prvky až na druhém. Klasifikace funguje tak, že prvky které se rozdělením podobají trénovaným datům označuje jako první třídu a všechny ostatní jako druhou, což je právě přesně obráceně v případě, že třída s trénovacími daty je na druhém místě. Přepínač nastavený na `true` způsobí, že klasifikace bude obrácená (tj. prvky podobající se funkci označí jako druhou třídu a zbytek prvků jako první) a v tomto případě správná. Všechny příklady v celé této práci používají tuto obrácenou metodu klasifikace.

Příklad souboru s parametry klasifikátoru využívajícího statistických metod:

```
ADVTIMING
600.5,true
```

B.2.4 Markovův klasifikátor

Tento klasifikátor nemá žádné nastavitelné parametry, proto tento soubor označuje pouze typ klasifikátoru ve tvaru:

```
MARKOV
```

B.2.5 Vylepšený Markovův klasifikátor

Zde se pro každou třídu nastavují hodnoty počátečního skóre a hodnota hran vedoucích do uzlu NEZNÁMÝ. Jsou opět na jedné řádce pro každou třídu a odděleny čárkou:

```
<hodnota hrany vedoucí do uzlu NEZNÁMÝ>,<základní skóre>
```

Celý soubor může vypadat takto:

```
ADVMARKOV
0.0,0.01
0.5,0.0
```

B.3 Závislosti jednotlivých projektů

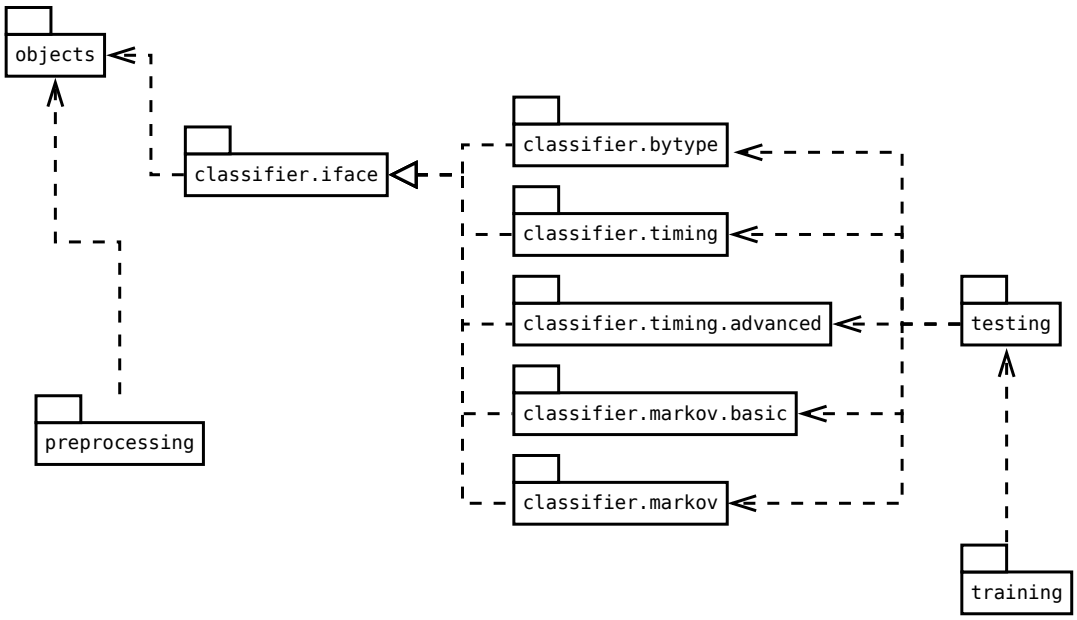
Celá aplikace se skládá z více `maven` projektů. Maven je nástroj pro správu projektů, jejich závislostí na jiných projektech a jejich sestavení (více se používá anglický tvar *build*). Nebudeme zde popisovat jeho celé fungování, ale zmíníme základní věci. Maven se řídí souborem `pom.xml`, ve kterém jsou definované závislosti projektu, tj. jiné projekty, které jsou k sestavení potřeba. Pro účely této aplikace dále `pom.xml` definuje výsledek sestavení (v této práci jsou pouze dva typy – `jar` soubor s projektem nebo spustitelný `jar` soubor s jednou ze tří spustitelných částí aplikace).

Všechny projekty s popisem jejich obsahu jsou vyjmenovány zde:

- `cz.zcu.masters.objects` – Projekt obsahující základní objekty používané v celé práci, například třídy pro ukládání návštěvy nebo slovníku.

- `cz.zcu.masters.classifier.iface` – Obsahuje jedno rozhraní, které musí implementovat všechny klasifikátory.
- `cz.zcu.masters.classifier.bytype` – Projekt pro klasifikátor podle typů stahovaných souborů.
- `cz.zcu.masters.classifier.timing` – Projekt pro jednoduchý klasifikátor podle časů.
- `cz.zcu.masters.classifier.timing.advanced` – Projekt pro klasifikátor podle časů s využitím statistických metod.
- `cz.zcu.masters.classifier.markov.basic` – Obsahuje projekt pro základní Markovův klasifikátor.
- `cz.zcu.masters.classifier.markov` – Projekt pro vylepšený Markovův klasifikátor.
- `cz.zcu.masters.preprocessing` – Projekt pro parsování logů a dělení návštěv. Popsán v sekci B.1.1.
- `cz.zcu.masters.testing` – Projekt pro testování klasifikátorů. Popsán v sekci B.1.2.
- `cz.zcu.masters.training` – Projekt pro trénování parametrů klasifikátorů. Popsán v sekci B.1.3.

Vztahy mezi projekty jsou znázorněny na UML digramu na obrázku B.1.



Obrázek B.1: UML diagram projektů

B.4 Build

Sestavení programu znamená sestavení všech jeho spustitelných částí a jejich závislostí, výstupem tedy nebude jeden soubor, ale tři. K sestavení je potřeba již zmíněný nástroj `maven`, a to minimálně ve verzi `3.0.5`.

Při buildu se musíme řídit závislostmi zobrazenými na UML diagramu na obrázku B.1. Musíme projekty kompilovat a sestavovat v takovém pořadí, aby závislosti projektu při jeho sestavování už byly vytvořené, tj. musíme postupovat proti směru šipek.

Pro všechny projekty musíme vytvořený soubor uložit do cache mavenu,

aby k němu další projekty měly přístup. Použijeme tedy dva přepínače `clean` a `install`. `clean` způsobí, že se projekt vymaže z cache a budeme mít zaručeno jeho nahrazení nejnovější verzí. `install` přidá do cache zkompilovaný a zabalený projekt. V případě spustitelných `jar` souborů potřebujeme, aby výsledkem sestavení byl také samotný `jar` s přibalenými závislostmi (tj. aby byl samostatně spustitelný). Použijeme přepínač `assembly:single`.

`maven` se spouští zkratkou `mvn` z příkazové řádky a ze složky, kde je umístěn `pom.xml`. Například sestavení projektu `cz.zcu.masters.classifier.iface` provedeme ze složky `Source/cz.zcu.masters.classifier.iface` na přiloženém DVD spuštěním příkazu:

```
mvn clean install
```

V případě spustitelného projektu jako například `cz.zcu.masters.testing` použijeme příkaz:

```
mvn clean install assembly:single
```

Spustitelný soubor se poté nachází ve složce `target` a má jméno:

```
<jméno projektu><verze>-jar-with-dependencies.jar
```

Tedy například:

```
cz.zcu.masters.testing-0.0.1-SNAPSHOT-jar-with-dependencies.jar
```

B.4.1 `build.sh`

Kvůli usnadnění kompilace a sestavení v Linuxovém prostředí je součástí programu také skript pro Linuxový nástroj `bash`, který sestaví všechny projekty ve správném pořadí a spustitelné `jar` soubory přejmenuje na kratší název a přesune do jedné složky.

Skript se spouští zavoláním „`bash build.sh`“ ze složky `Sources`. Zároveň skript počítá s tím, že byl zavolán z této složky a že všechny zdrojové soubory jsou v dalších podsložkách, tak jako na přiloženém DVD. Výsledkem skriptu je vytvoření spustitelných `jar` souborů `preprocessing.jar`, `testing.jar` a `training.jar` ve složce `Sources`. Logy mavenu jsou zapisovány do souboru `mvn.log`, který je nutné prozkoumat v případě, že k vytvoření souborů nedošlo.

Literatura

[1] SUMATHI, S., SIVANANDAM, S.N. *Introduction to Data Mining and its Applications*.
Springer, 2006
ISBN 3540343512, 9783540343516

[2] *What is the difference between the Web and the Internet?* [cit. 14.4.2017], dostupné na `https://www.w3.org/Help/#webinternet`

[3] *A quick introduction to HTML* [cit. 14.4.2017], dostupné na `https://www.w3.org/TR/html5/introduction.html#a-quick-introduction-to-html`

[4] *The HTML grammar definition* [cit. 14.4.2017], dostupné na `http://taligarsiel.com/Projects/howbrowserswork1.htm`

[5] *What language types are allowed in the HTML script tag?* [cit. 14.4.2017], dostupné na `http://stackoverflow.com/questions/2872037/what-language-types-are-allowed-in-the-html-script-tag`

[6] *URIs, URLs, and URNs: Clarifications and Recommendations 1.0* Joint W3C/IETF URI Planning Interest Group (21 September 2001), [cit. 14.4.2017],
dostupné na `https://www.w3.org/TR/uri-clarification/`

[7] BERNERS-LEE, Tim. *Historie protokolů, HTTP*. [cit. 19.4.2017], dostupné na `https://www.w3.org/History/19921103-hypertext/hypertext/WWW/Protocols/HTTP.html`

[8] *Protkol HTTP/1.1 pod lupou*. [cit. 19.4.2017],
dostupné na `https://www.root.cz/clanky/protokol-http-1-1-pod-lupou/`

[9] BRIGHT, Peter. *HTTP/2 finished, coming to browsers within weeks.* Ars Technica, [cit. 19.4.2017], dostupné na `https://arstechnica.com/information-technology/2015/02/http2-finished-coming-to-browsers-within-weeks/`

[10] *HTTP Overview, History, Versions and Standards.* September 20, 2005 [cit. 19.4.2017], dostupné na `http://www.tcpipguide.com/free/t_HTTPOverviewHistoryVersions-andStandards-3.htm`

[11] *Usage of HTTP/2 for websites.* [cit. 19.4.2017], dostupné na `https://w3techs.com/technologies/details/ce-http2/all/all`

[12] FIELDING, et al. *RFC 2616 - 10 Status Code Definitions.* [cit. 19.4.2017] dostupné na `https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`

[13] *Co je proxy server?.* [cit. 19.4.2017], dostupné na `http://www.sprava-site.eu/proxy-server/`

[14] RESCORLA, Eric. *HTTP Over TLS.* [cit. 19.4.2017], dostupné na `https://tools.ietf.org/html/rfc2818#section-2`

[15] *Information Geographies - The anonymous Internet.* [cit. 20.4.2017], dostupné na `http://geography.oii.ox.ac.uk/?page=tor`

[16] *Merriam-Webster Dictionary - Classification.* [cit. 21.4.2017], dostupné na `https://www.merriam-webster.com/dictionary/classification`

[17] *Transport Layer Security.* [cit. 21.4.2017], dostupné na `https://cs.wikipedia.org/wiki/Transport_Layer_Security`

[18] DORAN, Derek., GOKHALE, Swapna S. *Web robot detection techniques: overview and limitations.* [cit. 23.4.2017], dostupné na `https://www.researchgate.net/profile/Derek_Doran/publication/225108539_Web_robot_detection_techniques_Overview_-and_limitations/links/546dfcbf0cf29806ec2e658f.pdf`

[19] *About /robots.txt.* [cit. 23.4.2017] dostupné na `http://www.robotstxt.org/robotstxt.html`

[20] PARK, KyoungSoo., PAI, Vivek S., LEE, Kang-Won., CALO, Seraphin., *Securing Web Service by Automatic Robot Detection.* [cit. 23.4.2017], dostupné

na http://static.usenix.org/event/usenix06/tech/full_papers-/park/park_html/

[21] *Google's reCAPTCHA now invisible, separates bots from people without challenges.* [cit. 23.4.2017], dostupné na https://arstechnica.co.uk/gadgets/2017/03/google-recaptcha-invisible-no-challenge/

[22] STEINBERGER, Josef. *Information Retrieval: Lecture 1: Crawling.* [cit. 26.4.2017], dostupné na https://portal.zcu.cz/CoursewarePortlets2-/DownloadDokumentu?id=118568

[23] DORDA, Michal *Pearsonův χ^2 test dobré shody.* [cit. 27.4.2017], dostupné na http://homel.vsb.cz/ dor028/KMORII_5.pdf

[24] *P values.* [cit. 28.4.2017], dostupné na http://www.statsdirect.com/help/basics/p_values.htm

[25] *Statistical hypothesis testing, Clairvoyant card game.* [cit. 28.4.2017], dostupné na https://en.wikipedia.org/wiki/Statistical_-hypothesis_testing#Clairvoyant_card_game

[26] *Apache Server v2.4, Log Files, Combined Log Format.* [cit. 6.5.2017], dostupné na http://httpd.apache.org/docs/current/logs.html#combined

[27] POWERS, David M W *Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation.* Journal of Machine Learning Technologies. 2 (1): 37–63. (2011)