UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Boštjan Kovač

FILTER ZA OZNAČEVANJE NEPRIMERNIH BESEDIL NA SPLETNIH STRANEH

DIPLOMSKO DELO NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: doc. dr. Janez Demšar

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00415/2008
Datum: 15.10.2008

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **BOŠTJAN KOVAČ**

Naslov: **FILTER ZA OZNAČEVANJE NEPRIMERNIH BESEDIL NA SPLETNIH STRANEH**

FILTER FOR MARKING INAPPROPRIATE CONTENT ON WEB PAGES

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Sodobne tehnologije spletnega komuniciranja omogočajo, da lahko vsi, navadno tudi anonimni uporabniki, objavljajo sporočila na forumih, komentarje na novičarskih straneh in blogih in podobno. Med temi sporočili je veliko takšnih, ki so iz različnih razlogov nezaželena, denimo izrazito napadalna sporočila.

Sestavite dodatek za spletni brskalnik Firefox, s katerim bo lahko uporabnik označil dele strani, ki se mu zdijo neprimerni. Podatki o označenem besedilu (frekvence besed) naj se pošiljajo na strežnik, ki bo agregiral oznake različnih uporabnikov. Obenem naj dodatek s strežnika sprejema agregirane podatke in jih uporablja za klasifikacijo in samodejno označevanje neželenih sporočil. Pri tem uporabite enake postopke, kot se uporabljajo za razvrščanje neželene elektronske pošte.

Razviti dodatek naj bo integriran v Firefox, njegova uporaba pa preprosta, nezahtevna in nemoteča.

Mentor:

doc. dr. Janez Demšar

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Franc Solina

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a Boštjan Kovač,
z vpisno številko 63030088,
sem avtor/-ica diplomskega dela z naslovom:

Filter za označevanje neprimernih besedil na spletnih straneh.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  doc. dr. Janeza Demšarja
  in somentorstvom (naziv, ime in priimek)

  _____

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja/-ice: _____

Zahvala

Zahvaljujem se doc. dr. Janezu Demšarju za prevzeto mentorstvo ter nasvete pri izdelavi diplomske naloge, brez katerih bi težko naredil tako dober izdelek.

Zahvala gre tudi moji družini, ki mi je nudila podporo skozi vsa leta študija. Ne gre pa pozabiti še vseh sošolcev, s katerimi smo skozi skupaj trepetali pred izpiti in se veselili naših uspehov.

Kazalo vsebine

Kazalo slik

Povzetek

Danes so spletne strani preplavljene s komentarji obiskovalcev, ki so pogostokrat neprimerni, žaljivi ali nasilni. Vendar pa se je mogoče s primernim orodjem takim komentarjem izogniti ali pa biti na njih opozorjen. V tej diplomi smo se tega lotili z Bayesovim klasifikatorjem, ki je najbolj uporabljana metoda za zaznavanje neželene elektronske pošte. Želeli smo naučiti brskalnik, da bo uporabnika opozoril na neprimerno vsebino na spletni strani.

Da bi bilo orodje uporabniku prijazno, je bilo potrebno uporabiti različne tehnologije, ki smo jih združili v dodatku za spletni brskalnik Mozilla Firefox. V delu so opisane vse uporabljene tehnologije in povezave med njimi. Na začetku dela je tudi kratek opis filtra SpamBayes, na katerem temelji matematični del modela. V drugem delu pa je opisan postopek klasifikacije pri dodatku, ter vse podrobnosti in triki, uporabljeni za čimbolj učinkovito delovanje.

Končni izdelek potrjuje, da je tovrsten pristop zelo primeren za ugotavljanje neprimernosti, saj zelo učinkovito ocenjuje pogostost pojavljanja posameznih besed v dobrih in slabih komentarjih ter te ocene združi. Prav zaradi tega razviti dodatek ni končna verzija, pač pa ga bomo še naprej izboljševali. Še bolj pa bi bil vesel, če bom z svojim diplomskim delom navdušil še koga, ki bi prav tako želel sodelovati, saj so seveda vsi dodatki za Firefox odprto kodni, kjer moj dodatek ne bo izjema.

Ključne besede: Bayesov klasifikator, dodatki za spletni brskalnik Mozilla Firefox, podatkovna baza, spletne tehnologije

Summary

Modern web sites often contain visitors' comments which are inappropriate, offensive or even violent. It is however possible to avoid having to read such text with appropriate tools. We developed one based Bayesan techniques used in spam filters. The goal was to teach the web browser to alert the user about any inappropriate content on the web page.

To make the tool user friendly, we used a number of different technologies packed in a Firefox add-on. This work describes the used technologies and their relations. We start with a description of SpamBayes, the most efficient word-based spam filter at the moment, o which we based our work. In the second part we describe the extension itself, with all details and tricks to make it more efficient.

The resulting tool confirms the usefulness of the approach to filtering inappropriate messages. For this reason, we will further develop the presented add-on, and we hope that it will atract new users and, possibly, developers.

Keywords: Bayesan classifier, Mozilla Firefox web browser add-ons, database, web technologies

1. UVOD

V začetku so bile internetne strani namenjene le pridobivanju informacij. Obiskovalci so bili le pasivni opazovalci strani in niso mogli vplivati na njihovo vsebino. To se je začelo spreminjati z razmahom spletnih forumov. Na njih je vsak uporabnik izrazil svoje mnenje o neki temi in tako je nastal splošen pogovor. S spletnimi zapisi (blogi), pa se je pojavilo še komentiranje posameznih objav, ki se je nazadnje pojavilo tudi na spletnih straneh, ki objavljajo novice.

Do tu vse lepo in prav. Naravnost odlično je vedeti mnenje drugega človeka. In dokler smo na forumih, kjer si na primer mlade mamice izmenjujejo mnenja o vzgoji svojih otrok je to lahko zelo uporabno. So pa določene teme, kjer se nekateri ne znajo zadržati in žalijo vse povprek. Ena takih tem je recimo politika, kjer ima vsak svojo opcijo, le da nekateri ne priznavajo ostalih in pljuvajo čez njih. Tam pa naletimo na neprimerne komentarje, ki vsebujejo cel kup neprimernih besed. Obstajamo pa ljudje, ki se nam branje takih komentarjev zdi nesmiselno in nam pomeni izgubo časa, zato se jih izogibamo. Idealno bi bilo, če bi bili na take komentarje opozorjeni, še preden bi jih začeli brati.

V svoji diplomski nalogi sem se lotil prav tega. Želel sem naučiti brskalnik, da na strani označi komentarje z neprimerno vsebino. Naredil sem dodatek za spletni brskalnik Mozilla Firefox, s pomočjo katerega lahko uporabnik izbere del besedila in ga označi kot primernega ali neprimernega. Brskalnik naj te podatke zbira in naslednjič, ko na strani zazna podoben komentar, tega primerno označi.

2. UPORABLJENE TEHNIKE

Razviti dodatek temelji na Bayesovem klasifikatorju, obenem pa uporablja pester nabor različnih računalniških tehnologij.

2.1.Filter SpamBayes

Neželena elektronska pošta je danes velik problem, saj predstavlja okrog 70% vse poslane pošte. Ker pa je problem ločevanja dobrega od slabega sporočila zelo podoben problemu ugotavljanja primernosti komentarja na internetu, sem za osnovno razumevanje problematike vzel enega boljših filtrov za zaznavanje neželene pošte SpamBayes.

Filter SpamBayes deluje s pomočjo Bayesovega klasifikatorja. Začetek SpamBayesa sega v leto 2002, ko je Paul Graham napisal članek z naslovom »A plan for spam«.[1] V njem je opisal tehniko odkrivanja neželene pošte z Bayesovim klasifikatorjem. Kasneje so različni strokovnjaki njegovo tehniko še izboljšali in prišli do današnjega filtra.

Filter za začetek potrebuje dve zbirki sporočil. V eni zbirki so ham sporočila (tista, ki jih želimo prebrati), v drugi pa spam sporočila (tista, ki jih ne želimo prebrati). Ta sporočila razbije na posamezne besede in prešteje kolikorat se posamezna beseda pojavi v spam in kolikokrat v ham sporočilih. Nato izračuna

$$p(w) = b(w) / (b(w) + g(w))$$

Kjer velja b(w) = število spam sporočil, ki vsebujejo besedo w / skupno število spam sporočil, g(w) = skupno število ham sporočil, ki vsebuje besedo w / skupno število ham sporočil.

Vrednost p(w) lahko v grobem interpretiramo kot verjetnost, da je naključno sporočilo, ki vsebuje besedo w, spam sporočilo. Ta izračun bi bil v popolnem svetu, kjer bi imeli enako število ham in spam sporočil povsem zadosten. Ker pa se lahko zgodi, da je v poštnem predalu 10% spam in 90 % ham sporočil bo to imelo precejšen vpliv na verjetnost. V zgornjem izračunu tega vpliva ne upoštevamo.

Če verjetnost računamo po zgornjem principu, pa naletimo še na drug problem. Recimo, da imamo besedo, ki se pojavi samo v enem sporočilu in je to sporočilo spam. Izračunan p(w) bo 1.0, očitno pa ni povsem nujno, da bodo vsa bodoča sporočila, ki bodo vsebovala to besedo spam sporočila.

Na tem mestu pa nastopi tako imenovana stopnja zaupanja, ki jo poznamo iz Bayesove statistike. Iz izkušenj vemo, da se praktično vsaka beseda lahko pojavi v slabem ali dobrem sporočilu, zato združimo podatek p(w) z našim predznanjem. To stopnjo zaupanja izračunamo tako:

$$f(w) = ((s * x) + (m * p(w)) / (s + m)$$

Kjer je s moč, ki jo želimo dati našemu predznanju, x domnevna verjetnost, ki temelji na predznanju, da je neka beseda, ki je še nismo videli, spam, n pa število sporočil, ki smo jih prejeli in vsebujejo besedo w.

S tem dobimo oceno za eno besedo. Nato pa je potrebno te ocene združiti. SpamBayes za to uporablja inverzni hi-kvadrat.

$$H = C^{-1}\left(-2\ln\prod_{w} f(w),\ 2n\right)$$

Prvi parameter enačbe je zmnožek vrednosti f(w), drugi pa temu primerna stopnja svobode. Vendar pa je to le enostranski test. Ta izračun nam pove, kako »hammy« je naše sporočilo. Za učinkoviti izračun pa potrebujemo test še iz druge strani.

$$S = C^{-1}\left(-2\ln\prod_{w}\left(1 - f(w)\right),\ 2n\right)$$

Ta enačba je bolj občutljiva na vrednosti, ki se obračajo v smer, da je sporočilo spam. Ti dve vrednosti nato združimo in dobimo skupno oceno o tem, ali je neko sporočilo ham ali spam

$$I = \left(1 + H - S\right) / 2$$

Enačbo smo obrnili tako, da je rezultat vrednost na intervalu (0, 1). Ta izračun pomeni, da kadar sta vrednosti H in S obe blizu 0, je rezultat vrednost nekje na sredini. Takšno sporočilo nato lahko klasificiramo kot nedoločeno in se uporabnik sam odloči, ali je spam ali ham. [2, 3]

2.2. Dodatki za Firefox

Eden od razlogov, da je spletni brskalnik Mozilla Firefox tako popularen, je prav gotovo možnost izdelovanja raznoraznih dodatkov. Ko uporabniki brskajo po spletu, hitro naletijo na stvar, ki bi jo radi imeli za lažje brskanje. Velika verjetnost je, da je na podobno stvar pomislil že nekdo na svetu, ki po možnosti zna dobro programirati in tako je ustvaril dodatek, ga dodal v Mozillino bazo in s tem omogočil, da si ga lahko naložijo vsi, ki ga želijo uporabljati.

Tudi sam sem nad dodatki navdušen, zato me je zanimalo kako delujejo, kakšen je postopek izdelave, seveda pa sem tudi sam želel ustvariti dodatek, ki bi bil uporaben še za koga.

2.2.1. Programski jezik XUL za izdelavo uporabniškega vmesnika

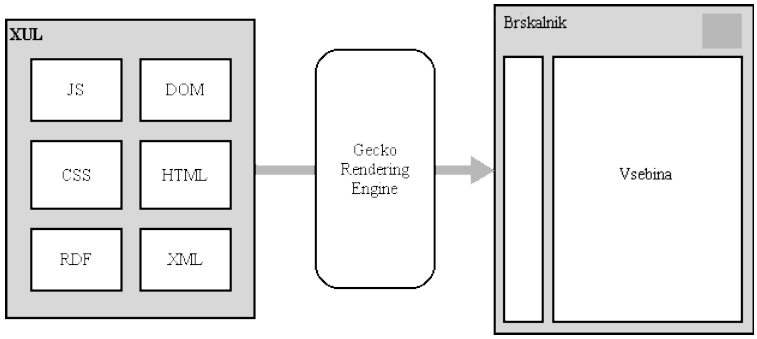
Tisti del dodatka, ki skrbi za interakcijo s spletnim brskalnikom je napisan v programskem jeziku XUL. Gre za označevalni jezik, ki so ga razvili pri Mozilli, namenjen pa je preprosti in hitri

izdelavi grafičnih uporabniških vmesnikov oziroma aplikacij. Za uporabo je izjemno enostaven, saj se ga vsi, ki poznajo HTML, lahko hitro naučijo.

2.2.2. Glavne lastnosti in prednosti XUL-a

Cilj XUL-a je izdelava aplikacij, ki niso odvisne od platforme, za razliko od DHTML-ja, ki je namenjen za razvijanje spletnih strani. Usmerjen je k izdelavi aplikacij kot so okna, gumbi, vnosna polja in se ne ukvarja s stranmi, povezavami…

Ustvarjen je na že obstoječih standardih in sicer temelji na W3C standardu XML 1.0. Prav tako vsebuje elemente drugih standardiziranih jezikov kot so HTML 4.0, CSS, DOM in Javascript (Slika 1).



Slika 1: Prikaz vseh tehnologij, ki jih vključuje XUL

Tako kot HTML je tudi XUL neodvisen od platforme, zato deluje na vseh operacijskih sistemih, kjer teče Mozilla. Glede na širok nabor platform, ki jih podpira Mozilla, je to ena od najbolj nepremagljivih lastnosti XUL-a. Uporabniški vmesniki vseh aplikacij Mozille (brskalnik, adresar, messenger) so napisani v XUL-u.

Velika pomanjkljivost večine spletnih aplikacij je tesno združevanje uporabniškega vmesnika in programskega dela aplikacije. To predstavlja precejšen problem predvsem v razvoju saj teh dveh stvari običajno ne pripravljajo isti ljudje. To je v XUL-u jasno ločeno. Razdeljeno je na tri dele in sicer:

- 'content', ki vsebuje XUL, XBL in Javascript
- 'skin', v katerem sta shranjena CSS in slike, ter
- 'locale', kjer se nahajajo datoteke DTD in besedilo v datoteki .properties, ki vsebuje različne napise za različne jezike.

Prav zaradi zadnje stvari, pa je programiranje v XUL-u enostavno kadar želimo svoj izdelek predstaviti po celem svetu in smo zanj naredili več različnih jezikov, saj je programska logika povsod enaka, le dodajamo prevode za različne jezike. [4]

2.2.3. Izdelava dodatka

Dodatki za Firefox, ki jih je možno povleči s spleta so pakirani v datoteko ZIP, s končnico XPI. Tipični dodatek ima sledečo zgradbo:

```
FireBayes.xpi:
              /components/*
              /components/cmdline.js
              /defaults/
              /defaults/preferences/*.js
              /plugins/*
              /chrome/icons/default/*
              /chrome/
              /chrome/content/
              /install.rdf
              /chrome.manifest
```

Pri izdelovanju dodatka se je priporočljivo držati te strukture datotek. Razlog za to je predvsem v tem, da ostali, ki želijo gledati podrobnosti dodatka, lahko vse elemente hitro najdejo. Vse navedene mape niso nujne, lahko pa dodajamo tudi svoje.

Dodatek začnemo izdelovati z dvema tekstovnima datotekama in sicer chrome.manifest in install.rdf. Vsebina datoteke install.rdf za dodatek FireBayes zgleda tako:

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:em="http://www.mozilla.org/2004/em-rdf#">
  <Description about="urn:mozilla:install-manifest">
    <em:id>bk@firebayes.net</em:id>
    <em:version>1.0</em:version>
    <em:type>2</em:type>
    <em:targetApplication>
      <Description>
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
        <em:minVersion>1.5</em:minVersion>
        <em:maxVersion>3.0.*</em:maxVersion>
      </Description>
    </em:targetApplication>
    <!-- Front End MetaData -->
    <em:name>FireBayes</em:name>
    <em:description>Dodatek, ki uporablja Bayesov klasifikator za oznacevanje
neprimernih komentarjev na spletnih straneh.</em:description>
    <em:creator>Bostjan Kovac</em:creator>
    <em:homepageURL>http://w3things.com/firebayes</em:homepageURL>
    <em:optionsURL>chrome://firebayes/content/firebayesset.xul</em:optionsURL>
  </Description>
</RDF>
```

V začetku je navedenih nekaj osnovnih podatkov, ki so potrebni za delovanje XUL-a. Nato sledi opis dodatka, kjer najprej napišemo ID dodatka. To je ime dodatka, ki mora biti unikatno. Običajno je napisan v formatu elektronskega naslova, lahko pa ga napišemo tudi kot GUID, vendar pa tak način ni popolnoma primeren, saj ga kasneje pri uporabi težje identificiramo. Sledi trenutna verzija dodatka in pa tip z oznako 2, ki pove, da gre za dodatek.

Naslednji del opisa vsebuje podatke o ciljni aplikaciji. Najprej je ID ciljne aplikacije, ki je v tem primeru Firefox, ki ima tako GUID kodo, kot vidimo v primeru. Sledita še minimalna in maksimalna verzija Firefox-a, v katerima deluje dodatek.

Zadnji del te datoteke so splošni podatki o dodatku in o izdelovalcu. Najprej je napisano ime in kratek opis dodatka. Sledi ime izdelovalca in domača stran. V primeru, da ima dodatek tudi možnost nastavitev, pa v tem delu navedemo pot in ime datoteke, ki predstavlja okno z nastavitvami. Tu se srečamo tudi z zapisom poti, ki jo uporablja Mozilla. Prvi del je ime dodatka (firebayes), nato sledi ime mape (content) in na koncu še ime dejanske datoteke, ki vsebuje programsko kodo okna z nastavitvami (firebayesset.xul).

Datoteka chrome.manifest vsebuje podatke o tem, kje se nahajajo posamezni deli kode. Za osnovno delovanje je dovolj le ena vrstica:

```
content     firebayes     chrome/content/
```

Najprej navedemo tip vsebine paketa (content), nato ime chrome paketa (firebayes) in pa lokacijo datotek. Z zgornjim stavkom smo tako povedali, da lahko za paket firebayes najdemo datoteke, ki predstavljajo njegovo vsebino (content), na lokaciji chrome/content, gledano relativno od lokacije datoteke chrome.manifest. Na ta način lahko definiramo še pakete za prevode v različne jezike (locale) ter za nastavitve glede izgleda (skin).

Če ti dve datoteki in strukturo map, ki smo jo omenili na vrhu, dodamo v skupno mapo, jo poimenujemo z imenom dodatka (v našem primeru firebayes) in dodamo v mapo, kjer so shranjeni dodatki (v MS Windows je to v mapi *C:\Documents and Settings\<uporabniško ime>\ Application Data\Mozilla\Firefox\Profiles\<profil>\Extensions)*, bo ob naslednjem zagonu Firefox že registriral dodatek. Ko pa dodatek še razvijamo, pa nam ni potrebno datotek vstavljati v mapo z dodatki, pač pa lahko hranimo vsebino dodatka nekje drugje, v mapi z dodatki pa izdelamo tekstovno datoteko, ki jo poimenujemo z ID-jem našega dodatka, vanjo pa napišemo pot, kjer se dejansko nahaja dodatek.

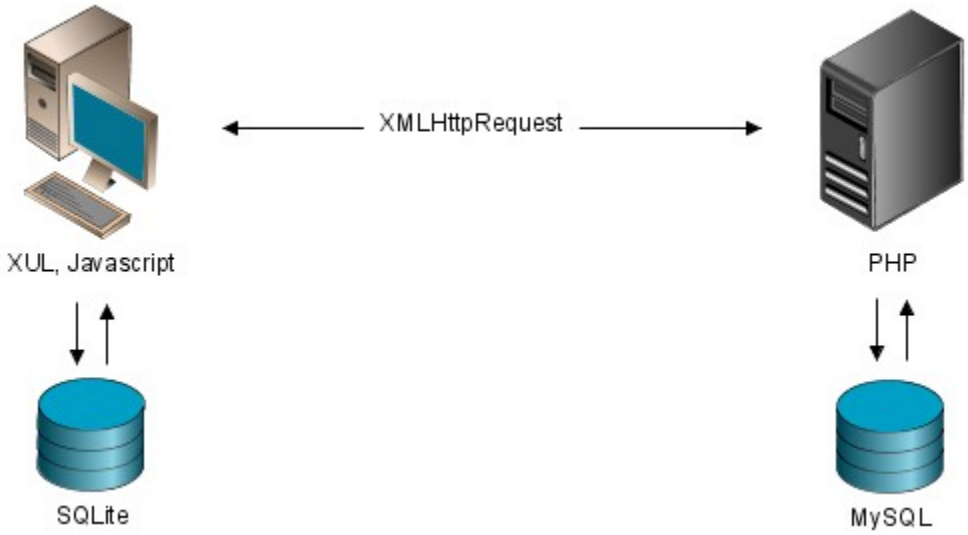
Seveda pa želimo dodatku dodati funkcionalnost. Najprej moramo v datoteko chrome.manifest dodati vrstico, s katero registriramo prevleko (overlay).

```
overlay chrome://browser/content/browser.xul chrome://firebayes/content/firebayes.xul
```

S tem Firefox-u povemo, naj spoji datoteko *firebayes.xul* z *browser.xul*, ko se *browser.xul* naloži. Poleg prevleke lahko spojimo še nastavitve za izgled (style), lahko pa tudi v celoti prepišemo izgled osnovnega okna (override). [5, 6]

2.3. Ostale uporabljene tehnologije

Kot smo že omenili pa XUL skrbi samo za predstavitveni del dodatka. Vsa logika, ki stoji za tem, pa je napisana v programskem jeziku Javascript. Podatki o besedah se lokalno hranijo v podatkovni bazi SQLite, povezava s strežnikom pa je narejena s pomočjo tehnologije AJAX. Na strežniku se podatki obdelajo s pomočjo programskega jezika PHP in se na koncu shranijo v podatkovno bazo tipa MySQL (Slika 2).



Slika 2: Prikaz povezav različnih tehnologij

2.3.1. JavaScript

JavaScript je skriptni programski jezik, katerega primarna uporaba je izdelava dinamičnih vsebin na spletnih straneh. Spada v skupino tako imenovanih tolmačev, ki napisane kode ne prevede v program z izvršilno datoteko, pač pa po vrsti razčlenjuje in izvršuje ukaze, ki smo jih napisali. Je objektno usmerjen programski jezik, na prvi pogled precej podoben programskima jezikoma C++ in Java. Uporabniki ga predvsem s slednjim zaradi podobnega imena dostikrat povezujejo, vendar pa ta dva jezika nimata nikakršne povezave.

Največkrat ga srečamo v spletnih brskalnikih, pri Netscapu pa so ga vgradili v svoj omrežni strežnik (Server-side Javascript). Lahko pa ga vključimo v različne vrste dokumentov, kot so HTML, XML in celo PDF. Prav zaradi svoje prilagodljivosti je postal zelo priljubljen, zato ga uporabljajo tudi pri Mozilli. [7]

2.3.2. SQLite

SQLite je odprto kodna podatkovna zbirka. Za razliko od večine drugih SQL podatkovnih zbirk, SQLite za svoje delovanje ne potrebuje ločenega procesa na strežniku saj bere in piše neposredno v običajno datoteko na disku. Tako je celotna zbirka s tabelami, indeksi, pogledi shranjena v eno datoteko na disku. SQLite ne smemo razumeti kot zamenjavo za večje baze, kot je Oracle, ampak bolj kot za menjavo tekstovne datoteke, v katero se pogostokrat shranjuje manjše zbirke podatkov. Je izjemno kompaktna baza, saj z vsemi vključenimi funkcijami zasede okrog 300kB. Če pa izklopimo posamezne funkcije, pa lahko to velikost še zmanjšamo.

Tako kot vsak odprto kodni projekt, je tudi SQLite pred vsakim novim izidom velikokrat testiran, zato je dobil ugled kot izjemno zanesljiva podatkovna baza. Njena priljubljenost v zadnjih letih

strmo narašča in tudi sami razvijalci ne sledijo več produktom kjer se pojavlja. Velikokrat jo lahko vidimo tudi v mobilnih telefonih, dlančnikih ter tudi v MP3 predvajalnikih. Od različice Firefox 3 naprej pa jo uporabljajo tudi pri Mozilli. Vanjo se shranjujejo piškotki, zgodovina obiskanih spletnih strani, zgodovina prenosov in tako dalje. [8]

2.3.3. AJAX

AJAX (Asynchronous Javascript + XML) je skupina povezanih tehnik za razvoj spletnih aplikacij, ki ustvarijo interaktivno spletno aplikacijo. Z AJAX-om lahko spletne aplikacije dobijo podatke iz spletnega strežnika asinhrono v ozadju, ne da bi bilo potrebno novo nalaganje obstoječe spletne strani. Uporaba AJAX-a prinaša porast interaktivnih vsebin na spletnih straneh. Podatki se pridobijo z uporabo objekta XMLHttpRequest, katerega začetki segajo že v leto 1999, ko je Microsoft v Internet Explorerju 5 izdelal XMLHTTP ActiveX kontrolo. Aprila 2006 pa je W3C izdelal prvi osnutek specifikacije za objekt z namenom izdelati uradni spletni standard.
Poleg vseh prednosti, ki jih AJAX prinaša, pa je naletel tudi na nekaj kritik. Ker se stran naloži dinamično, ne moremo slediti zgodovini odprtih strani, prav tako pa je težje neko stran dodati med priljubljene, saj po nekaj AJAX zahtevah na strani nimamo več vsebine, ki jo predstavlja povezava v naslovni vrstici.

2.3.4. PHP

PHP je odprto kodni skriptni programski jezik za izdelavo dinamičnih in interaktivnih spletnih strani. Je široko uporabljen, zastonj programski jezik, ki je še najbolj primeren za izdelavo spletnih aplikacij in ga je možno vstavljati neposredno v HTML. Teče na strežniku in kot vhod dobi PHP kodo, na izhodu pa izdela spletno stran v obliki HTML. Sintaksa PHP-ja je precej podobna Perl-u in C-ju. Pogosto je uporabljen skupaj s spletnim strežnikom Apache na različnih operacijskih sistemih. [9]

2.3.5. MySQL

MySQL je sistem za upravljanje relacijske baze z več kot 11 milijonov namestitev. Najbolj popularen je pri izdelavi spletnih strani in deluje kot podatkovna zbirka tako imenovane LAMP platforme (Linux, Apache, MySQL, PHP/Perl/Python). Njegova popularnost pri uporabi spletnih aplikacij je tesno povezana s priljubljenostjo PHP-ja.

Napisan je v programskem jeziku C in C++ in deluje na različnih platformah (Linux, Mac OS X, Windows…). Knjižnice za dostop do podatkovne baze MySQL so na voljo v vseh večjih programskih jezikih, preko tako imenovanih vmesnikov API.

Za administracijo podatkovnih baz se lahko uporablja že vgrajeno orodje z uporabo ukazne vrstice (ukaza mysql in mysqladmin). Na spletu je največkrat uporabljen phpMyAdmin, za uporabo na lokalnem računalniku pa je na voljo MySQL Administrator in MySQl Query Browser. [10]

3. RAZVITI DODATEK: FIREBAYES

V tem poglavju bo podrobneje opisano delovanje dodatka in njegov uporabniški vmesnik.

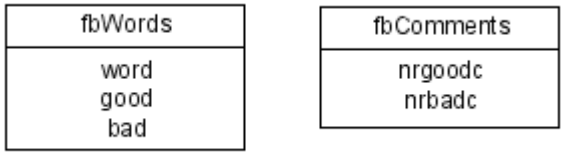
3.1.Podatkovne zbirke

Preden lahko začnemo uspešno klasificirati sporočila na spletnih straneh potrebujemo podatkovno zbirko s podatki o besedah, njihovo pogostostjo pojavljanja med dobrimi in slabimi komentarji ter tudi skupno število naučenih dobrih in slabih komentarjev. Kot smo omenili že prej imamo podatkovne zbirke na dveh lokacijah. Ena je na lokalnem računalniku, druga pa na strežniku.

3.1.1. Podatki o novih sporočilih na strani odjemalca

Za hranjenje podatkov na lokalnem računalniku se uporablja podatkovna baza SQLite, ki je v programskem jeziku Javascript izjemno enostavna za uporabo.

Za hranjenje besed, ki jih uporabnik označi kot dobre in slabe potrebujemo dve tabeli. Prvo smo poimenovali fbWords. Ta tabela hrani podatke o besedi (word), številu pojavitev te besede v dobrem (good) in slabem (bad) komentarju. Druga tabela, fbComments, pa vsebuje le eno vrstico v katero se zapisuje skupno število dobrih (nrgoodc) in slabih (nrbadc) komentarjev. Ti podatki so potrebni za kasnejši izračun verjetnosti (Slika 3).



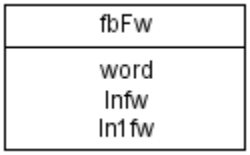
Slika 3: struktura tabel fbWords in fbComments

3.1.2. Podatki za klasifikacijo na strani odjemalca

Za delovanje samega klasifikatorja bi ti dve tabeli zadostovali. Dovolj bi bilo, da bi ti dve tabeli imeli shranjeni na računalniku, iz njiju pa bi v trenutku, ko bi se stran naložila, pobrali znanje o besedah. Vendar pa se je s testiranjem izkazalo, da je postopek precej dolgotrajen, saj je računalnik moral za vsako besedo dostopati do podatkovne baze, zanjo izračunati ln(f(w)) ter to potem seštevati z vrednostmi ostalih besed. To bi bilo predvsem moteče za uporabnika, saj bi ob vsaki odprti strani moral čakati nekaj sekund. Vendar pa sam izračun vrednosti ni tako časovno potraten, kot je počasno dostopanje do podatkovne baze za vsako besedo.

Zato bomo pri tem delu izvedli trik, ki nam bo znatno pospešil delovanje klasifikatorja. Namesto, da bi vrednost f(w) vsakič sproti izračunali, nam bo strežnik pripravil tabelo, kjer bodo vrednosti že izračunane.

Za uspešno klasificiranje potrebujemo logaritem produktov f(w) za vsako besedo iz besedila na strani. Velja pa, da je logaritem produktov enak vsoti logaritmov. V našem primeru je to ln PROD f(w) = SUM ln f(w). Zato bomo na strežniku pripravili že izračunane vrednosti ln(f(w)) za ugotavljanje ali so besede primerne in ln(1-f(w)), za ugotavljanje ali so besede neprimerne. Tako končna struktura tabele iz strežnika zgleda tako (Slika 4):



Slika 4: struktura tabele fbFw

Kot povedo že imena stolpcev, se na strežniku za vsako besedo izračuna ln(f(w)) in ln(1-f(w)). Ko bomo te vrednosti prenesli v lokalno podatkovno zbirko, nam ne bo več potrebno do baze dostopati za vsako besedo posebej, pač pa bo zadostoval en SQL stavek:

```
SELECT SUM(lnfw), SUM(ln1fw), COUNT(1) FROM fbFw WHERE word IN ("+readytext+")
```

kjer spremenljivka *readytext* predstavlja niz besed ločen z vejicami ('to','so','besede','v','nizu').

Vsoto SUM(lnfw) nato pomnožimo z -2 in izračunamo inverzni hi-kvadrat z 2n prostostnih stopenj, kjer je n število besed ki smo jih upoštevali pri izračunu oziroma vrednosti, ki jo v zgornji poizvedbi vrne COUNT(1). Tu pa se spet srečamo s počasnim delovanjem, saj ima računanje inverznega hi-kvadrata zahtevnost O(n), kjer je n spet število besed. Zato bomo to tabelo izračunali vnaprej.

Naj bo sfw vsota logaritmov in n število besed. Za klasifikacijo je potrebno izračunati C(-2*sfw, 2*n). To številko bomo dobili tako, da bomo pogledali v tabelo z izračunanimi vrednostmi. Vendar pa tabela ne obstaja za vsako možno kombinacijo, pač pa bomo uporabili nekaj zakonitosti, ki kompleksnost tabele občutno zmanjšajo. Naj bo *chi = -2*sfw, chi1 = -2*(1-sfw)* in *df = 2*n*. Pri tem velja

```
if (n < chi-30)
    H = 0;
else if (n >
chi+30)
    H = 1;
```

Ko n doseže vrednost 200, se številke ne spreminjajo več. Ker je vseeno ali je n enak 200 ali 250 bomo tabelo naredili samo do 200

```
if (n > 200)
        n =
200;
```

Naslednji korak je, da chi in chi1 zaokrožimo na 5, df pa na 10, ker je za naše potrebe to dovolj natančno

```
chi = 5*Math.round(chi/5);
chi1 = 5*Math.round(chi1/5);
df = 10*Math.round(df/10);
```

To številko pa potem lahko najdemo v tabeli chisqprob, ki si jo pripravimo vnaprej.

```
H=chisqprob[df][chi];
S=chisqprob[df][chi1];
```

Tu H pomeni verjetnost, da je neko besedilo dobro, S pa verjetnost, da je slabo. Končni rezultat nam da skupno verjetnost

```
I=(1+H-S)/2;
```

Vrednosti H in S sta na intervalu od 0 do 1. Zato bo besedilo, o katerem imamo veliko dokazov da je dobro (vrednost blizu 1), kot tudi veliko dokazov da je slabo (prav tako vrednost blizu 1) imelo vrednost približno 0.5.

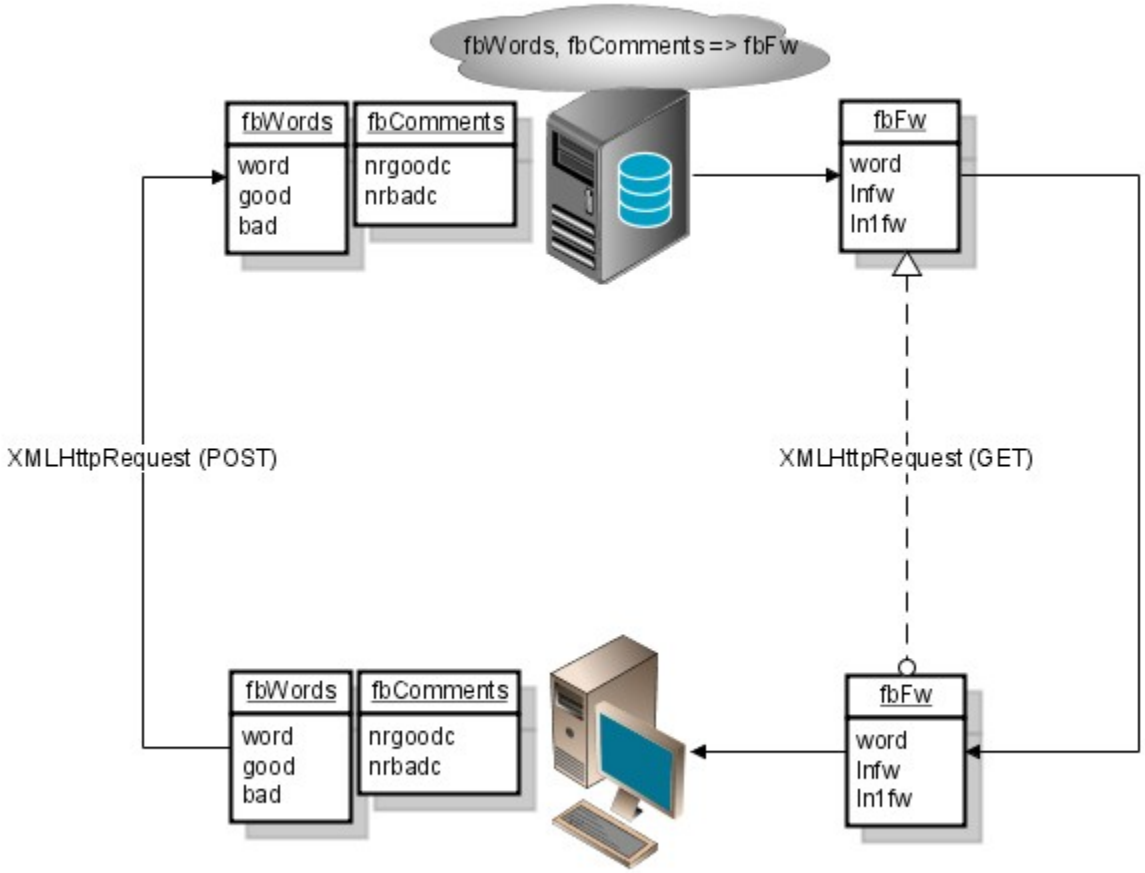
3.1.3. Podatki na strežniku

Struktura tabel na strežniku je enaka tisti na lokalnem računalniku. Ko iz lokalnega računalnika pošljemo podatke o besedah, katere smo označili, se na strežniku prav tako shranijo v tabelo fbWords, skupno število dobrih in slabih komentarjev pa v tabelo fbComments. Ob vsaki posodobitvi se nato izračunajo nove vrednosti ln(f(w)) in ln(1-f(w)) ter shranijo v tabelo fbFw. Ta tabela pripravljena čaka na zahtevo uporabnika, ki jo bi želel prenesti na lokalni računalnik.

3.1.4. Sinhronizacija podatkovnih zbirk

Zaradi vsega naštetega je podatke o pogostosti pojavljanja posameznih besed potrebno spraviti do strežnika. Težko je verjeti, da bi vsak uporabnik zase označeval tekst kot dober in slab, saj je sam postopek precej dolgotrajen, za uspešno označevanje pa mora biti podatkovna zbirka precej obsežna in ni dovolj le nekaj besed.

Za pošiljanje tabel fbWords in fbComments na strežnik uporabljamo prenos s pomočjo XMLHttpRequest. Ta sistem se je uveljavil pri programiranju AJAX, saj omogoča pošiljanje, obdelavo ter povratno informacijo iz strežnika brez ponovnega nalaganja spletne strani. Običajno je sicer pri AJAX-u rezultat kasneje izpisan na zaslonu, v tem primeru pa je važno le, da se podatki iz tabele prenesejo na strežnik ter tam shranijo (Slika 5).



Slika 5: Prikaz komunikacije med lokalnim računalnikom in strežnikom

3.1.5. Pošiljanje vrednosti iz lokalne tabele

Za kreiranje objekta je v Javascriptu že pripravljen stavek:

```
var fbupload = new XMLHttpRequest();
```

S tem ustvarimo objekt, ki bo komuniciral z oddaljenim strežnikom. Naslednja naloga je sestavljanje stavka, ki ga bomo poslali z zahtevo. Ker moramo na strani strežnika znati prebrati sporočilo, je potrebno ustvariti format zapisa, ki bo lahko berljiv. Tu bomo uporabljali:
word,good,bad|word,good,bad|...
Tako koda za kreiranje stavka izgleda takole:
Da se izognemo znaku "|" na koncu stavka, izvedemo še stavek

```
while (statementUP.executeStep())
{
    uptext += statementUP.getString(0) + "," + statementUP.getString(1) + "," +
statementUP.getString(2) + "|";
}
```

ki nam ob koncu odreže zadnji znak.
Poleg podatkov o besedah pa moramo poslati tudi število dobrih in slabih komentarjev, ki jih je označil uporabnik. Stavek za sestavljanje je podoben, s tem da tu tudi že upoštevamo, da bomo zahtevo morali združiti s prejšnjim stavkom, zato dodamo znake &:

```
while (statementUPCOUNT.executeStep())
{

upcount="&nrgoodc="+statementUPCOUNT.g
etString(0)+"&nrbadc="+statementUPCOUN
T.getString(1);
}
```

Celoten stavek sedaj združimo v zahtevi open, ki že pripravi vse potrebno za pošiljanje podatkov:

```
fbupload.open("POST","http://w3things.com/firebayes/fbupload.php?txt="+uptext+upcount,
false);
```

Ko je pošiljanje uspešno izvedeno, pa na strani uporabnika še pobrišemo tabelo z besedami in število dobrih in slabih komentarjev postavimo na 0. To je pomembno zato, da ob naslednjem pošiljanju ne bomo ponovno pošiljali istih podatkov.
Za to ustvarimo dva stavka SQL. Prvi pobriše vsebino tabele fbWords, drugi pa postavi na nič vrednosti v tabeli fbComments.

```
connDB.executeSimpleSQL("DELETE FROM fbWords");
connDB.executeSimpleSQL("UPDATE fbComments SET nrgoodc=0, nrbadc=0 WHERE id=1");
```

Primer celotnega ukaza za dve besedi, ki predstavljata dober komentar je videti tako:

```
http://www.w3things.com/firebayes/fbupload.php?text=hello,1,0|
world,1,0&nrgoodc=1&nrbadc=0
```

3.1.6. Sprejemanje sporočila na strani strežnika

Nadaljnji potek pošiljanja podatkov se odvija na strani strežnika. Na strežniku uporabljamo programski jezik PHP in podatkovno bazo MySQL, ki je precej bolj zmogljiva od SQLite, ki jo uporabljamo pri lokalni zbirki. Poleg tega, da na strežniku v bazo zapišemo podatke o besedah, pa obenem tudi že izračunamo vrednost ln(f(w)) in ln(1-f(w)).
Vendar pojdimo lepo po vrsti. Najprej moramo iz prejete zahteve izluščiti posamezne podatke. V tri lokalne spremenljivke shranimo vrednosti posameznih atributov:

```
$fbtxt = $_GET['txt'];
$nrgoodc = $_GET['nrgoodc'];
$nrbadc = $_GET['nrbadc'];
```

Že prej smo omenili, da se celoten tekst iz lokalne podatkovne baze pretvori v format, katerega bomo na strani strežnika lahko prebrali. V programskem jeziku PHP imamo za razbijanje besedila uporabno funkcijo *explode*, ki za vhod potrebuje celotno besedilo ter znak, ki loči besedilo na posamezne dele.
V našem primeru bomo ta stavek uporabili dvakrat – prvič bomo z njim razbili posamezno vrstico in bomo kot ločilo uporabili znak '|'

```
$lines = explode("|",$fbtxt);
```

Tako dobimo vrsto, ki je v formatu 'word,good,bad'. Naslednji korak je, da posamezen zapis še bolj razbijemo v zanki, le da tokrat za ločilo uporabimo znak ','

```
for($i=0; $i < sizeof($lines); $i++)
{
    $line = explode(",",$lines[$i]);
    $updateline = "INSERT INTO fbWords (word, good, bad) VALUES ('$line[0]',
'$line[1]', '$line[2]') ON DUPLICATE KEY UPDATE good=good+'$line[1]',
bad=bad+'$line[2]'";
    mysql_query($updateline, $conn);
}
```

Kot lahko vidimo, dobljene vrednosti v tej zanki tudi že vpisujemo v podatkovno bazo. SQL stavek je narejen tako, da vstavi besedo, število pojavitev v dobrem in število pojavitev v slabem komentarju. V primeru, da določena beseda že obstaja, pa trenutno vrednost v podatkovni bazi povečamo za vrednost, ki jo dobimo z zahtevo. Pri tem uporabimo stavek ON DUPLICATE KEY UPDATE, ki omogoči, da za vsako besedo samo enkrat dostopamo do podatkovne baze.

3.1.7. Izračun vrednosti ln(f(w)) in ln(1-f(w))

Kot smo že omenili, se izračun vrednosti ln(f(w)) in ln(1-f(w)) zaradi hitrejšega izvajanja klasifikatorja izvede na strežniku. Lahko bi na novo izračunali le vrednosti pri besedah, ki smo jih na novo naložili oziroma tistih, katerim vrednost smo spremenili. Vendar pa se zaradi spremenjenih vrednosti skupnega števila dobrih in slabih komentarjev spremeni f(w) tudi pri ostalih besedah. Zato ob vsakem nalaganju na novo preračunamo celotno tabelo. Če bi to počeli na lokalni podatkovni bazi, bi lahko postopek trajal precej časa in upočasnil delovanje nalaganja. Ker pa se vsa zadeva izvaja na strežniku pa za uporabnika ni moteče, saj mu računanje ne obremenjuje lokalnega računalnika.

Najprej je potrebno iz tabele fbComments vzeti novo število dobrih in slabih komentarjev. Izvedemo stavek SQL, ki nam ti dve vrednosti prebere iz podatkovne baze ter shranimo vrednosti v dve spremenljivki ngc in nbc:

```
$result_comm = mysql_query("SELECT nrgoodc, nrbadc FROM fbComments WHERE id=1", $conn);
$row_c=mysql_fetch_array($result_comm);
$ngc=$row_c['nrgoodc'];
$nbc=$row_c['nrbadc'];
```

V naslednjem koraku preberemo vse vrednosti iz tabele fbWords

```
$result_words = mysql_query("SELECT word, good, bad FROM fbWords", $conn);
```

Rezultat poizvedbe SQL damo v zanko in ponavljamo, dokler ne izračunamo vrednosti za vse besede. Izračunane vrednosti zapisujemo v tabelo fbFw.

```
while ($row_w=mysql_fetch_array($result_words))
{
    $w=$row_w['word'];
    $tig=$row_w['good'];
    $tib=$row_w['bad'];
    $m = $tig+$tib;
    $g_w = $tig/$ngc;
    $b_w = $tib/$nbc;
    $p_w = $b_w / ($b_w + $g_w);
    $f_w = (($s*$x) + ($m * $p_w)) / ($s + $m);
    $lnf_w = log($f_w);
    $ln1f_w = log(1-$f_w);
    $update_words = "INSERT INTO fbFw (word, lnfw, ln1fw) VALUES ('$w', '$lnf_w',
'$ln1f_w') ON DUPLICATE KEY UPDATE lnfw='$lnf_w', ln1fw = '$ln1f_w'";
    mysql_query($update_words, $conn);
}
```

Sam izračun ni zapleten, če razumemo ozadje, ki smo ga predstavili v prejšnjih poglavjih. Preberemo vrednosti iz tabele, nato pa izračunamo g(w), b(w) iz rezultatov lahko izračunamo p(w) ter na koncu še f(w).

Vrednosti še vpišemo v bazo s podobnim stavkom SQL kot smo to storili pri vpisovanju novih besed in zadeva je končana.

3.1.8. Prenašanje podatkov iz strežnika

Ko želimo iz strežnika prenesti posodobljene podatke prav tako pošljemo zahtevo s pomočjo XMLHttpRequest, le da tokrat argument POST zamenjamo z GET.

```
var fbdownload = new XMLHttpRequest();
fbdownload.open("GET","http://www.w3things.com/firebayes/fbdownload.php",false);
```

Ob prejeti zahtevi s poizvedbo SQL preberemo vse vrednosti iz tabele fbFw. Tu pa bomo napravili trik, ki bo pripomogel k občutno hitrejšemu vstavljanju podatkov iz podatkovne baze na lokalni verziji. Lahko bi namreč celotne podatke oblikovali v podobno obliko kot smo jo imeli pri nalaganju. Vendar to pomeni, da bi bilo potrebno na lokalnem računalniku ta tekst obdelati ter vsak zapis vpisovati v bazo, kar pa bi bilo zelo zamudno za računalnik. Zato bomo celoten stavek SQL, ki se bo izvedel na lokalnem računalniku, kreirali že na strežniku.

```
$getlnfw="SELECT word, lnfw, ln1fw FROM fbFw";
$resultlnfw=mysql_query($getlnfw, $conn);
$sendtext="BEGIN;";
while($row_lnfw=mysql_fetch_array($resultlnfw))
{
    $sendtext.="INSERT INTO fbFw VALUES ('".$row_lnfw['word']."',".
$row_lnfw['lnfw'].",".$row_lnfw['ln1fw'].");";
}
$sendtext.="COMMIT;";
echo $sendtext;
```

S tem smo se izognili zamudnemu vstavljanju vrednosti v tabelo. Dejstvo namreč je, da je pri podatkovnih bazah zamudno dostopanje do baze. In če izvedemo nekaj tisoč vstavljanj, pomeni, da smo prav tolikokrat morali dostopati do baze, kar pa lahko traja nekaj sekund. Bolje je, da celoten stavek uokvirimo s stavkoma BEGIN in COMMIT. S tem smo si zagotovili, da se podatki ne vpisujejo sproti, pač pa se vse skupaj prenese v bazo ob stavku COMMIT, kar pomeni, da gre le za eno poizvedbo. Najprej pa moramo na strani uporabnika sporočilo sprejeti

```
txt_fw=fbdownload.responseText;
```

Kot smo že omenili, se lokalno uporablja podatkovna zbirka SQLite, ki pa ima v tem koraku precejšno pomanjkljivost, saj ne pozna stavka ON DUPLICATE KEY UPDATE. Zadevo bi lahko rešili s stavkom REPLACE, ki pa je v SQLite-u zelo počasen, saj za vsak zapis, ki ga že imamo v podatkovni bazi in zanj želimo narediti samo posodobitev, staro vrednost izbriše, novo pa doda na koncu tabele. Ker trenutnega stanja tabele ne bomo več potrebovali, je najlažje, da preden začnemo vpisovati vrednosti, enostavno pobrišemo vrednosti v celotni tabeli.

```
connDB.executeSimpleSQL("DELETE FROM fbFw");
```

Sedaj imamo tabelo fbFw izpraznjeno. Vse, kar potrebujemo sedaj, je izvedba stavka, ki smo ga dobili iz strežnika

```
connDB.executeSimpleSQL(txt_fw);
```

3.2. Klasificiranje sporočil

Zaključni del dodatka in hkrati tisti del, ki dejansko daje rezultat na strani je klasifikacija besedila in obarvanje tega besedila, v primeru da gre za neprimeren komentar. Najprej moramo dodati poslušalca, ki se aktivira ob zagonu programa.

```
window.addEventListener("load", FireBayes.init, false);
```

Kot lahko vidimo, v tem dogodku kličemo funkcijo init(), ki še ne preverja teksta, pač pa definira drugega poslušalca, ki preverja vsebino brskalnika in proži dogodek ob naložitvi strani

```
init: function()
{
    var appcontent = document.getElementById("appcontent");
    if(appcontent)
          appcontent.addEventListener("DOMContentLoaded", FireBayes.gettext, true);
}
```

3.2.1. Pobiranje besedila iz spletne strani

Ko se stran naloži v brskalnik, je iz nje potrebno povleči besedilo. Vendar pa je besedilo enostavno pobrati iz strani, če poznamo njeno strukturo. Bolj kompliciran pa je to posplošiti in iz vseh strani pobrati le uporaben tekst.

Problem je, da ima vsaka stran drugačno strukturo in ima besedilo shranjeno v različnih delih strani. Ena od možnosti so odstavki oziroma značka <p>, ki pomeni *paragraph*.

```
Windows.content.document.getElementsByTagName('p');
```

Vendar pa nam nadaljnje raziskovanje pokaže, da najbolj razširjeni modul za forum, phpBB z značko <p> ne bo ponudil nobenega teksta. Forum je v bistvu sestavljen iz tabel, telo objave na forumu pa je tam, kjer je class='postbody'. Tam tekst lahko dobimo z:

```
Windows.content.getElementsByClassName('postbody');
```

Kljub vsemu si na tem mestu oglejmo primer klasifikacije po prvem principu. Ko torej poberemo iz strani vse besedilo v oznakah <p> dobimo vrsto, kjer vsak element vrste pomeni besedilo iz enega odstavka. Izdelamo zanko, ki se ponovi tolikokrat, kot imamo odstavkov na strani

```
for (p=0; p<FireBayes.pagetext.length; p++)
```

Besedilo nato pripravimo za poizvedbo SQL. Za to uporabimo dve funkciji, ki smo jih prej pripravili – *tokenize* in *preparetext*.

```
FireBayes.paratext=FireBayes.tokenize(FireBayes.pagetext[p].innerHTML);
readytext=FireBayes.preparetext(FireBayes.paratext);
```

Prva razbije besedilo na posamezne besede, druga pa te besede spravi v niz, ki je primeren za vstavljanje v stavek SQL. Ko imamo niz pripravljen, izvedemo stavek SQL, s katerim dobimo vsoti logaritmov f(w)

Kot lahko vidimo, te vrednosti tudi že shranimo v spremenljivke. Klasifikacija naprej poteka

```
var statementFW = dbConn.createStatement("SELECT SUM(fw), SUM(fw2), COUNT(1) FROM
fbFw WHERE word IN ("+readytext+")");
while (statementFW.executeStep())
{
    sumfw = statementFW.getDouble(0);
    sum1fw = statementFW.getDouble(1);
    n = statementFW.getInt32(2);
}
```

```
chi = -2*sumfw;
chi1 = -2*sum1fw;
df = 2*n;
if(n<chi-30)
    H=0;
else if (n>chi+30)
    H=1;
else
{
    if(df>200)
           df=200;
    chi = 5*Math.round(chi/5);
    chi1 = 5*Math.round(chi1/5);
    df = 10*Math.round(df/10);
    H=chisqprob[df][chi];
    S=chisqprob[df][chi1];
    I=(1+H-S)/2;
}
if(I>firebayesPref.getCharPref("extensions.firebayes.badrank"))
    FireBayes.color(p);
```

Dodana sta le zadnja dva stavka. S prvim preverimo, če je ocena besedila presegla vrednost, ki jo je uporabnik nastavil kot mejo. Če jo je presegla, kličemo funkcijo *color*, ki kot argument dobi številko odstavka, ki ga ta funkcija obarva.

```
color: function(par_nr)
{
    window.content.document.getElementsByTagName("p")
[par_nr].style.backgroundColor=firebayesPref.getCharP
ref("extensions.firebayes.color");
}
```

Kot lahko vidimo, je tudi tukaj vrednost za barvo prenesena iz nastavitev uporabnika. Tako si lahko uporabnik sam izbere barvo, s katero želi imeti obarvano ozadje neprimernega besedila.

3.3. Učenje

Učenje komentarja je razdeljeno na dva dela in sicer se lahko odločimo, da komentar označimo kot dober ali kot slab. V obeh primerih je postopek enak: označimo tekst na spletni strani in pritisnemo ustrezen gumb. Pomembno pri vsem je, da se tekst tudi razbije na posamezne besede in da se iz njega izloči posebne znake, kot so na primer ločila, ki lahko otežijo nadaljnje delo.

3.3.1. Komentar je dober

Najprej moramo iz spletne strani pobrati tekst, ki ga želimo dodati v podatkovno zbirko. To seveda ni preveč kompliciran, saj Javascript za to vsebuje že pripravljene stavke

```
var goodtext=window.content.document.getSelection();
```

Besedilo se potem obdela, zato da v podatkovno zbirko vpisujemo le besede, brez ločil in drugih znakov, ki jih ne želimo imeti shranjenih. Najprej celoten tekst pomanjšamo v male črke.

```
tcomment = tcomment.toLowerCase();
```

Nato besedilo razbijemo na posamezne besede. Za delitev uporabimo regularni izraz, s katerim računalniku povemo, da naj besede deli po vsem, kar ni kot črka ali številka.

```
tokens = tcomment.split(/[^a-zA-Z0-9čšž]+/);
```

Preden se lotimo vpisovanja besed v podatkovno bazo, pa v tabeli fbComments povečamo vrednost v stolpcu nrgoodc za ena, saj smo dodali en nov, dober komentar.

```
connDB.executeSimpleSQL("UPDATE fbComments SET nrgoodc = nrgoodc+1 WHERE id = 1");
```

Naslednja stvar je, da moramo čez vse besede v vrsti in za vsako vpisati vrednosti v bazo. Če v bazi že obstaja, potem moramo povečati vrednost good za ena, če pa beseda v bazi še ne obstaja, pa je potrebno dodati celotno novo vrstico z vrednostmi word, good, bad, kjer je očitno, da je vrednost good=1 in vrednost bad=0.
Celoten postopek je v SQLite-u potrebno izpeljati na malce daljši način, saj ta baza ne podpira nekaterih stavkov, ki jih imajo zmogljivejše podatkovne baze. Zato moramo za vsako besedo narediti

```
SELECT word, good, bad FROM fbWords WHERE word = ?1
```

kjer je ?1 argument, ki predstavlja posamezno besedo iz vrste. Če nam ta SQL stavek vrne prazno tabelo, potem te besede še ni v podatkovni bazi in jo moramo dodati

```
INSERT INTO fbWords (word,good,bad) VALUES (?1,1,0)
```

kjer je ?1 ponovno beseda iz vrste, 1 in 0 pa sta vrednosti good in bad za novo besedo v podatkovni zbirki. V primeru da beseda že obstaja pa je potrebno le povečati trenutno vrednost v podatkovni zbirki

```
UPDATE fbWords SET good = good+1 WHERE word = ?1
```

3.3.2. Komentar je slab

V primeru, da želimo komentar označiti kot slab komentar, je postopek popolnoma enak, le da spreminjamo druge vrednosti v tabelah. V tabeli fbComments povečamo vrednost nrbadc

```
UPDATE fbComments SET nrbadc = nrbadc+1 WHERE id = 1
```

Nato pa v primeru, da beseda še ne obstaja dodamo obratne vrednosti za good in bad in sicer good je sedaj 0, bad pa 1

```
INSERT INTO fbWords (word,good,bad) VALUES (?1,0,1)
```

Če pa je beseda že v podatkovni zbirki, pa povečamo vrednost bad

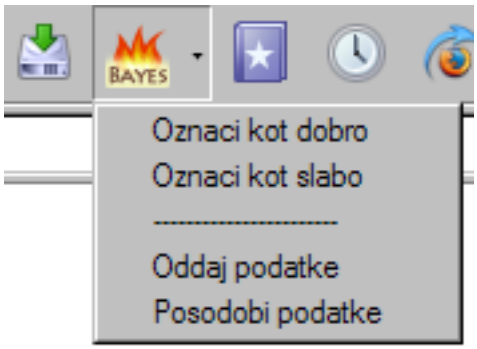
```
UPDATE fbWords SET bad = bad+1 WHERE word = ?1
```

3.4. Uporabniški vmesnik

Dodatek ima v orodni vrstici tipko z menijem, ki omogoča enostavno upravljanje z dodatkom (Slika 6).

V meniju se nahajajo naslednje možnosti:

- Označi kot dobro (z njim izbrano besedilo označimo kot dobro)
- Označi kot slabo (z njim izbrano besedilo označimo kot slabo)
- Oddaj podatke (z njim oddamo na strežnik zbirko besed iz označenih besedil)
- Posodobi podatke (z njim pridobimo iz strežnika posodobljene podatke)



Slika 6: Gumb in meni v orodni vrstici Firefox-a

Tipka in meni sta določena v programskem jeziku XUL. Sama koda je kratka in enostavna. Na začetku definiramo, da gre za prevleko okna, torej za *overlay*.

```
<overlay id="firebayes"
xmlns="http://www.mozil
la.org/keymaster/gateke
eper/there.is.only.xul"
>
```

S tem stavkom Firefox-u povemo, da bomo naredili prevleko čez osnovno okno brskalnika. V naslednjem koraku izdelamo gumb, ki bo dodan v orodno vrstico, hkrati pa naredimo še menijske postavke, ki se odprejo ob kliku na ta gumb.

```
<toolbarpalette id="BrowserToolbarPalette">
    <toolbarbutton id="firebayes-button"
           type="menu"
           label="FireBayes"
           image="chrome://firebayes/content/fb_icon_on.png"
           tooltiptext="FireBayes">
           <menupopup>
                  <menuitem label="&firebayes.frontend.traingood;"
oncommand="FireBayes.traingood()"/>
                  <menuitem label="&firebayes.frontend.trainbad;"
oncommand="FireBayes.trainbad()"/>
                  <menuitem label="------------------------"/>
                  <menuitem label="&firebayes.frontend.upload;"
oncommand="FireBayes.upload()"/>
                  <menuitem label="&firebayes.frontend.download;"
oncommand="FireBayes.download()"/>
           </menupopup>
    </toolbarbutton>
</toolbarpalette>
```

Kot lahko vidimo, je za to pripravljena značka <toolbarpalette>, pod njo naredimo nov gumb z značko <toolbarbutton>. Dodamo mu nekaj osnovnih lastnosti: tip, ki je v tem primeru meni, napis, sliko in pa napis, ki naj se pojavi, ko z miško počakamo nad gumbom. Sledijo posamezne menijske postavke. Vsaka vsebuje napis in pa funkcijo, ki naj se izvede ob kliku na določen gumb. Če želimo, da se funkcija dejansko izvede, moramo na začetku dodati še

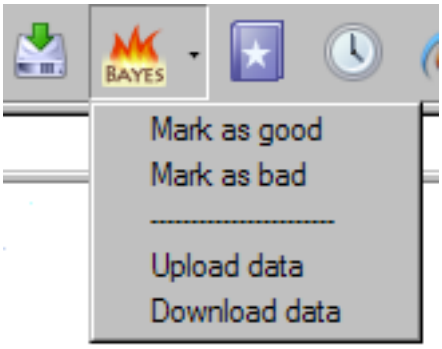
```
<script type="application/x-javascript" src="chrome://firebayes/content/firebayes.js"/>
```

Vidimo pa lahko tudi, da vrednosti pri lastnosti *label* ne vsebujejo teksta, pač pa gre za spremenljivke. Namen tega je, da lahko kasneje lokaliziramo dodatek z napisi v različnih jezikih. Vrednost za posamezen napis je shranjena v datoteki firebayes.dtd, ki se nahaja v mapi *chrome/locale* znotraj map dodatka.

Vsebina datoteke je takšna:

```
<!ENTITY firebayes.frontend.traingood "Oznaci kot dobro">
<!ENTITY firebayes.frontend.trainbad "Oznaci kot slabo">
```

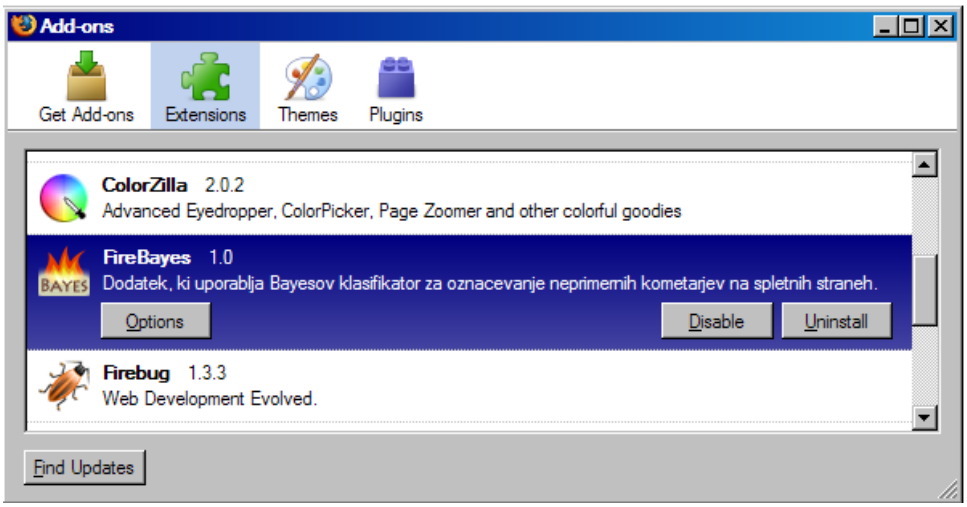
Kot lahko vidimo, v vsako vrstico dodamo ime spremenljivke ter tekstovno vrednost. Tako lahko enostavno priredimo dodatek še za druge jezike. Firefox potem ob zagonu preveri, katera različica brskalnika je nameščena in temu primerno jemlje napis iz ustrezne mape. Slika 7 kaže angleško različico menija.



Slika 7: Angleška različica menija v orodni vrstici

3.5.Nastavitve dodatka

Na koncu si oglejmo še okno z nastavitvami. Do njega dostopamo preko okna, kjer si lahko ogledamo dodatke (Slika 8).

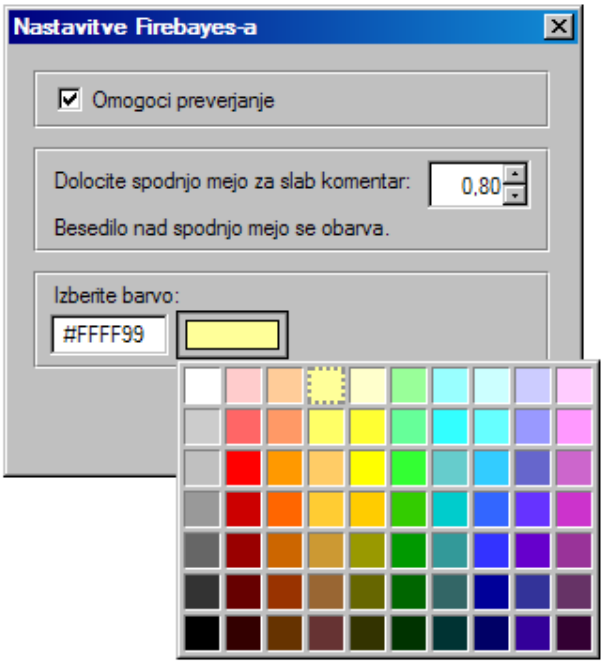


Slika 8: Izgled okna z dodatki, potem ko vanj dodamo FireBayes

Kot smo omenili že prej, nastavitve registriramo že prej v datoteki *chrome.manifest*.

```
<em:optionsURL>chrome://firebayes/content/firebayesset.xul</em:optionsURL>
```

S tem brskalniku povemo, kje se nahaja XUL datoteka s podatki o oknu. Ta stavek v oknu z dodatki omogoči gumb *Options*. Ob kliku nanj se nam odpre okno z nastavitvami (Slika 9).



Slika 9: Okno z nastavitvami za FireBayes

Samo okno vsebuje osnovne možnosti za nastavljanje dodatka. V njem lahko omogočimo oziroma onemogočimo preverjanje besed ob odprtju strani. Prav tako lahko določimo spodnjo mejo za slab komentar, ki je v vrednostih od 0.01 do 0.99. Zadnji del okna nam ponuja možnosti izbire barve, ki jo želimo imeti kot podlago za neprimerno besedilo.

Vrednosti, ki se pojavljajo kot nastavitve so shranjene v skupnem seznamu nastavitev brskalnika Firefox. Dovolj je že, da izdelamo datoteko z nastavitvami, ki ima končnico .js in jo damo v mapo */defaults/preferences* relativno glede na domačo mapo dodatka. Ob naslednjem zagonu bo dodatek te nastavitve registriral in dodal v skupni seznam nastavitev. Ta seznam vsebuje vse spremenljivke, ki jih za delovanje potrebuje Firefox, prav tako pa je v njem veliko nastavitev ostalih dodatkov. Datoteka z nastavitvami je takšna:

```
pref("extensions.firebayes.checkonload", true);
pref("extensions.firebayes.badrank", "0.90");
pref("extensions.firebayes.color", "#ffff7f");
```

Vrednosti pa so potem vidne v brskalniku (Slika 10):

extensions.dss.switchPending	default	boolean	false
extensions.enabledItems	**user set**	**string**	**aardvark@rob.brown:**
extensions.firebayes.badrank	**user set**	**string**	**0.8**
extensions.firebayes.checkonload	default	boolean	true
extensions.firebayes.color	**user set**	**string**	**#FFFF99**
extensions.firebug-service.breakOnErrors	default	boolean	false

Slika 10: Nastavitve, shranjene v notranjosti brskalnika Firefox

Sedaj si na kratko še oglejmo vsebino datoteke XUL, ki nam izriše okno z nastavitvami. Z značko <prefwindow> ustvarimo okno z nastavitvami.

```
<prefwindow type="prefwindow" id="firebayesset" title="&firebayes.set.title;" …
```

Nato ustvarimo ploščo z nastavitvami. Teh plošč bi lahko bilo več, vendar za naš dodatek zadostuje ena, saj ni potrebno ločevati posameznih nastavitev.

```
<prefpane id="mainoptions" label="FireBayes">
```

Zatem povežemo nastavitve, ki smo jih ustvarili prej, z oknom z nastavitvami. Vsako posebej dodamo v značko <preference>, ki jih zaokrožimo s <preferences>

```
<preferences>
    <preference id="fb_check" name="extensions.firebayes.checkonload"  type="bool"/>
    <preference id="fb_badrank" name="extensions.firebayes.badrank" type="string"/>
    <preference id="fb_color" name="extensions.firebayes.color" type="string"/>
</preferences>
```

Vsaka nastavitev ima id, ime, ki pomeni povezavo z globalno nastavitvijo ter tip spremenljivke, ki predstavlja nastavitev. Dodamo prvi del nastavitve, torej polje, ki ga označimo, če želimo preverjanje.

```
<groupbox>
   <hbox>
      <checkbox id="fb_onloadbox" label="&firebayes.set.enable;" preference="fb_check"/
>
   </hbox>
</groupbox>
```

Dodamo mu id, napis ob polju ter ga povežemo z nastavitvijo, ki smo jo definirali pred tem. Na podoben način dodamo še drugi del, ki vsebuje <textbox> s puščicami, s katerimi lahko uporabnik prestavlja spodnjo mejo za slab komentar.

```
<groupbox>
   <hbox>
      <label control="bad" style="padding-top:5px" value="&firebayes.set.limit;"/>
      <textbox id="badbox" type="number" size="4" decimalplaces="2" increment="0.01"
max="0.99" min="0.01" preference="fb_badrank"/>
   </hbox>
   <label id="fb_explanation" value="&firebayes.set.explanation;"/>
</groupbox>
```

Zadnji del okna, s katerim nastavljamo barvo označenega komentarja vsebuje <textbox>, v katerem je vrednost barve ter <colorpicker>, s katerim prikažemo orodje za prikazovanje barv. Oba povežemo s spremenljivko, ki predstavlja vrednost barve.

```
<groupbox>
   <label id="fb_colortext" value="&firebayes.set.color;"/>
   <hbox>
      <textbox id="fb_color" size="7" maxlength="7" preference="fb_color"/>
      <colorpicker type="button" id="color" width="60px" color="color"
preference="fb_color"/>
   </hbox>
</groupbox>
```

4.

5. SKLEPNE UGOTOVITVE

Z dodatkom FireBayes sem uporabnikom, ki nočejo na internetu brati napadalnih vsebin omogočil, da se temu izognejo. Izdelava dodatka za Mozilla Firefox je bila izjemno zanimiva, ne toliko zaradi samega dodatka in njegove uporabne vrednosti, kot predvsem zaradi združevanja različnih tehnologij, ki sem jih uporabil v nalogi, od popolnoma matematičnih in statističnih metod, ki sem jih uporabil za klasifikacijo do najnovejših spletnih tehnologij, kot je na primer AJAX. Vmes pa je še cel kup ostalih tehnologij in programskih jezikov, ki združijo to v eno celoto.

Največji problem, s katerim sem se moral spoprijeti, je bilo počasno delovanje, k čemur je največ pripomoglo dostopanje do podatkovne zbirke. Ta problem sem postopoma reševal in mislim, da je končni rezultat povsem zadovoljiv, saj se delovanje dodatka ne opazi tudi pri straneh, ki imajo veliko količino besedila.

Možnosti za izboljševanje je kar precej. Dodatek bo deloval boljše, ko se bo povečevala baza besed. Dejstvo je namreč, da je v začetku precej besed, ki se še ne pojavijo v bazi, ali pa se pojavljajo izjemno redko, najboljše pa je, če je takih besed čim manj. Dodatek bi lahko prevedel v še več različnih jezikov, vendar pa bi ga bilo potem potrebno naučiti tudi besed v drugih jezikih.

Med nujnejšimi izboljšavami pa je boljše pobiranje besedil iz spletne strani. Kot sem omenil že v delu, sem se za ta namen odločil pobirati besedilo, ki se nahaja v znački <p>. Vendar pa za učinkovito delovanje to ni dovolj, zato bi bilo potrebno tu izdelati bolj prilagodljivo pobiranje besedila, ki bi zaznalo vrsto foruma oz. bloga in besedilo ustrezno razdelilo na komentarje.

Čeprav je končni rezultat te naloge dodatek za spletni brskalnik Mozilla Firefox, pa bi lahko vse znanje uporabil tudi na drugačen način. Lahko bi klasifikator vgradil v novičarsko spletno stran, ki ima možnost dodajanja komentarjev. Ko bi uporabnik oddal komentar, bi ga lahko klasifikator ocenil in to oceno napisal ob komentarju ali pa neprimernega posebej označil. Na ta način bi bili obiskovalci že vnaprej opozorjeni, kateri komentar je vredno brati in katerega ne.

Viri in literatura

[1] P. Graham, A plan for spam. Dostopno na: http://www.paulgraham.com/spam.html
[2] SpamBayes, Bacground reading. Dostopno na: http://spambayes.sourceforge.net/background.html
[3] G. Robinson, A statistical approach to the spam problem. Dostopno na: http://www.linuxjournal.com/article/6467
[4] The joy of XUL. Dostopno na: https://developer.mozilla.org/en/The_Joy_of_XUL
[5] Building an extension. Dostopno na: https://developer.mozilla.org/En/Building_an_Extension
[6] Mozilla XUL. Dostopno na: https://developer.mozilla.org/en/XUL
[7] Wikipedia, Javascript. Dostopno na: http://en.wikipedia.org/wiki/Javascript
[8] About SQLite. Dostopno na: http://www.sqlite.org/about.html
[9] Wikipedia, PHP. Dostopno na: http://en.wikipedia.org/wiki/Php
[10] Wikipedia, MySQL. Dostopno na: http://en.wikipedia.org/wiki/Mysql