Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Florjan Bartol

Ocenjevanje energijske porabe spletnih strani z metodami strojnega učenja

diplomsko delo

univerzitetni študijski program računalništvo in informatika

Mentor: doc. dr. Matjaž Kukar

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Preverite hipotezo, da je mogoče z metodami strojnega učenja na podlagi strukturnih značilnosti spletnih strani oceniti njihovo energijsko zahtevnost med nalaganjem na mobilnem telefonu. Razvijte avtomatski merilni sistem, s katerim boste merili električni tok in napetost na telefonu med nalaganjem spletnih strani. Sistem uporabite za zbiranje podatkov o porabi množice popularnih spletnih strani in jih sestavite v učni problem združen s strukturnimi podatki o straneh. Na dobljeni množici podatkov preizkusite različne metode strojnega učenja in ovrednotite dobljene rezultate.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Florjan Bartol, z vpisno številko **63070028**, sem avtor diplomskega dela z naslovom:

Ocenjevanje energijske porabe spletnih strani z metodami strojnega učenja

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matjaža Kukarja,
- 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 15. junija 2014 Podpis avtorja:

Lepo se zahvaljujem vsem, ki so prispevali k nastanku diplomskega dela. Predvsem bi se rad zahvalil mentorju doc. dr. Matjažu Kukarju za vso pomoč, nasvete in odgovore, ki mi jih je nudil tekom izdelave.

Hvala doc. dr. Domnu Hudoklinu ter vsem članom Laboratorija za metrologijo in kakovost Fakultete za elektrotehniko za opremo, prostor za izvajanje eksperimentov ter pomoč pri meritvah. Hvala Patricku Meenanu, ki upravlja projekt WebPagetest, za omogočen dostop do WebPagetest API-ja, ki mi je precej olajšal delo.

Zahvaljujem se tudi podjetju Zemanta za prilagodljivost, razumevanje in podporo, še posebej mag. Dušanu Omerčeviću, ki mi je svetoval pri začetni izbiri problemske domene.

Moji dragi Martini.

Kazalo

Povzetek

Cilj diplomskega dela je bil preveriti hipotezo, da je mogoče z metodami strojnega učenja oceniti energijsko zahtevnost spletnih strani med nalaganjem na mobilnem telefonu. V ta namen je bil razvit avtomatski merilni sistem, sestavljen iz modificiranega mobilnega telefona, baterije ter dveh multimetrov, s katerima sta se merila tok in napetost med nalaganjem spletnih strani. Sistem je bil uporabljen za zbiranje podatkov o porabi strani, ti pa so bili nato združeni s strukturnimi podatki o straneh. Na dobljeni množici podatkov je bilo preizkušenih pet metod strojnega učenja: naivni Bayesov klasifikator, odločitveno drevo, k najbližjih sosedov, metoda podpornih vektorjev in naključni gozd. Rezultati so pokazali smiselnost pristopa in tako potrdili hipotezo, z nadgradnjami pa bi lahko bila dosežena še večja klasifikacijska točnost.

Ključne besede

Merjenje porabe, avtomatsko zbiranje podatkov, mobilne spletne strani, metode strojnega učenja

Abstract

The main goal of the thesis was to verify the hypothesis stating that it is possible to estimate energy consumption of a web page during loading on a mobile phone. For this purpose an automatic measuring system was developed, consisting of a modified mobile phone, battery and two multimeters, which were used to measure current and voltage during the page load. The system was used to gather energy consumption data, which was then joined with structural information about web pages. Five machine learning methods were tested on the resulting dataset: naive Bayes classifier, decision tree, k nearest neighbors, support vector machines and random forest. The results showed that this is a feasible approach and thus confirmed the hypothesis, while better accuracy could be achieved with certain improvements.

Keywords

Consumption measurement, automatic data collection, mobile web pages, machine learning methods

Poglavje 1

Uvod

V zadnjih letih so se na področju računalništva zgodili veliki premiki. Niso daleč časi, ko je bilo do svetovnega spleta možno dostopati le preko osebnih računalnikov, priklopljenih na počasne, analogne modeme. Temu primerne so bile takratne spletne strani - enostavne in skromno oblikovane, brez nepotrebnih dodatkov, a hkrati zelo omejene v svoji uporabnosti. Večinoma niso ponujale nič drugega kot informacije v tekstovni obliki in povezave na druge strani - skriptni jeziki so bili takrat še v povojih.

A tehnologija gre naprej. Ni se razvijala le strojna oprema, kar se kaže v obliki vedno zmogljivejših naprav, temveč je izredno napredovalo samo medmrežje ter seveda programska oprema. Spletni brskalniki so postali prava izvajalna okolja, v katerih lahko poganjamo vse od iger do konferenčnih video klicev. Skladno z razvojem so zrasla tudi pričakovanja uporabnikov - na spletne strani se ne gleda več kot le na enostavne nosilce tekstovnih informacij, temveč se od njih čedalje bolj pričakuje zmogljivost in hitrost, primerljiva z namiznimi aplikacijami. Ob tem ni nenavadno, da se je uveljavil izraz spletne aplikacije.

Seveda pa je potrebno omeniti še enega najbolj zanimih področij razvoja - napredek mobilnih tehnologij. Naprave z možnostjo dostopa do spleta dandanes nosimo kar v žepu - brez njih marsikdo ne zna več živeti. Vendar pa so mobilne naprave kljub svoji zmogljivosti vseeno precej različne od običajnih

računalnikov, kar pri brskanju po spletu prinese s seboj vrsto izzivov. Eden od teh je recimo hitrost, še posebej ob dejstvu, da 85% uporabnikov pričakuje, da se bo mobilna spletna stran naložila enako hitro ali hitreje od strani na običajnem računalniku. Drugi izziv pa je doseganje čim manjše porabe energije.

Ena od ključnih razlik med namiznimi in mobilnimi napravami je ravno v tem, da slednje poganja relativno majhna baterija z majhno kapaciteto. Če to povežemo z vedno večjimi pričakovanji uporabnikov, se hitro izkaže, da je potrebno z razpoložljivo energijo ravnati čim bolj varčno. Tovrstna odgovornost ni le na strani proizvajalcev, temveč tudi razvijalcev programske opreme. Sem spadajo tako mobilne aplikacije kot tudi mobilne spletne strani, ki jih uporabniki nalagajo v brskalniku na svojem telefonu. Raziskave so namreč pokazale, da so med stranmi nezanemarljive razlike v energijski potratnosti [2].

Glavni problem, s katerim se srečujejo razvijalci pri optimizaciji svojih spletnih aplikacij za čim manjšo porabo energije, je v pomanjkanju ustreznih orodij. Medtem, ko za analizo običajnih aplikacij obstaja nekaj možnosti, je situacija pri spletnih straneh veliko slabša: razvijalci nimajo nobene možnosti, da bi ocenili energijsko porabo svoje spletne strani na enostaven način.

V diplomskem delu smo se odločili preveriti, ali je mogoče zbrati primerne podatke o spletnih straneh in z metodami strojnega učenja mogoče ustvariti model, ki omogoča enostavno klasifikacijo spletnih strani glede na porabo energije. Namen takega modela je omogočiti razvijalcem, da vsaj v grobem ocenijo energijsko zahtevnost svoje spletne strani ter jo po potrebi optimizirajo. V ta namen smo razvili sistem za merjenje porabe energije na mobilnemu telefonu. S sistemom smo generirali učne primere, ki smo jih nato uporabili za medsebojno primerjavo različnih klasifikacijskih metod. Izbrani model smo uspešno uporabili za klasifikacijo novih primerov z relativno dobro točnostjo. S tem smo pokazali, da je pristop obetaven, omogočil pa nam je tudi oceno ekološkega vpliva v svetovnem merilu.

Poglavje 2

Merjenje porabe

Ključnega pomena za naše delo je ustrezna metoda za merjenje porabe energije na mobilnem telefonu. Ta mora biti dovolj natančna, da lahko z veliko ločljivostjo zaznava trenutne spremembe. Le tako lahko zanesljivo izmerimo porabo specifične sletne strani.

Za merjenje v grobem obstajata dve možnosti, opisani v nadaljevanju.

2.1 Vmesniki API

Odčitavanje porabe je mogoče preko vmesnikov API, ki so na voljo razvijalcem pri nekaterih mobilnih operacijskih sistemih. Razvijalcem aplikacij za operacijski sistem iOS je v razredu `UIDevice` na voljo lastnost `batteryLevel` [1], ki vrne trenutno stanje baterije, v operacijskem sistemu Android pa je na voljo razred `BatteryManager` [3], ki nudi podobno funkcionalnost.

Težava pri uporabi vmesnikov API je njihova nenatančnost. Razlog je v tem, da velika večina mobilnih telefonov ni opremljenih z ustreznimi senzorji, ki bi omogočali natančno merjenje stanja baterije. Zaradi tega gre večinoma le za približke. Omenjeni vmesniki so predvsem namenjeni temu, da mobilna aplikacija zazna nizko stanje baterije ter ustrezno prilagodi svoje delovanje.

Na voljo so tudi orodja za analizo delovanja aplikacij (*profiling tool*). Ta

med drugim v nekaterih primerih omogočajo tudi analizo porabe baterije, na primer orodje Instruments za iOS. Spet pa je treba poudariti, da so namenjena razvijalcem aplikacij in spremljanju porabe skozi daljše časovno obdobje, medtem ko gre v našem primeru za kratkotrajne meritve dolžine nekaj sekund ali minut. Zaradi tega podobno kot vmesniki API niso primerna za merjenje porabe pri spletnih straneh.

2.1.1 Preverjanje natančnosti

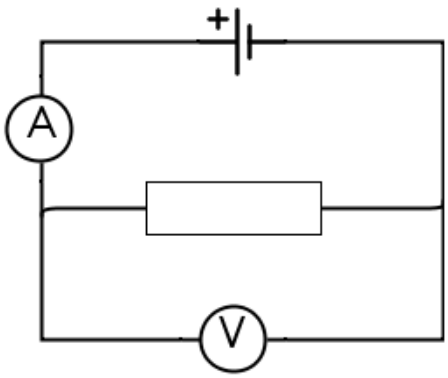
Kljub slabim obetom smo se vseeno odločili preveriti, ali bi bilo mogoče porabo energije učinkovito oceniti z uporabo vmesnikov API. Za testiranje smo uporabili telefon Apple iPhone 5. Da bi lahko dosegli čim večjo natančnost, smo se odločili uporabiti enega od vmesnikov API, ki jih Apple uporablja interno: `IOPowerSources.h`. Razvijalci naj jih v svojih aplikacijah ne bi uporabljali, zato jih je mogoče poganjati le na odklenjenih (*jailbroken*) napravah. Na tak način lahko dobimo podatke o trenutni kapaciteti baterije z natančnostjo 0.1%.

Natančnost smo testirali tako, da smo isto spletno stran stokrat naložili na telefonu, nato pa izračunali razliko med začetno in končno kapaciteto. To smo ponovili trikrat. Izkazalo se je, da so razlike med vsemi tremi poskusi zelo velike.

To je bil dokaz, da ta metoda žal ne nudi dovolj velike natančnosti. Poleg tega bi tudi v primeru, da bi bila natančnost ustrezna, na ta način težko zbrali veliko število učnih primerov.

2.2 Merjenje z multimetrom

Druga možnost je merjenje s pomočjo multimetra. Multimeter je elektronska merilna naprava, ki združuje več merilnih funkcij v eni enoti [4]. Običajno je možno z njim meriti napetost, tok in upornost, in sicer pri enosmernem ali izmeničnem toku.



Slika 2.1: Merjenje toka in napetosti.

Za merjenje porabe potrebujemo dva multimetra. S prvim merimo napetost v voltih in ga povežemo vzporedno, z drugim pa tok v amperih ter ga vežemo zaporedno (slika 2.1). Pridobivamo torej povprečen tok in povprečno napetost v intervalu meritve. Če za vsak interval pomnožimo ti dve vrednosti, tako dobimo povprečno moč za vsak interval meritve. Tako lahko izračunamo porabo v Wh po sledeči enačbi:

$$E = \int_{t_0}^{t_1} U(t) \cdot dt \tag{2.1}$$

Integriramo torej moč po času ter tako dobimo porabljeno energijo v določenem časovnem obdobju.

Metoda merjenja z multimetrom je veliko natančnejša kot odčitavanje porabe preko vmesnika API, še posebej v primeru uporabe kvalitetnih multimetrov, ki omogočajo visokofrekvenčno zajemanje vzorcev. Zaradi tega je za naš namen veliko primernejša. Vendar pa je sam proces merjenja veliko bolj zapleten, saj je potrebno multimetre priklopiti med baterijo in telefon, kar ni trivialno.

Rešitev tega problema je možna na več načinov. Kot primer: v [5] so avtorji s pomočjo 3D tiskalnika natisnili posebno držalo za baterijo ter nadomestek za baterijo, ki so ga vstavili v telefon namesto le-te. V oboje so vgradili ustrezne konektorje, kar jim je omogočilo povezavo obeh modulov,

posledično pa so lahko vmes priklopili multimetre za merjenje.

Poglavje 3

Merilni sistem

Pri snovanju sistema za merjenje porabe smo identificirali naslednje zahteve:

- Sistem mora omogočati čim bolj natančno merjenje porabe.
- Spletne strani se morajo na telefonu nalagati avtomatično. Prav tako se morajo avtomatično izvajati meritve.

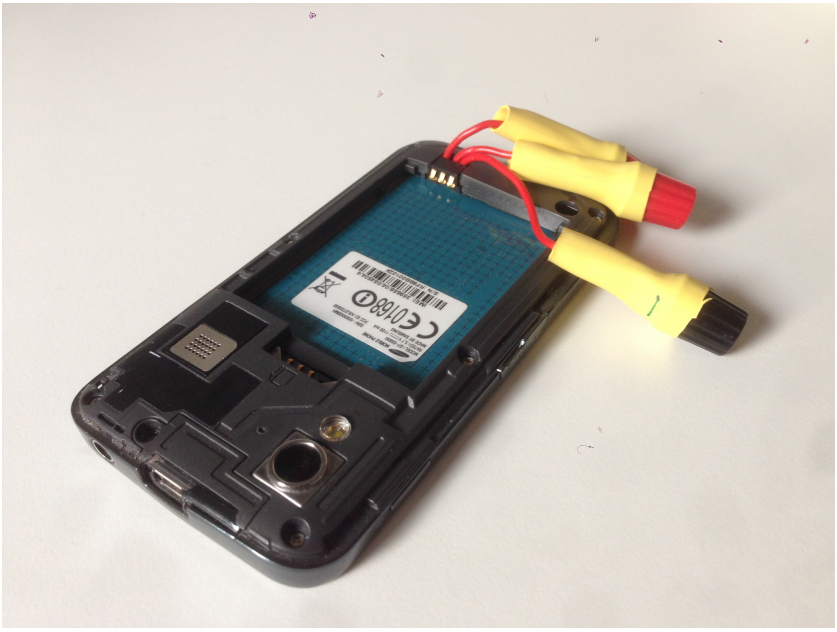
3.1 Implementacija merjenja

Kot smo opisali v poglavju 2, je za naš primer bolj smiselna odločitev merjenje z multimetrom. To je s seboj prineslo tudi nekaj izzivov.

3.1.1 Modifikacija telefona

Za eksperimente smo uporabili telefon Samsung Galaxy Ace S5830. Gre za telefon, izdan februarja 2011, s 3,5 palčnim zaslonom TFT ločljivosti 320x480 slikovnih točk, 800 MHz procesorjem, 278 MB pomnilnika RAM ter operacijskim sistemom Android 2.3 Gingerbread. Vsebuje 3,5 V litij-ionsko baterijo kapacitete 1350 mAh [8].

Kot rečeno, je eden glavnih problemov pri merjenju porabe telefona z multimetrom dejanski priklop med baterijo in konektorje telefona. Možnosti je več. Žal tiskanje nadomestkov s 3D tiskalnikom, ki bi jih lahko enostavno

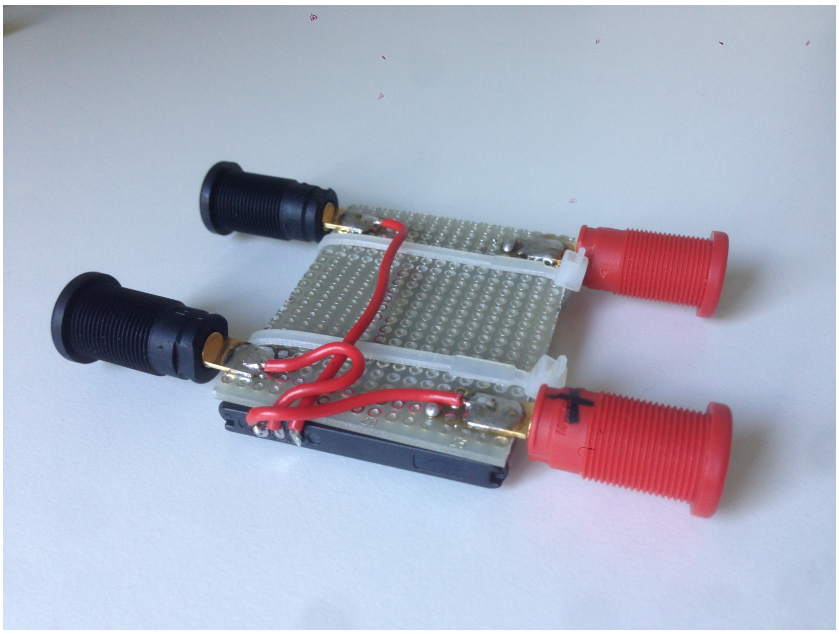


Slika 3.1: Modificiran telefon.

vstavili v telefon namesto baterije ter jih povezali z eksternim virom energije kot v [5] ni prišlo v poštev, saj 3D tiskalnika nismo imeli na voljo. Alternativna možnost je bila nakup dodatne, rabljene baterije, ki bi jo lahko odprli in odstranili vsebino iz ohišja, ohranili pa bi zunanje konektorje. Preko njih bi nato prazno ohišje povezali s pravo baterijo ter ga vstavili v telefon kot nadomestek. Problem tega pristopa je v tem, da je odpiranje litij-ionske baterije slabo dokumentirano in tudi nevarno, saj lahko pri tem pride do vžiga baterije.

Namesto tega smo se odločili za drugačen pristop. Izkaže se, da se da uporabljeni telefon precej enostavno razstaviti in tako dostopati do delov, skritih v notranjosti. Ugotovili smo, da je posledično mogoče na baterijske konektorje v telefonu prispajkati žice s priključki, kot prikazuje slika 3.1. Podobno smo naredili tudi z baterijo (slika 3.2). To nam je omogočilo enostavno povezavo obeh komponent, hkrati pa tudi vmesni priklop merilnih instrumentov.

Baterija in telefon sta povezana s tremi žicami. Prvi dve predstavljata



Slika 3.2: Baterija z dodatnimi konektorji.

pozitivni in negativni terminal, tretja pa je povezana z varnostnim internim termostatom. Ta zaznava morebitno previsoko temperaturo v bateriji ter tako služi kot dodaten varnostni ukrep pred kratkim stikom [9].

3.1.2 Multimeter

Za merjenje smo uporabili multimeter Agilent 34411A. To je digitalni multimeter s frekvenco merjenja 50kHz ter možnostjo komunikacije z računalnikom preko povezave USB ali preko mrežnega vmesnika [6]. Za naš eksperiment smo uporabili dva, enega za merjenje napetosti ter drugega za merjenje toka.

Oba multimetra smo povezali z računalnikom preko USB povezave. Uporabili smo računalnik s sistemom Windows, na katerega smo namestili program Digital Multimeter Connectivity Utility Software [7], ki ga nudi Agilent. Preko le-tega je mogoče enostavno upravljati z vsemi priklopljenimi multimetri hkrati. Omogoča nam spreminjanje vseh nastavitev in parametrov meritve, sinhronizacijo vseh merilnih naprav, izris grafov ter izvoz zajetih

Merilna funkcija	Napetost (DC)	Tok (DC)
Območje	10 V	3 A
Interval	200 ms	200 ms
NPLC	2	2

Tabela 3.1: Merilni parametri. NPLC - število ciklov električnega voda (*number of power line cycles*).

podatkov v več različnih formatov. Parametri, uporabljeni pri meritvah, so opisani v tabeli 3.1 .

Kot je razvidno iz tabele 3.1, smo za integracijski čas (*integration time*) uporabili 2 cikla električnega voda (*power line cycles*). Razlog za to je periodičen šum na električnem vodu. Če za integracijski čas uporabimo večkratnik trajanja cikla električnega voda, lahko periodični šum izničimo [11]. Hkrati to pomeni, da z daljšim integracijskim časom povečamo točnost meritev, vendar pa zmanjšamo ločljivost. V Sloveniji (kot tudi v večini ostalih držav) en cikel električnega voda ustreza 20 ms.

Agilent 34411A integracijski čas uporablja interno ter tako v našem primeru izmeri 25 vzorcev na sekundo [12]. Druga pomembna lastnost je interval zapisovanja. Oba multimetra smo na računalnik priklopili preko povezave USB. Vzorci, ki jih izmeri multimeter, se ponovno integrirajo ter se sproti shranjujejo v pomnilniku računalnika. Pri tem smo interval nastavili na 200 ms, kar pomeni, da smo tako dobili 5 integriranih vzorcev na sekundo. Parametre smo določili z eksperimentiranjem. Tako smo dobili rezultate, ki so predstavljali najboljše ravnovesje med ločljivostjo in natančnostjo.

3.2 Samodejno upravljanje telefona

Ker smo želeli zbrati čim več kakovostnih vzorcev, smo morali najti ustrezen način za izvedbo eksperimentov. Ročno nalaganje spletnih strani na mobilnem telefonu ni prišlo v poštev, saj je mnogo prepočasno, premalo natančno

ter težko ponovljivo. Namesto tega smo se odločili, da mora eksperiment potekati avtomatsko, brez nepotrebnega posredovanja uporabnika.

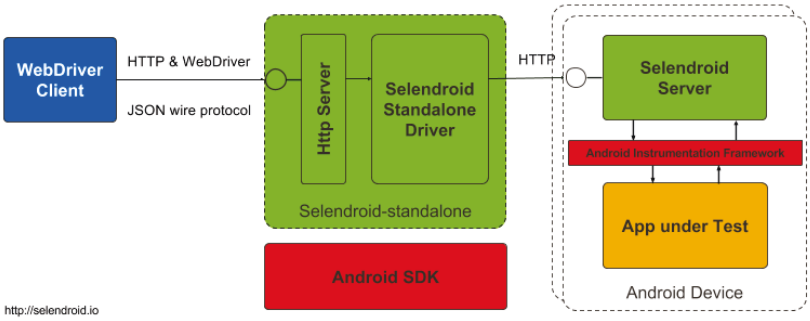
3.2.1 Selendroid

V ta namen smo uporabili programsko ogrodje Selendroid [13]. Omogoča nam avtomatizirano testiranje mobilnih aplikacij na sistemu Android. Temelji na ogrodju za instrumentacijo Androida (*Android Instrumentation Framework*) [14], teste zanj pa lahko pišemo v kateremkoli jeziku, ki podpira Selenium 2 Client API.

Uporabljamo ga tako, da na računalniku poženemo program, imenovan Selendroid klient, z ustreznimi nastavitvami. Z njimi določimo, katero aplikacijo želimo testirati. Klient ob zagonu poišče vse združljive emulatorje in naprave, ki so trenutno priklopljene na računalnik. Nanje naloži aplikacijo, ki jo želimo testirati, ter na vsaki napravi in emulatorju požene Selendroid strežnik. Aplikacija je lahko vnaprej prevedena (*built*), kar pomeni, da lahko testiramo tudi aplikacije, pri katerih nimamo dostopa do izvorne kode, temveč le končni paket `apk` (ta predstavlja Android aplikacijo). Vendar pa mora ta obstajati na računalniku, kjer poganjamo klienta, saj morata biti tako aplikacija kot Selendroid strežnik podpisana z istim certifikatom.

Klient, ki smo ga pognali na našem računalniku, nato preko protokola HTTP komunicira s strežnikom na mobilni napravi, ta pa preko ogrodja za intrumentacijo Androida upravlja z aplikacijo, ki jo testiramo. Teste pišemo z uporabo vmesnika Selenium 2 Client API. Shema arhitekture je prikazana na sliki 3.3.

Selendroid nam ne omogoča le testiranja mobilnih aplikacij, temveč tudi testiranje mobilnih spletnih strani. To je implementirano v obliki posebne aplikacije, ki za nalaganje strani uporablja komponento WebView. WebView je ena od komponent, ki jih razvijalci lahko uporabijo pri razvoju, ter omogoča integracijo spletnega brskalnika v aplikacijo. To nam je omogočilo, da smo spisali kratko skripto v Python-u, ki iz datoteke bere naslove spletnih strani ter jih zaporedoma nalaga na telefonu.



Slika 3.3: Arhitektura Selendroida.

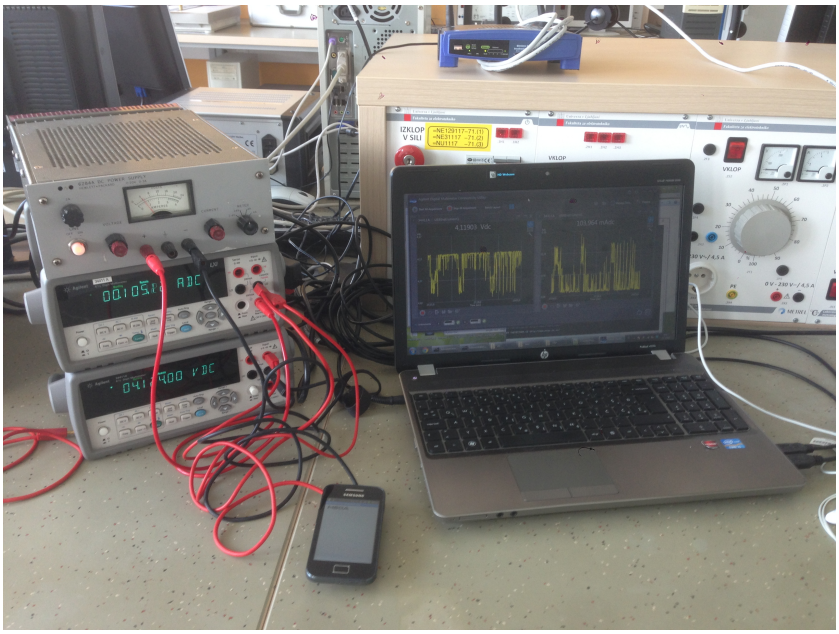
3.3 Končni sistem

Vse komponente smo na koncu povezali v celovit sistem za merjenje. Sistem je prikazan na sliki 3.4.

Sistem vsebuje naslednje sestavne dele:

- Multimeter za merjenje toka.
- Multimeter za merjenje napetosti.
- Modificiran telefon.
- Vir energije.
- Brezžični usmerjevalnik.
- Računalnik.

Kot vir energije smo sprva uporabljali baterijo, opisano v poglavju 3. To je sicer delovalo dobro, vendar se je izkazalo, da ta pristop ni najučinkovitejši. Baterija je bila zmožna med eksperimentom telefon poganjati le kakšno uro in pol. Nato jo je bilo potrebno ponovno napolniti, kar je trajalo dobro uro. To se je izkazalo za zelo nepraktično. Zaradi tega smo baterijo zamenjali z usmernikom, s katerim smo izmenični tok iz električnega omrežja z napetostjo 220 V pretvorili v enosmernega z napetostjo 4,1 V. Slednjo napetost smo



Slika 3.4: Sistem za merjenje porabe.

določili eksperimentalno z baterijo - tolikšna je bila izmerjena napetost pri polni bateriji.

3.3.1 Povezovanje komponent

Komponente smo med seboj združili tako, da smo najprej povezali telefon in vir energije. Zatem smo prvi multimeter, namenjen merjenju toka, povezali zaporedno, drugega, s katerim smo merili napetost, pa vzporedno.

Na računalniku smo pognali klienta za Selendroid, ki je skrbel za avtomatsko nalaganje strani. Nanj smo prav tako priklopili oba multimetra. Tako smo ju ustrezno nastavili in nato med merjenjem shranjevali vzorce.

Brezžični usmerjevalnik smo potrebovali iz dveh razlogov:

1. telefon je moral imeti dostop do interneta za nalaganje spletnih strani,

2. tako računalnik kot tudi telefon sta morala biti povezana v isto brezžično omrežje, saj sta med seboj komunicirala preko WiFi povezave.

Razlog za to je preprost: mobilni telefon s sistemom Android se navadno z računalnikom poveže preko povezave USB. Nezaželeni stranski učinek pri taki povezavi pa je, da se pri tem telefon napaja preko kabla, kar bi lahko vplivalo na naše meritve. Tega obnašanja žal ni mogoče preprečiti, zato smo to rešili s povezavo preko WiFi omrežja. Ta funkcionalnost je v sistemu Android dobro podprta, vendar pa je pri starejših verzijah za njen vklop potrebno imeti korenski dostop (*root access*).

Zaradi tega smo telefon pred eksperimenti tudi programsko ustrezno pripravili. Na njem smo omogočili korenski dostop ter z njim odstranili vse nepotrebne aplikacije, ki bi lahko tekle v ozadju in vplivale na meritve porabe. Telefon ni imel vstavljene kartice SIM, da morebitni klici ali sporočila SMS ne bi motila merjenja. Svetlost zaslona smo nastavili na najmanjšo. Omogočili smo povezavo z računalnikom preko WiFi omrežja.

To je pripeljalo do dodatnega zapleta: telefon se med eksperimentom ni smel avtomatsko zakleniti, saj bi bil eksperiment s tem prekinjen. To se sicer v sistemu Android da nastaviti, vendar le, če je telefon priklopljen preko povezave USB - v nasprotnem primeru je najdaljši čas aktivnosti 30 minut. Ta problem smo uspešno rešili z aplikacijo Screen On Toggler [15].

3.4 Izvedba eksperimenta

Ko smo uspešno postavili sistem za merjenje porabe, je prišlo na vrsto zbiranje vzorcev. Najprej je bilo treba določiti seznam spletnih strani, ki se bodo analizirale. Za ta namen smo se odločili, da je najbolj smiselno analizirati trenutno najpopularnejše strani (torej strani z največjim številom obiskov). Seznam teh strani je mogoče dobiti na spletni strani www.alexa.com. Svojim obiskovalcem ponujajo razširitev za brskalnike v obliki orodne vrstice, preko katere sledijo uporabnikom in zbirajo podatke o obiskih spletnih strani. Glede na to, da je delež uporabnikov interneta, ki imajo orodno vrstico nameščeno, precej majhen (ocenjuje se na okoli 10 milijonov uporabnikov), ti rezultati niso zelo natančni. Kljub temu pa zadoščajo našim

Time	Elapsed Time (sec)	Adc	Vdc
4.6.2014 08:13:37,267	3,3901939	0,100703478	4,13387852
4.6.2014 08:13:37,480	3,6032061	0,097586851	4,13283124
4.6.2014 08:13:37,685	3,8077177	0,100438339	4,13388551

Tabela 3.2: Primer vzorcev.

potrebam.

Za naš vzorec smo izbrali 700 najpopularnejših spletnih strani z omenjenega seznama. Najpopularnejše strani na seznamu so po vrsti google.com, facebook.com, youtube.com, yahoo.com, baidu.com, wikipedia.org, qq.com, taobao.com, twitter.com in amazon.com.

Eksperiment je potekal tako, da smo strani na telefonu nalagali zaporedoma. Vsako stran smo pustili nalagati 60 sekund. Nato smo nalaganje prekinili (ne glede na to, ali se je stran naložila do konca ali ne) ter začeli nalagati naslednjo stran. Pri vsaki strani smo v posebno datoteko zapisali točen čas pričetka nalaganja. Ta čas smo po koncu merjenja uporabili za določanje, katere meritve multimetrov pripadajo kateri strani.

Ves čas eksperimenta smo z obema multimetroma merili napetost in tok ter vzorce sproti zapisovali v datoteko na računalniku. Po koncu eksperimenta smo podatke izvozili v format `.tsv` (*tab-separated values*). Primer končnega rezultata je prikazan v tabeli 3.2.

Porabo smo prvim 100 stranem v seznamu izmerili dvakrat, saj smo želeli med analizo rezultatov primerjati tudi, ali je poraba specifične spletne strani konstantna, ali pa se ob vsakem nalaganju razlikuje ter za koliko.

Zbiranje podatkov je trajalo precej časa. Veliko je k temu prispevalo tudi dejstvo, da je aplikacija na telefonu relativno pogosto prenehala delovati, predvidoma zaradi pomanjkanja pomnilnika na telefonu. Zaradi tega je bilo eksperiment potrebno večkrat ponovno zagnati, kar pomeni, da ni deloval popolnoma avtonomno.

Slika 3.5: Primer grafa pri merjenju napetosti in toka.

3.4.1 Računanje porabe

Iz zbranih meritev napetosti in toka smo po enačbi 2.1 za vsako stran izračunali, koliko energije je porabila v eni minuti od začetka nalaganja.

Z eksperimentom smo določili tudi spodnjo mejo porabe, torej količino energije, ki jo porabi telefon v eni minuti, če se na njem ne nalaga nobena stran. Sem je med drugim všteta poraba zaslona in osnovnih procesov operacijskega sistema. Ker je ta energija skupna vsem stranem, smo jo odšteli od končnih rezultatov meritev, ter tako dobili normalizirano porabo.

Poglavje 4

Podatki o spletnih straneh

Za gradnjo ustrezne množice učnih primerov seveda ni dovolj le energijska poraba vsake spletne strani. Da bi lahko uporabili klasične metode stojnega učenja, mora biti vsaka stran opisana z množico dodatnih atributov.

Tukaj je na mestu vprašanje, s katerimi atributi lahko spletne strani pravzaprav opišemo. V splošnem delimo podatkovno rudarjenje spleta in spletnih strani v tri glavne skupine [16]:

- Rudarjenje uporabe spleta (*web usage mining*) se nanaša na odkrivanje vzorcev v obnašanju uporabnikov na specifičnih spletnih straneh. Izvaja se z analizo strežniških zapisov (*server logs*), v katerih so navadno zabeležene interakcije uporabnikov s stranjo.

- Rudarjenje vsebine spleta (*web content mining*) iz vsebine spletnih strani izlušči uporabne informacije in znanje. To omogoča na primer grupiranje spletnih strani glede na njihovo tematiko, ter indeksiranje strani, kar omogoča iskanje po njihovi vsebini.

- Rudarjenje strukture spleta (*web structure mining*) se posveča odkrivanju uporabnega znanja iz povezav med stranmi, ki predstavljajo zgradbo spleta. Tako je na primer mogoče določiti pomembne spletne strani, kar s pridom izkoriščajo spletni iskalniki. V širšem smislu

lahko pomeni tudi analizo strukture spletne strani v smislu zgradbe dokumenta HTML ali uporabljenih virov.

Z informacijami, ki jih lahko zberemo z rudarjenjem uporabe spleta, bi bilo sicer mogoče določiti tipično obnašanje uporabnikov na specifični spletni strani in meriti porabo pri različnih načinih uporabe, vendar pa pri našem eksperimentu obnašanje uporabnikov ne igra nobene vloge. Poleg tega je rudarjenje uporabe spletne strani mogoče le, če imamo dostop do strežniških zapisov.

Podobno velja za rudarjenje vsebine spleta. Vsebina spletne strani in informacije na njej ne vplivajo na porabo baterije, prav tako ne njihova tematika. Kot so pokazali raziskovalci v [2], na porabo spletne strani najbolj vpliva njena struktura. S tem je mišljena količina podatkov, ki se prenesejo v uporabnikov brskalnik pri obisku strani, število zahtevkov, ki jih stran izvede, porazdelitev obojega med vsebinske tipe in tako dalje.

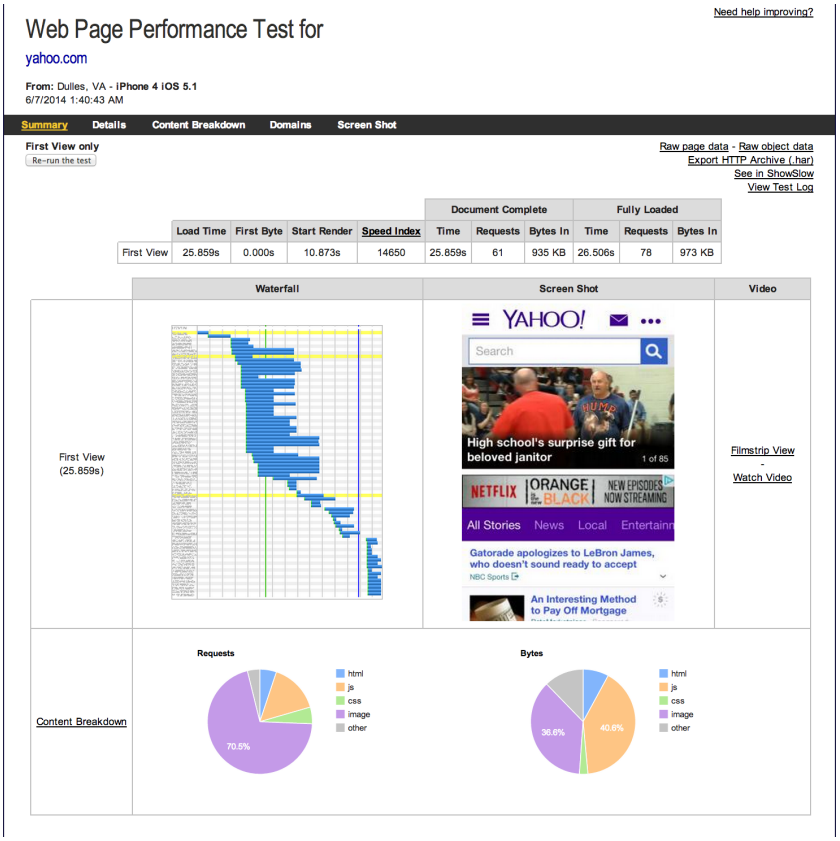
4.1 Zbiranje podatkov o strukturi strani

Naslednje vprašanje je, kako pristopiti k zbiranju podatkov. V idealnem scenariju bi medtem, ko bi merili porabo, hkrati tudi analizirali zgradbo spletne strani. V praksi je to težavno iz dveh razlogov. Vzporedna analiza bi lahko vplivala na rezultate merjenja porabe, poleg tega pa je to na telefonu, ki smo ga uporabili pri eksperimentih, težko izvedljivo. Lahko bi uporabili enega od namiznih brskalnikov in ga nastavili tako, da se spletnim stranem predstavlja kot mobilni brskalnik. Kljub temu pa je zbiranje podatkov zapleteno in zamudno.

Namesto tega obstaja boljša možnost - Http Archive [17]. Gre za zbirko strukturnih podatkov o spletnih straneh, kjer že več kot 10 let za več kot 10000 najpopularnejših spletnih strani beležijo in hranijo podatke, kot so na primer velikost strani, zahtevki, ki jih je stran izvedla med nalaganjem, čas nalaganja in tako dalje. Pregled nekaterih atributov je na voljo v tabeli 4.1.

Ime	Opis
renderStart	čas prikaza prvega vidnega rezultata na zaslonu
onContentLoaded	čas sprožitve dogodka `DOMContentLoaded`
onLoad	čas sprožitve dogodka `load`
fullyLoaded	čas do popolne naložitve strani
visualComplete	čas do vizualno dokončane strani
PageSpeed	ocena hitrosti strani na lestvici med 1 in 100
SpeedIndex	hitrost izrisa strani
reqTotal	skupno število zahtevkov
reqHtml	število zahtevkov za HTML
reqJS	število zahtevkov za JavaScript
reqCSS	število zahtevkov za CSS
reqImg	število zahtevkov za slike
bytesTotal	skupno število prenesenih bajtov
bytesHtml	skupno število bajtov v obliki HTML
bytesJS	skupno število bajtov v obliki Javascripta
bytesCSS	skupno število bajtov v obliki CSS-ja
bytesImg	skupno število bajtov v obliki slik
numDomains	število različnih dostopanih domen
numRedirects	število preusmeritev
numErrors	število neuspešnih zahtevkov
numHttps	število zahtevkov preko HTTPS
numCompressed	število zahtevkov s stisnjenimi podatki
maxageNull	število zahtevkov brez predpomnenja
gzipTotal	skupno število bajtov v stisnjenih resursih

Tabela 4.1: Primer atributov spletnih strani v Http Archive-u.



Slika 4.1: Primer rezultata na webpagetest.org.

Zbiranje podatkov poteka s pomočjo orodja WebPagetest [18]. To je odprtokodni projekt, s katerim je mogoče testirati hitrost spletnih strani, vendar pa je poleg tega sposoben izmeriti tudi množico drugih lastnosti, ki se lahko določijo med nalaganjem. Preko svoje spletne strani nam omogoča, da izberemo testno lokacijo in testni brskalnik (ponujeno s strani partnerjev oziroma sponzorjev) in vnesemo naslov strani, ki jo želimo testirati. Test se zatem izvede avtomatsko, na strani z rezultati pa lahko nato med drugim vidimo izpis vseh zahtevkov skupaj s podrobnimi opisi, čas nalaganja ter celo videoposnetek nalaganja strani. Omogoča nam tudi prenos podatkov v formatu JSON. Primer rezultata je viden na sliki 4.1.

Http Archive vsakih 15 dni preko zasebne instance WebPagetest-a stestira

preko 10000 najpopularnejših spletnih strani v dveh brskalnikih: v Internet Explorerju 9, ki teče v operacijskem sistemu Windows, ter v Safari-ju na iPhone-u 4. Vsi podatki so pretvorjeni v ustrezno obliko in shranjeni v podatkovni bazi. V taki obliki so na voljo tudi za prenos.

Za naš eksperiment bi bili idealni podatki, zajeti z iPhone-om 4. Toda za to smo se morali najprej prepričati, da se spletne strani na iPhone-u obnašajo približno enako, kot na telefonu s sistemom Android, saj smo naš eksperiment izvajali na slednjem.

Da bi to preverili, smo se poslužili uporabe WebPagetest-a. Kot omenjeno, nam ta ponuja možnost testiranja preko svoje spletne strani. Na voljo imamo nekaj mobilnih brskalnikov, med drugim tudi iPhone 4 ter Google Nexus S, telefon z naloženim sistemom Android 2.3 Gingerbread. Tako smo lahko primerjali, ali je zgradba in obnašanje pri istih spletnih straneh podobna pri obeh sistemih.

Ker smo za to potrebovali dovolj velik vzorec, bi ročno testiranje trajalo predolgo. Na srečo WebPagetest ponuja vmesnik API preko HTTP protokola, preko katerega lahko programsko izvajamo veliko število testov in na koncu prenesemo rezultate. Za uporabo je potrebno zaprositi za ključ, ki nam omogoči dostop.

Izmed vzorca strani, ki smo jih uporabili za naš eksperiment, smo izbrali naključnih 50 strani ter jih testirali tako na iPhone-u kot tudi na Nexus-u. Rezultat vsakega testa smo izvozili v obliki JSON datoteke. Za analizo rezultatov smo napisali skripto, ki je iz vsakega JSON dokumenta pridobila željene atribute in jih primerjala med seboj.

Za vsako stran smo paroma primerjali vrednosti specifičnih atributov ter izračunali razliko med njima po enačbi 4.1. Nato smo srednjo napako določili s pomočjo mediane.

$$d_r = \frac{|x - y|}{max(|x|, |y|)} \tag{4.1}$$

Primerjali smo števila zahtevkov ter količino prenesenih bajtov po podatkovnih tipih.

Izkaže se, da povprečna srednja napaka znaša dobrih 22%. Rezultati so

torej pokazali, da je struktura večine spletnih strani precej podobna ne glede na to, kje jih naložimo. Zaradi tega smo zaključili, da je podatke iz Http Archive-a smiselno uporabiti za naše eksperimente.

Podatki, ki smo jih uporabili, so bili torej zbrani na telefonu Apple iPhone 4. To pomeni, da so primerni za naš eksperiment, saj so naše meritve relevantne le za mobilne naprave.

Poglavje 5

Vizualizacija zbranih podatkov

Podatke, pridobljene na način, opisan v poglavju 4, smo združili s podatki o porabi energije, ki smo jo določili skozi naš eksperiment. Tako smo sestavili množico primerov. Da bi jo bolje razumeli, smo si pomagali z različnimi metodami vizualizacije primerov.

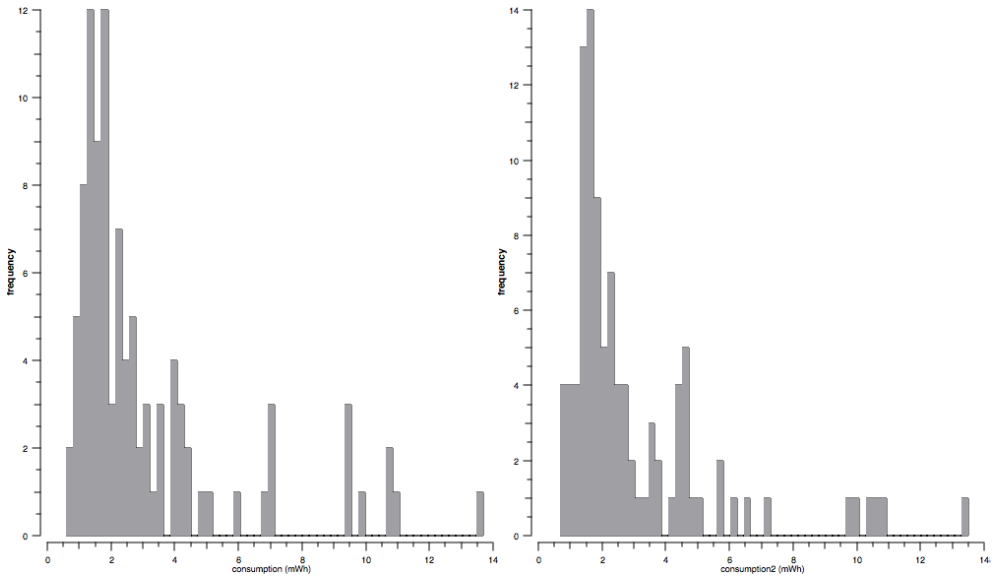
Pregled učne množice podatkov je razkril, da jo sestavlja 713 vzorcev. Vsak od njih je opisan s 65 atributi, od tega je 18 diskretnih in 45 zveznih.

5.1 Ocena šuma pri meritvah

Kot smo omenili v poglavju 3.4, smo prvim 100 spletnim stranem porabo izmerili dvakrat. Namen dvojne meritve je bil oceniti, ali se poraba energije od meritve do meritve razlikuje in za koliko. Slika 5.3 prikazuje primerjavo distribucij porabe v prvi in drugi ponovitvi poskusa v obliki histogramov.

Da bi lažje videli morebitno razliko med obema atributoma, smo ju vizualizirali v obliki razpršenega grafa (slika 5.2).

Iz grafa je razvidno, da je pri večini primerov poraba v obeh poskusih zelo podobna, vendar pa se pogosto vsaj nekoliko razlikuje. To ni nenavadno, saj na porabo vpliva mnogo dejavnikov. Svetovni splet je izredno dinamično okolje, zato je pri tovrstnih poskusih nemogoče popolnoma izničiti vse zunanje vplive. Zaradi tega je lahko čas, ki se porabi za prenos podatkov pri



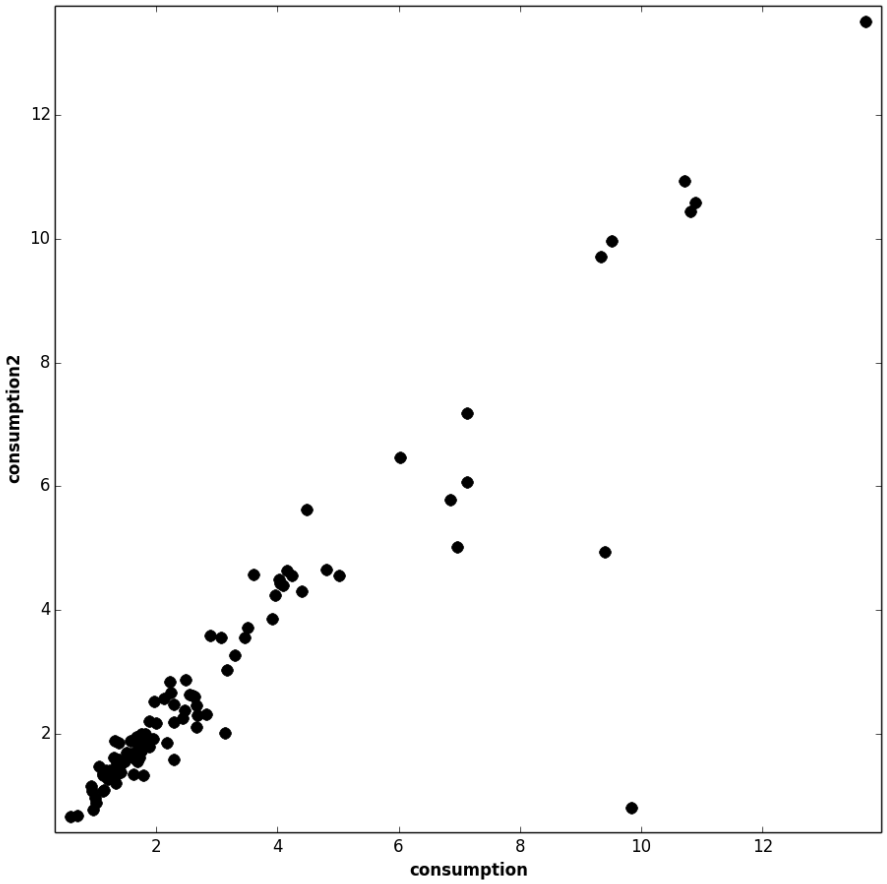
Slika 5.1: Primerjava distribucije porabe med obema ponovitvama.

posameznem zahtevku, vsakokrat nekoliko drugačen. Poleg tega se lahko spletne strani ob vsakem obisku obnašajo različno. Večina spletnih strani je dinamičnih - njihova vsebina se ves čas do neke mere spreminja. Primer tega so tudi spletni oglasi, ki se pogosto razlikujejo ob vsakem obisku. Lahko se zgodi, da so nekateri zahtevki neuspešni, posledično pa se del podatkov sploh ne prenese.

Opazimo pa lahko tudi nekaj očitnih izjem. Možnih je več razlag za ta pojav:

- Spletna stran je bila v enem od poskusov nedosegljiva.
- Stran se je v obeh primerih naložila, vendar v precej drugačni obliki. Primer je na primer stran, ki ob nalaganju vračajočemu uporabniku prikaže celotno naslovno stran, novemu pa le vabilo, naj si na svoj telefon namesti mobilno aplikacijo. V slednjem primeru je poraba običajno veliko manjša.

V splošnem lahko vidimo, da je poraba relativno podobna tudi skozi več ponovitev eksperimenta.



Slika 5.2: Razpršeni graf porabe pri obeh ponovitvah poskusa.

5.2 Distribucija energijske porabe

Seveda nas je najprej zanimalo, kakšna je distribucija porabe energije v celotni učni množici. To je razvidno iz slike 5.3.

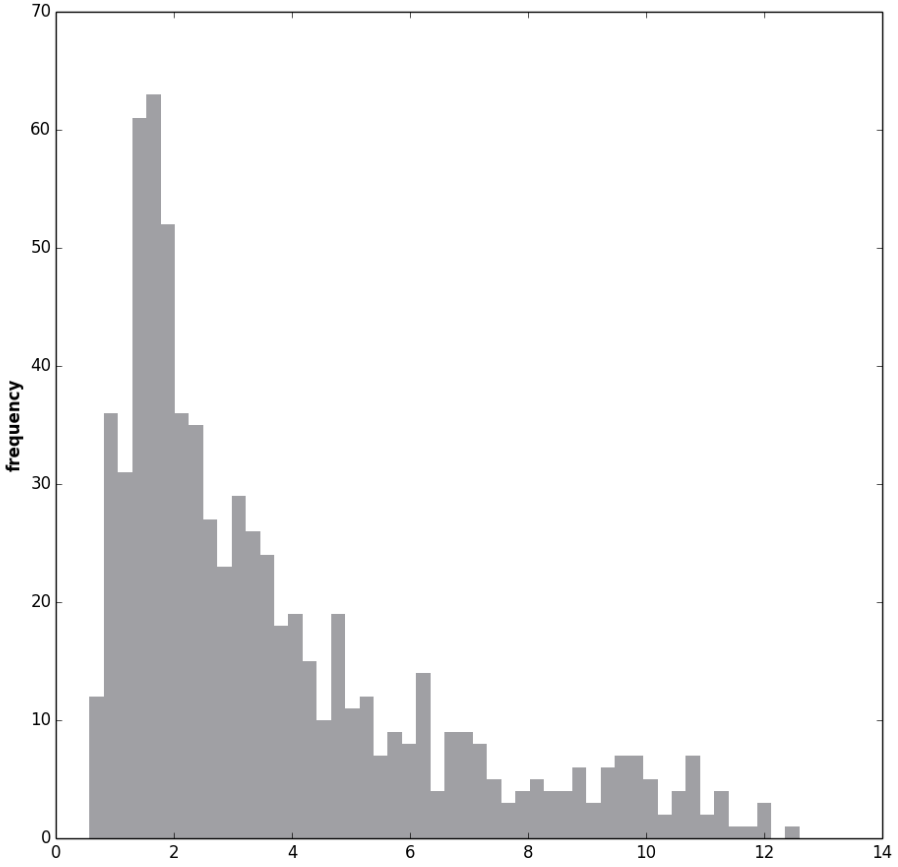
Poraba obsega vrednosti na intervalu od 0.57 mWh do 12.6 mWh. Pri celotni učni množici se je distribucija porabe v primerjavi z množico prvih 100 strani nekoliko spremenila - v večji množici je večji delež strani z večjo porabo (nad 5 mWh). Celotna množica je videti nekoliko lepše porazdeljena. Največ strani ima porabo okoli 2 mWh.

Poleg tega smo izračunali tudi vrednosti nekaterih statističnih mer. Povprečna vrednost je 3.71 mWh, standardni odklon je enak 2.71 mWh, mediana pa je enaka 2.75.

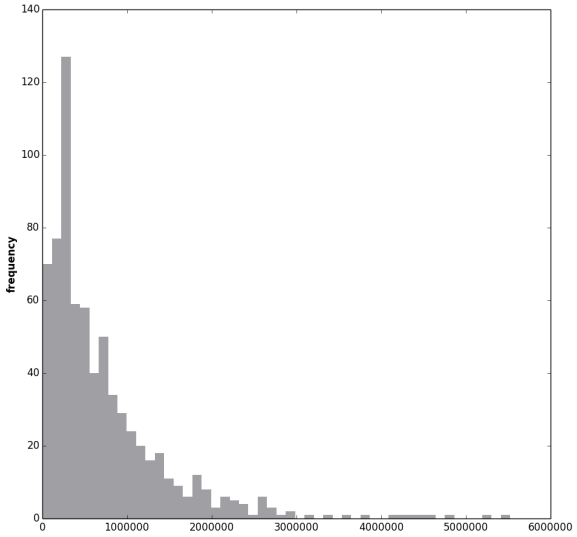
5.3 Vpliv ostalih atributov na porabo

Zanimalo nas je tudi, kako na porabo vplivajo ostali atributi. Najboljši način vizualizacije v tem primeru predstavlja razpršeni graf, zanima pa nas tudi distribucija atributov. Slike 5.4, 5.5, 5.6 in 5.7 predstavljajo primere vizualizacij za atributa `bytesTotal` in `numCompressed`.

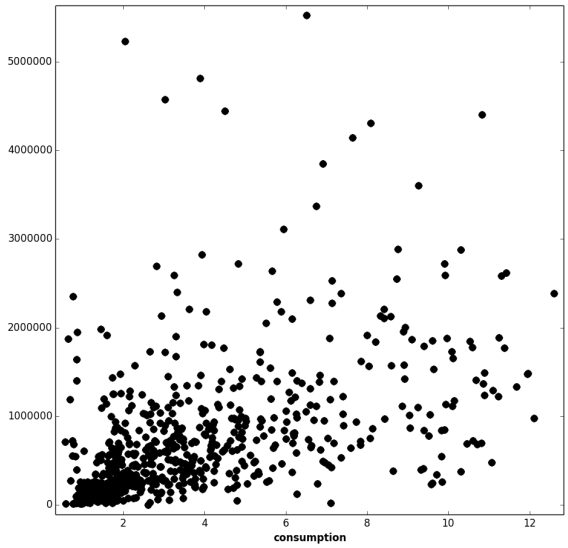
Opazimo lahko, da je distribucija pri teh dveh atributih podobna, kot distribucija porabe energije. Iz razpršenih grafov je mogoče razbrati, da obstaja korelacija med atributi.



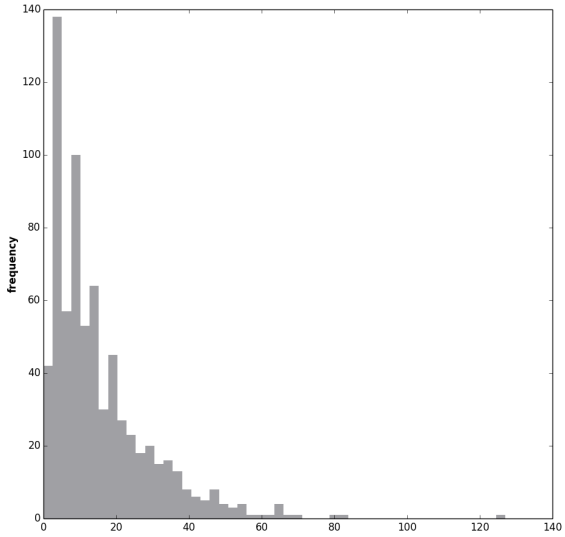
Slika 5.3: Distribucija energijske porabe v učni množici.



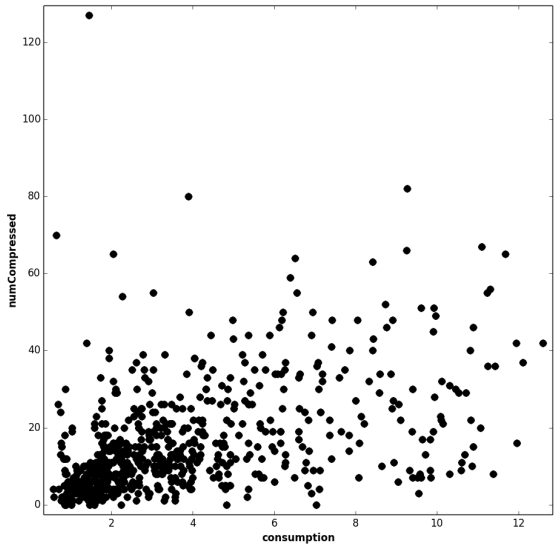
Slika 5.4: Distribucija skupnega števila prenesenih bajtov.



Slika 5.5: Razpršeni graf skupnega števila prenesenih bajtov in energijske porabe.



Slika 5.6: Distribucija skupnega števila stisnjenih podatkov.



Slika 5.7: Razpršeni graf skupnega števila stisnjenih podatkov in energijske porabe.

Poglavje 6

Modeliranje podatkov

Za bolj učinkovito klasifikacijo smo se odločili zbrane učne primere obdelati tako, da smo z njimi dosegli čim večjo klasifikacijsko točnost. Načini, na katere smo to storili, so opisani v nadaljevanju.

6.1 Predprocesiranje podatkov

Preden smo se lahko lotili analize podatkov, je bilo treba te ustrezno pripraviti. Prvi korak je bil pregled atributov ter izločanje nekoristnih z uporabo domenskega znanja. V množici atributov je bilo precej takih, ki so predstavljali meta podatke o strani ali testu. Zaradi tega smo lahko izločili atribute `url`, `urlShort`, `startedDateTime`, `wptid`, `rank`, `pageid`, `createDate`, `archive`, `label`, `crawlid`, `wptrun`, in `urlhash`. Nekateri so povezani z naslovom strani, drugi predstavljajo različne identifikatorje, začetek in zaporedno številko testa in tako dalje.

Ugotovili smo tudi, da sicer ni atributov, pri katerih bi vrednost manjkala samo za določen del vzorcev, bilo pa je nekaj takih, kjer je vrednost manjkala za vse vzorce. Ti atributi so bili `cdn`, `TTFB`, `onContentLoaded` in `PageSpeed`.

Nekateri atributi so imeli za vse vzorce nastavljeno enako vrednosti, in sicer `reqJson`, `bytesJson`, `numDomElements`, `maxageNull`, `gzipTotal`, `gzipSavings` in `connections`. Tudi take atribute smo odstranili, saj k ana-

lizi ne prispevajo ničesar.

Skupno smo tako odstranili 23 atributov. Preostali vzorci so bili tako opisani s skupno 40 atributi, od tega je bilo 39 zveznih in 1 diskreten.

6.2 Pretvorba v klasifikacijski problem

Energijska poraba, ki smo jo izmerili med našimi eksperimenti, v naši množici učnih primerov predstavlja neodvisno spremenljivko. Ta atribut je zvezen, kar pomembno vpliva na naše odločitve glede izbire ustreznega modela. Zvezna neodvisna spremenljivka implicira, da gre za regresijski problem. To pomeni, da je potrebno za napovedovanje uporabiti regresijski prediktor, čigar naloga je iz množice učnih primerov izračunati zvezno funkcijo, s katero lahko določimo vrednost regresijske spremenljivke novih primerov [19].

Pri tem se postavlja vprašanje, ali je naš problem smiselno obravnavati kot regresijskega. Vrednosti, ki smo jih izmerili, so namreč zelo specifične. Veljajo v uporabljenem eksperimentalnem okolju, vendar pa ne moremo dokazati, da bi bila absolutna poraba enaka tudi v drugačnih pogojih. Pri uporabi drugačnega telefona se lahko poraba komponent razlikuje, na primer poraba zaslona, sprejemnika WiFi, procesorja ter grafičnega čipa. Na porabo lahko vpliva tudi razpoložljivo omrežje: njegova latenca, pasovna širina ter vrsta.

Ob tem je očitno, da tak model ne more izpolnjevati svojega namena, torej da ne omogoča enostavnega določanja, ali je stran potratna ali ne. V primeru regresijskega modela kot rezultat dobimo absolutno porabo v mWh, ki jo moramo ustrezno interpretirati. Vendar pa ne glede na razlike v okolju struktura spletnih strani ostane enaka, kar pomeni, da lahko rezultate uporabimo za medsebojno primerjavo strani.

Zaradi tega se izkaže, da je neodvisno spremenljivko bolj smisleno diskretizirati ter tako problem iz regresijskega pretvoriti v klasifikacijskega. Vse učne primere lahko glede na izmerjeno porabo energije razvrstimo v majhno število razredov. Ta pristop ima poleg tega, da rešimo prej opisani problem,

še dodatno prednost: z njim lahko omilimo vpliv šuma [19], ki smo ga opisali v poglavju 5.1. Namreč, če smo strani prej v več ponovitvah poskusa izmerili nekoliko različne vrednosti, je zdaj velika verjetnost, da vse meritve padejo v isti razred. Še vedno to ne reši problema izjem, za katere v več meritvah izmerimo zelo različen rezultat, vendar pa je takih primerov malo (kot smo pokazali na sliki 5.2).

6.2.1 Določanje intervalov

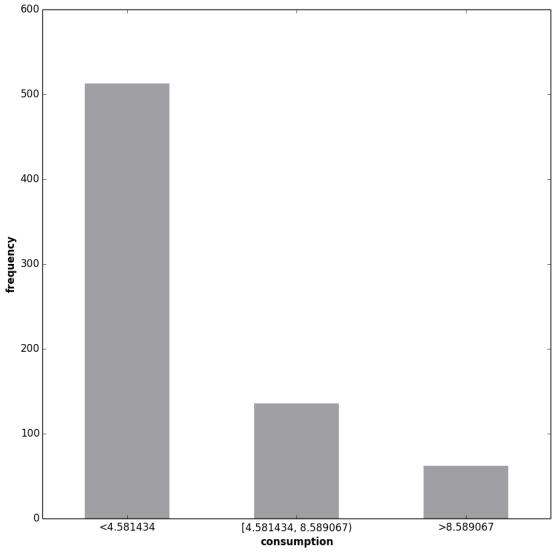
Večina atributov v naši učni množici je zveznih. Ker smo se odločili problem rešiti s klasifikacijskimi metodami in ker zna veliko klasifikacijskih metod delati le z diskretnimi atributi, smo se odločili poleg neodvisne spremenljivke diskretizirati tudi vse ostale.

Metode diskretizacije glede na to, ali pri svojem delovanju upoštevajo neodvisni atribut, delimo v dve skupini: nadzorovane in nenadzorovane [20]. Diskretizacijo neodvisne spremenljivke lahko opravimo s katero od nenadzorovanih metod. Najbolj pogosta in najenostavnejša med njimi je deljenje (*binning*), ki se lahko izvaja na dva načina:

- glede na enako širino (*equal-width binning*), kjer zvezno spremenljivko razdelimo na enako široke intervale, ter
- glede na enako frekvenco (*equal-frequency binning*), pri katerem meje intervalov postavimo tako, da v vsakega pade približno enako število učnih primerov.

Glede na prisotnost šuma v podatkih je smiselno vzorce razdeliti v majhno število intervalov. Odločili smo se za 3. Prvi predstavlja strani z nizko porabo, v drugega padejo strani, ki porabijo povprečno količino energije, v zadnjega pa strani z zelo visoko porabo energije.

Delitev na intervale bi lahko izvedli z eno od zgoraj opisanih metod, vendar pa imata obe določene pomanjkljivosti. Pri deljenj glede na enako širino kot rezultat dobimo zelo neuravnoteženo množico učnih primerov, kot

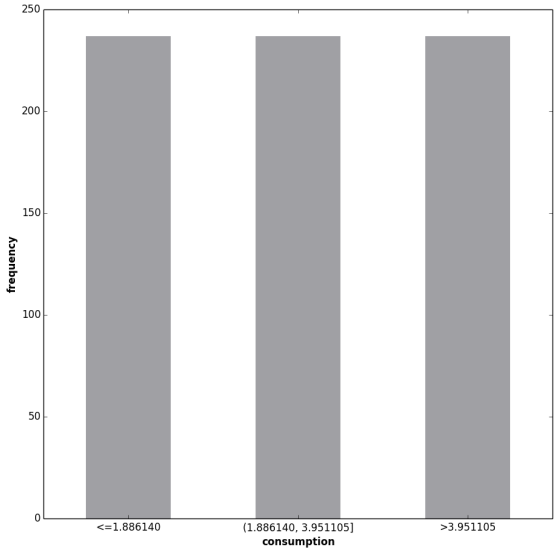


Slika 6.1: Deljenje glede na enako širino.

je razvidno s slike 6.1. To lahko predstavlja problem pri klasifikaciji, saj imajo klasifikacijske metode navadno težave z napovedovanjem slabo zastopanih razredov.

Deljenje glede na enako frekvenco sicer reši ta problem, saj v vsak razred spada enako število učnih primerov (slika 6.2). Vseeno pa tudi ta način ni idealen, saj distribucija razredov slabo odraža porazdelitev podatkov. Obe omenjeni metodi pa imata še en problem: intervale bi radi določili tako, da bo v srednji razred spadal največji delež vzorcev, saj želimo, da ta predstavlja običajno energijsko porabo.

Zaradi tega smo izbrali drugačen pristop. Med vsemi vzorci smo izbrali mediano energijske porabe, ki je enaka 2.75 mWh. To vrednost smo uporabili kot osnovo srednjega intervala. Njegovo širino smo določili s pomočjo središčnega absolutnega odklona ((median absolute error)), ki znaša 1.29 mWh. Za širino intervala smo uporabili dvakratno omenjeno vrednost. V srednji interval torej spadajo vsi učni primeri z energijsko porabo med 1.46 mWh ter 4.04 mWh.



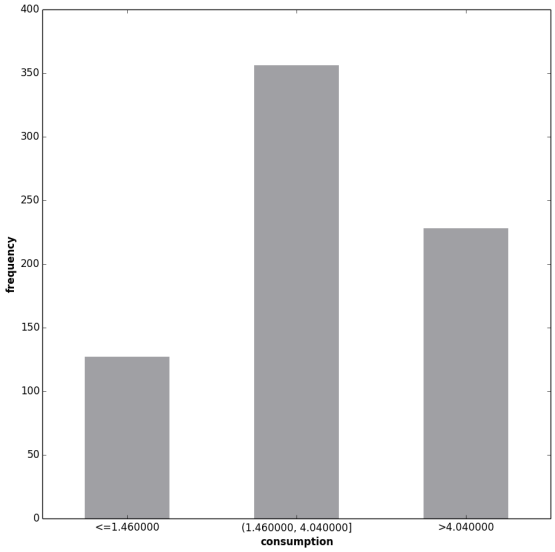
Slika 6.2: Deljenje glede na enako frekvenco.

Razlog, da za določanje intervalov nismo uporabili povprečja in standardnega odklona, je v tem, da morebitni osamelci lahko močno vplivajo na ti dve meri. Mediana ter središčni absolutni odklon sta v tem oziru robustnejši meri [21].

Kot je razvidno s slike 6.3, razred varčnih strani tako predstavlja približno 18%, razred strani z običajno porabo 50%, razred energijsko zahtevnih strani pa 32% učnih primerov.

6.2.2 Diskretizacija zveznih atributov

Temu je sledila diskretizacija vseh ostalih zveznih atributov. Pri tem uporabljamo nadzorovane metode diskretizacije, saj imamo na voljo tudi informacijo o neodvisni spremenljivki. V praksi pri tem uporabljamo enega od dveh požrešnih pristopov: od zgoraj navzdol (*top-down*) in od spodaj navzgor (*bottom-up*). Pri prvemu imamo na začetku en velik interval, v katerem so vsi učni primeri. V vsakem koraku izberemo mejo tako, da izboljšamo



Slika 6.3: Deljenje s pomočjo mediane in središčnega absolutnega odklona.

kvaliteto atributa glede na nek optimizacijski kriterij. Pri tem uporabljamo standardne mere za kakovost atributov. Pristop od spodaj navzdol deluje podobno, le da začnemo s toliko intervali, kot je učnih primerov, in jih nato združujemo [19].

Za optimizacijski kriterij smo uporabili kriterij MDL, predlagan v [22].

6.3 Ocenjevanje kvalitete atributov

Pri gradnji učinkovitega modela za strojno učenje je zelo pomembno določanje kakovosti atributov. Pomembno je, da lahko za vsak atribut ocenimo, kako uporaben je pri določanju ciljne spremenljivke. To nam omogoča, da izločimo tiste atribute, ki k točnosti ocenjevanja ne doprinesejo ničesar [19]. Poleg tega preveliko število nerelevantnih atributov pogosto negativno vpliva na pravilnost klasifikacije pri klasifikacijskih metodah [25].

Za ocenjevanje kakovosti atributov se uporablja več različnih funkcij. Med najbolj pogosto uporabljene spadajo [19]:

- **Informacijski prispevek** temelji na količini informacije in je definiran kot prispevana informacija atributa za določitev vrednosti ciljne spremenljivke. Glavni problem je precenjevanje večvrednostnih atributov.

- **Razmerje informacijskega prispevka** poskuša odpraviti omenjeno pomanjkljivost informacijskega prispevka preko normalizacije z entropijo vrednosti atributa. kar pa pomeni, da precenjuje atribute z zelo majhno entropijo.

- **Mera najkrajšega opisa (MDL)** temelji na kompresivnosti atributa - bolj kot je kompresiven, bolj je pomemben. Izkaže se, da je zelo nepristranska pri ocenjevanju večvrednostnih atributov.

- **Gini-indeks** definira pomembnost atributa kot razliko med apriornim in pričakovanim posteriornim Gini-indeksom. Tudi Gini-indeks precenjuje večvrednostne atribute.

- **ReliefF** je nadgradnja metode Relief. Osnovni algoritem naredi bolj robusten, saj izboljša delovanje pri manjkajočih vrednostih, šumnih podatkih ter večrazrednih problemih.

Problem večine mer za ocenjevanje atributov je v tem, da predpostavljajo medsebojno neodvisnost atributov. Zaradi tega se zlahka zgodi, da sta dva atributa ločeno ocenjena slabo, čeprav je njuna kombinacija zelo pomembna. Ta problem rešuje ReliefF. Kvaliteto atributa oceni glede na lokalne značilnosti razločevanja razredov tako, da za vsak učni primer poišče `k` primerov iz istega razreda ter po `k` primerov iz vsakega od ostalih razredov. Tako implicitno ocenjuje atribute v odvisnosti od ostalih atributov.

Drugi problem je precenjevanje atributov z velikim številm vrednosti. Ta lastnost je še posebej očitna pri informacijske prispevku, velja pa tudi za nekatere ostale mere. Temu se lahko izognemu z binarizacijo atributov, kar pa ni vedno mogoče oziroma zaželjeno. Primera mer, ki ne precenjujeta večvrednostnih atributov, sta mera najkrajšega opisa ter ReliefF.

V našem primeru se za najprimernejšo izbiro mere izkaže ReliefF, saj smo na podlagi domenskega znanja ocenili, da so nekateri atributi v naši učni množici med seboj zelo odvisni. Poleg tega gre v našem primeru za večrazredni problem, prav tako pa večine atributov ne želimo binarizirati, saj bi to pomenilo preveliko izgubo informacij. To sta dodatna razloga v prid meri ReliefF, saj bi mere, kot je na primer informacijski prispevek, atribute lahko ocenile pristransko.

6.3.1 Izbira podmnožice atributov

Kot rečeno, nam ocene atributov pomagajo pri določanju atributov, ki jih je smiselno izločiti iz učne množice. Osnovni in tudi najenostavnejši pristop je metoda filtriranja atributov. Pri njej z izbrano ocenitveno funkcijo ocenimo vse atribute ter glede na rezultate izberemo najboljše. To lahko storimo tako, da vnaprej določimo prag kvalitete atributov, ali pa vnaprej določimo število najboljših atributov, ki jih bomo ohranili [19].

Omenjeni pristop je sicer hiter, vendar pa ima tudi svoje pomanjkljivosti. Kot smo omenili, bi bila v našem primeru najprimernejša mera za oceno atributov ReliefF. Problem, ki se pojavi pri tem, je naslednji: Relief (ter podobno RefliefF) ne izloči odvečnih atributov. V domenah, kjer je večina atributov relevantnih, bo Relief izbral vse, čeprav bi za določanje vrednosti ciljne spremenljivke zadostoval le majhen del atributov [26].

Alternativni pristop je uporaba metode notranje optimizacije (*wrapper*). Pri njej preiskujemo prostor podmnožic atributov glede na optimizacijski kriterij, dokler ne najdemo lokalnega optimuma. V tem primeru ta predstavlja optimalno podmnožico atributov. V praksi to pomeni, da na vsakem koraku optimizacije s trenutno izbranimi atributi zgradimo klasifikacijski model po metodi, ki jo določimo ob začetku. Učne primere prav tako na vsakem koraku razdelimo na učno in testno množico. Prvo uporabimo za konstrukcijo modela, drugo pa za preverjanje le-tega. Ker je v primeru velikega števila atributov prostor podmnožic lahko zelo velik, je preverjanje vseh možnosti pogosto preveč dolgotrajno. V tem primeru se poslužujemo hevrističnih me-

tod preiskovanja (na primer požrešno iskanje). Poznamo več pristopov:

- **Iskanje naprej**, kjer prazni množici na vsakem koraku dodajamo atribute, ki maksimizirajo optimizacijski kriterij.
- **Iskanje nazaj** deluje podobno, le da začnemo z množico vseh atributov in jih na vsakem koraku odstranjujemo.
- **Mešano iskanje**, ki je kombinacija prejšnjih dveh pristopov. V tem primeru začnemo z naključno množico atributov.

Glede na to, da sta tako število učnih primerov kot tudi število atributov relativno majhni, je določanje podmnožice atributov z metodo notranje optimizacije izvedljivo. Zaradi tega smo se odločili množico atributov optimizirati na več načinov ter jih med seboj primerjati.

Odločili smo se za naslednja načina:

- izbira atributov glede na oceno ReliefF,
- metoda notranje optimizacije z uporabo ustreznega klasifikacijskega modela kot optimizacijski kriterij.

6.4 Identifikacija osamelcev

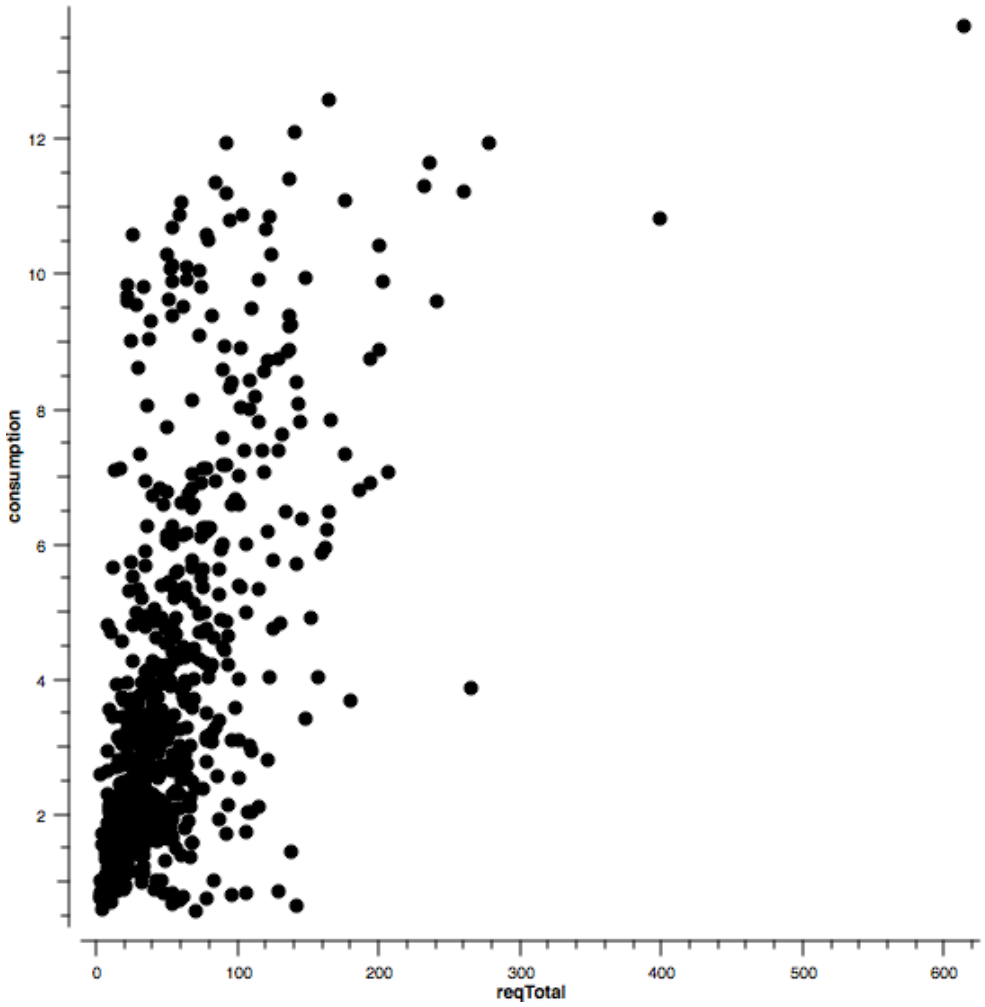
Pri zbiranju množice učnih primerov se v njej pogosto znajdejo osamelci (*outliers*). Osamelec je učni primer, ki se tako močno razlikuje od drugih učnih primerov, da vzbudi sum, da je bil pridobljen na drugačen način [23]. Pojavi se lahko zaradi šuma, ali pa gre dejansko za izjemo. V vsakem primeru so osamelci nezaželjeni, saj slabo vplivajo na klasifikacijsko točnost.

Obstaja več načinov za iskanje osamelcev. Obstaja vrsta metod, ki jih lahko uporabimo za ta namen, na primer metoda `k` najbližjih sosedov, pri kateri učni primer označimo kot osamelca, če v radiju `d` ne obstaja vsaj `p` sosedov [23].

Osamelce lahko iščemo tudi ročno preko vizualizacije podatkov. Slika 6.4 prikazuje primer vizualizacije, kjer se eden od učnih primerov precej razlikuje od ostalih. Treba pa je dodati, da je potrebno osamelce na tak način določati previdno, saj pri tem lahko hitro pride do napake, poleg tega pa je precej zamudno.

Da bi se bolje prepričali v pravilnost našega početja, smo se iskanja osamelcev lotili z uporabo z-vrednosti (*z-score*) [24]. Postopek je sledeč: za vsak učni primer izračunamo povprečno razdaljo do `n` najbližjih sosedov ter jo ovrednotimo z z-vrednostjo. Z-vrednost, večja od 0 pomeni, da je učni primer od svojih sosedov oddaljen bolj od povprečja. Vse primere, ki presegajo vnaprej določeno z-vrednost, označimo kot osamelce.

Za računanje razdalj smo uporabili evklidsko razdaljo, računali pa smo povprečno razdaljo do najbližjih petih sosedov. Kot rezultat smo določili 2 primera, katerih z-vrednost je bila precej višja kot pri ostalih, ter ju izločili iz učne množice.



Slika 6.4: Primer osamelca (zgoraj desno).

Poglavje 7

Klasifikacijske metode

Klasifikacijske metode so metode strojnega učenja, imenovane tudi klasifikatorji, katerih naloga je za določen objekt, opisan z množico atributov, določiti razred, ki mu pripada. Atributi so lahko zvezni ali diskretni, razred pa je v primeru klasifikacijskih metod vedno diskretna spremenljivka [19].

Da bi klasifikatorji lahko pravilno napovedali, kateremu razredu pripada posamezni primer, morajo za to določiti funkcijo, ki podan primer preslika iz prostora atributov v ustrezen razred. Temu procesu pravimo učenje, saj se klasifikatorji te funkcije naučijo iz podanih učnih primerov. Glede na klasifikacijski model je lahko izražena na različne načine, na primer v obliki pravil, odločitvenih dreves ali matematičnih formul [20]. Učenje klasifikatorjev označujemo kot nadzorovano učenje, saj je za vsak učni primer vnaprej znano, kateremu razredu pripada.

7.1 Testiranje klasifikatorjev

Učenje je prvi del procesa. Drugi, enako pomembni del, je preverjanje klasifikacijske točnosti - preveriti moramo, ali klasifikator pravilno razvršča nove primere. Za to uporabimo neodvisno testno množico. Rezultati bi v primeru testiranja na učni množici najverjetneje bili preveč optimistični. Razlog je v tem, da se v nekaterih primerih klasifikator preveč prilagodi učni množici

ter s tem morebitnim anomalijam, ki so v njej prisotne, niso pa prisotne v splošni množici primerov [20]. To pomeni, da je pri klasifikacijskih metodah preverjanje klasifikacijske točnosti z učno množici zelo nezanesljivo.

Kako pridemo do testne množice podatkov? Če imamo na voljo veliko množico podatkov, jo lahko brez težav razdelimo na dva neodvisna dela: učno množico ter testno množico. Pri tem moramo paziti, da je frekvenčna porazdelitev razredov v obeh množicah čim bolj podobna. Temu procesu pravimo stratifikacija (*stratification*).

Žal pa vzorcev pogosto ne želimo deliti, če imamo na voljo le omejeno število vzorcev, saj bomo z dodatnim drobljenjem že sicer majhne množice izgubili dragocene učne primere. Za rešitev tega problema lahko uporabimo enega od naslednjih pristopov [19, 25]:

- **Prečno preverjanje**: množico testnih primerov razdelimo na `K` približno enako močnih podmnožic, pri čemer se za ohranitev distribucije lahko poslužujemo stratifikacije za večjo zanesljivost. Nato klasifikacijski model zgradimo `K`-krat, pri čemer pri vsaki ponovitvi eno podmnožico uporabimo za testiranje, ostale pa za učenje. Najpogostejša vrednost `K` je 10 - empirično je bilo pokazano, da je takšna delitev v splošnem zelo uspešna pri ocenjevanju uspešnosti klasifikacijske metode.

- **Izloči enega (*leave-one-out*)**: posebna oblika prečnega preverjanja, kjer je `K` enak številu primerov v množici. Glavna prednost te metode je, da za učenje uporabimo čim bolj veliko število vzorcev. Poleg tega pa je ocena napake bolj natančna, vendar pa je metoda računsko veliko bolj zahtevna, še posebej pri velikih množicah podatkov, kar pogosto preprečuje njeno uporabo.

- **Razmnoževanje učnih primerov (*bootstraping*)**: to metodo lahko uporabimo v primeru zelo majhne množice podatkov, kjer si ne moremo privoščiti, da za učenje ne bi uporabili vseh primerov. Poteka tako, da iz množice moči `N` naključno izberemo `N` učnih primerov, pri čemer lahko

iste primere izberemo večkrat. Tako v povprečju iz prvotne množice izberemo 63,2% unikatnih primerov, s katerimi zgradimo model. Model nato testiramo na preostalih 36,8% primerov. Ta postopek ponovimo 200-krat.

V našem primeru smo za preverjanje klasifikatorjev uporabili prečno preverjanje, saj smo z njo napako ocenili dovolj uspešno. Zaradi tega preverjanje po metodi izloči enega ni bilo potrebno, prav tako pa ne razmnoževanje učnih primerov, saj smo teh imeli dovolj.

7.2 Pregled klasifikacijskih metod

Skozi čas je bilo na področju strojnega učenja razvito precejšnje število klasifikacijskih metod. Ločimo jih glede na način predstavitve njihove funkcije [19]. Odločili smo se, da bomo pri klasifikaciji spletnih strani glede na porabo primerjali več klasifikatorjev ter tako ocenili, kateri je najbolj primeren. Klasifikatorji, ki smo jih pri tem uporabili, so opisani v nadaljevanju.

7.2.1 Naivni Bayesov Klasifikator

Bayesov klasifikator temelji na Bayesovm pravilu. Računa pogojne verjetnosti za vsak razred pri danih verjetnostih vseh atributov za dani nov primer, ki ga bi radi klasificirali. Žal ne poznamo Bayesovega klasifikatorja, ki bi lahko pogojne verjetnosti natančno izračunal. Naivni Bayesov klasifikator zato računa približke teh verjetnosti s predpostavko, da so pri danem razredu atributi med seboj pogojno neodvisni [19, 25]. Implementacije naivnega Bayesovega klasifikatorja ponavadi predpostavljajo diskretne atribute, kar pomeni, da je potrebno morebitne zvezne atribute vnaprej diskretizirati [19].

Kljub na videz omejujočim predpostavkam se naivni Bayesov klasifikator vsaj na nekaterih vrstah problemov obnese zelo dobro. Še posebno učinkovit

je v kombinaciji z enim od načinov za izbiranje podmnožice atributov, ki vnaprej odstrani so-odvisne (torej odvečne) atribute [25].

7.2.2 Odločitveno drevo

Odločitveno drevo je eden od najbolj pogosto uporabljanih algoritmov za strojno učenje. Delujejo tako, da glede na nek optimizacijski kriterij izbirajo atribute in podmnožice njihovih vrednosti ter na tak način gradijo drevesno strukturo [19]. Za optimizacijski kriterij lahko uporabimo katero od standardnih mer za ocenjevanje kakovosti atributov. V vsakem vozlišču drevesa je pravilo, na podlagi katerega izberemo enega od podreves. Listi drevesa predstavljajo konkretne razrede. Če želimo klasificirati nov primer, moramo glede na vrednosti njegovih atributov slediti pravilom do lista.

Za uporabo z odločitvenimi drevesi morajo biti vsi atributi diskretni, zato je potrebno zvezne atribute najprej diskretizirati. Zelo pomembno je tudi rezanje. To je postopek, pri katerem odstranimo določene poddrevesa, s čimer zagotovimo večjo klasifikacijsko točnost. Razlog je preveliko prilagajanje učni množici, ker vozliščem na nižjih nivojih poddrevesa ustreza premajhno število učnih primerov, da bi bila napoved v vsplošnem še zanesljiva.

Dobra lastnost odločitvenih dreves je, da je mogoče njihove odločitve zelo enostavno pojasniti, kar nam lahko pomaga odkriti, kateri so glavni razlogi za večjo porabo pri nekaterih spletnih straneh.

7.2.3 `k` najbližjih sosedov

Algoritem za klasifikacijo primerov uporablja kar celo učno množico. Ko je potrebno klasificirati nov primer, poišče `k` najbolj podobnih oziroma najbližjih primerov. Novemu primeru se pripiše razred, ki med njegovimi sosedi prevladuje [19].

Pri uporabi moramo atribute ustrezno pripraviti. Za pravilno računanje razdalj med instancami morajo biti zvezni atributi normalizirani, diskretni pa morajo imeti definirano matriko razdalj. Primer enostavne možnosti je,

da je v primeru enake vrednosti diskretnega atributa pri dveh primerih njuna razdalja definirana kot 0, v nasprotnem primeru pa kot 1 [25].

7.2.4 Metoda podpornih vektorjev

Gre za algoritem, ki se lahko uporablja za klasifikacijo tako linearnih kot tudi nelinearnih podatkov. Deluje tako, da množico atributov z jedrnimi funkcijami (`kernel functions`) pretvori v potencialno veliko množico novih atributov. Tam poišče koeficiente diskriminatorne funkcije, ki predstavlja hiperploskev med dvema razredoma [19]. Hiperploskev določa tako, da poskuša maksimizirati razdaljo od nje do najbližjih učnih primerov, ki jim zaradi te njihove lastnosti pravimo tudi podporni vektorji.

Ker lahko hiperploskev definiramo le v zveznem prostoru, morajo biti vsi atributi zvezni. V osnovi metoda podpornih vektorjev ločuje le med dvema razredoma. Če imamo razredov več, potrebujemo hiperploskev za vsak par razredov [19].

Metoda velja za zelo robustno kljub šumnim ali manjkajočim podatkom [28].

7.2.5 Naključni gozd

Ideja naključnih gozdov je, da namesto enega zgradimo večje število odločitvenih dreves, vendar s to razliko, da v vsakem vozlišču najbolj primernega atributa ne izbiramo med vsemi, temveč le med majhno naključno podmnožico. Noveme primere nato klasificiramo tako, da ga klasificira vsako od naključnih dreves, nato pa končni razred določijo z glasovanjem.

V praksi se izkaže, da naključni gozdovi dosegajo veliko natačnost napovedovanja. Vendar pa imajo tudi slabost - njihove napovedi so praktično nerazložljive [19].

Poglavje 8

Rezultati

Kot smo že omenili v poglavju 7.2, smo uporabili pet algoritmov strojnega učenja, za primerjavo pa smo vključili tudi večinski klasifikator.

8.1 Uporabljena programska oprema

Za gradnjo klasifikacijskih modelov smo uporabili program Weka 3 [29]. To je zbirka algoritmov za strojno učenje, ki jih lahko uporabimo za podatkovno rudarjenje. Program Weka je napisan v Javi, algoritme pa lahko apliciramo neposredno na množico učnih primerov s pomočjo grafičnega vmesnika, ali pa jih uporabimo v svojih lastnih Java aplikacijah.

8.1.1 Diskretizacija

Pri naivnem Bayesovem klasifikatorju, odločitvenih drevesih in naključnih gozdovih imamo glede zveznih atributov dve možnosti: lahko jih vnaprej diskretiziramo (kot smo opisali v poglavju 6.2.2), lahko pa to prepustimo algoritmu za gradnjo modela. Odločili smo se testirati obe možnosti.

Oba omenjena modela smo obravnavali na dva načina:

- Z diskretiziranimi zveznimi atributi.
- Z nespremenjenimi zveznimi atributi.

V primeru vnaprejšnje diskretizacije smo uporabili razred `weka.filters.supervised.attribute.Discretize`. Pri tem smo uporabili kriterij MDL, definiran v [30].

Ostali dve metodi bolje delujeta z zveznimi metodami, zato tam diskretizacije nismo uporabili.

8.1.2 Izbira atributov

V vseh primerih smo za izbiro atributov uporabili razred `weka.filters.supervised.attribute.AttributeSelection`, in sicer na dva načina:

- Z uporabo mere za ocenjevanje atributov ReliefF, pri čemer smo uporabili 10 najbolje ocenjenih atributov, saj so bili ti atributi ocenjeni občutno bolje kot ostali. To smo storili z razredom `weka.attributeSelection.ReliefFAttributeEval` v kombinaciji z `weka.attributeSelection.Ranker`.

- Z metodo notranje optimizacije, ki smo jo izvedli za vsak klasifikacijski model posebej. Uporabili smo razred `weka.attributeSelection.WrapperSubsetEval`, za preiskovanje prostora podmnožic atributov pa `weka.attributeSelection.BestFirst`.

Tabela 8.1 prikazuje 10 atributov, ki jih je mera ReliefF ocenila kot najboljše. Pri vsakem atributu je dodano tudi razmerje informacijskega prispevka.

8.1.3 Uporabljene metode

Za klasifikacijo smo uporabili naslednje razrede:

- **Naivni Bayesov klasifikator**: `weka.classifiers.bayes.NaiveBayes`. Pri gradnji smo uporabili opcijo `-D` (nadzorovana diskretizacija zveznih atributov).

Ime atributa	ReliefF	Razmerje inf. prispevka
fullyLoaded	0.0261	0.146
onLoad	0.0228	0.1327
numDomains	0.0223	0.1441
bytesJS	0.0182	0.1556
reqJS	0.0145	0.1738
reqTotal	0.0123	0.1755
numCompressed	0.0118	0.1696
visualComplete	0.0116	0.1173
bytesHtml	0.0106	0.1341
bytesTotal	0.0101	0.183

Tabela 8.1: Atributi, ki jih je ocena ReliefF ocenila kot najboljše.

- **Odločitveno drevo**: `weka.classifiers.trees.J48`. Gre za implementacijo algoritma C4.5. Faktor zaupanja (*confidence factor*) smo nastavili na 0.25, minimalno število instanc v listih pa na 25.
- **Naključni gozd**: `weka.classifiers.trees.RandomForest`. Gradili smo gozd z 200 drevesi, katerim nismo omejevali maksimalne globine.
- `k` **najbližjih sosedov**: `weka.classifiers.lazy.IBk`. Število upoštevanih najbližjih sosedov smo nastavili na 20. Pri računanju smo uporabili evklidsko razdaljo brez uteževanja.
- **Metoda podpornih vektorjev**: `weka.classifiers.functions.LibSVM`. Uporabili smo jedro RBF, prilagodili pa smo dva parametra: ceno (*cost*) smo nastavili na 140, parameter γ pa na 0.1.

Parametre smo optimizirali z interno validacijo.

8.2 Primerjava algoritmov

Primerjava klasifikacijske točnosti vseh metod je razvidna iz tabele 8.2.

Metoda	Izbira atr.	% Pravilno klas. primerov	AUC
M	/	50.07	0.500
NB	ReliefF	66.81	0.812
NB+D	ReliefF	67.93	0.821
NB	Wrapper	69.90	0.808
NB+D	Wrapper	70.61	0.81
DT	ReliefF	67.79	0.793
DT+D.	ReliefF	67.65	0.768
DT	Wrapper	68.50	0.765
DT+D	Wrapper	68.07	0.776
RF	ReliefF	68.21	0.82
RF+D	ReliefF	65.68	0.808
RF	Wrapper	67.65	0.815
RF+D	Wrapper	69.48	0.783
kNN	ReliefF	68.21	0.817
kNN	Wrapper	68.07	0.774
SVM	ReliefF	69.2	0.73
SVM	Wrapper	69.06	0.734

Tabela 8.2: Primerjava klasifikacijskih metod. M - večinski klasifikator, NB - naivni Bayesov klasifikator, DT - odločitveno drevo, NG - naključni gozd, kNN - `k` najbližjih sosedov, SVM - metoda podpornih vektorjev, +D - diskretizirani zvezni atributi.

Kot lahko vidimo, so vse metode podobno uspešne pri klasifikaciji. Iz povprečja nekoliko izstopa naivni Bayesov klasifikator, še posebej z v splošnem zelo visoko vrednostjo AUC. To pomeni, da bo izmed vseh metod z največjo verjetnostjo pravilno razločil med pozitivnim in negativnim primerom, zato bi lahko rekli, da je naivni Bayesov klasifikator v našem primeru najboljši med preizkušenimi metodami [19]. Še enkrat pa je potrebno poudariti, da so razlike zelo majhne.

Pri odločitvenem drevesu je razvidno, da k boljšemu rezultatu pripomore izbira atributov z metodo notranje optimizacije, saj je bila klasifikacijska točnost v tem primeru nekoliko boljša. V kombinaciji z diskretiziranimi zveznimi spremenljivkami se je izboljšala tudi vrednost AUC.

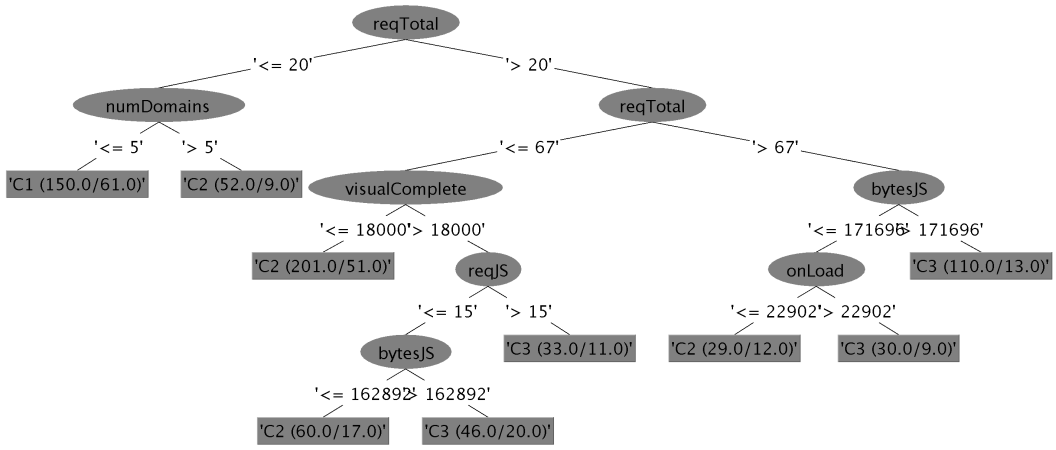
Preostali dve metodi (metoda `k` najbližjih sosedov ter metoda podpornih vektorjev) sta dosegli relativno visoko klasifikacijsko točnost, vendar pa tudi precej nizko vrednost AUC. Edina izjema pri tem je metoda `k` najbližjih sosedov v kombinaciji z izbiro atributov z mero ReliefF.

Seveda pa je očitno, da z nobenim klasifikatorjem nismo dosegli posebno dobre napovedne točnosti. Razloge za to gre iskati predvsem v dejstvu, da gre za zelo dinamično domeno, kjer je zbiranje kvalitetnih podatkov izredno težko. To posledično pomeni, da podatki vsebujejo šum, ki opazno vpliva na klasifikacijsko točnost.

Kljub vsemu pa smo pokazali, da je tudi z relativno preprostimi metodami zbiranja vzorcev mogoče precej dobro oceniti energijsko porabo spletne strani.

8.2.1 Analiza odločitvenega drevesa

Kljub trditvi, da se je naivni Bayesov klasifikator obnesel najbolje, je za nas vseeno bolj zanimivo odločitveno drevo. Radi bi namreč ugotovili, kateri atributi najbolj vplivajo na visoko porabo. Odločitveno drevo nam omogoča na enostaven način razumeti, kateri atributi so v največji meri vplivali na klasifikacijo. Slika 8.1 prikazuje drevo, zgrajeno pri izbiri atributov z mero ReliefF.



Slika 8.1: Graf odločitvenega drevesa.

Veliko razkrije tudi tabela 8.3, ki prikazuje klasifikacijske točnosti ter AUC vrednosti po razredih.

Vidimo lahko, da je klasifikacijska točnost pri vseh razredih približno enako dobra. Pri nobenem razredu sicer ni zelo zelo visoka. Vseeno pa to pomeni, da klasifikator ne daje prednosti specifičnemu razredu, kar je zaželjeno.

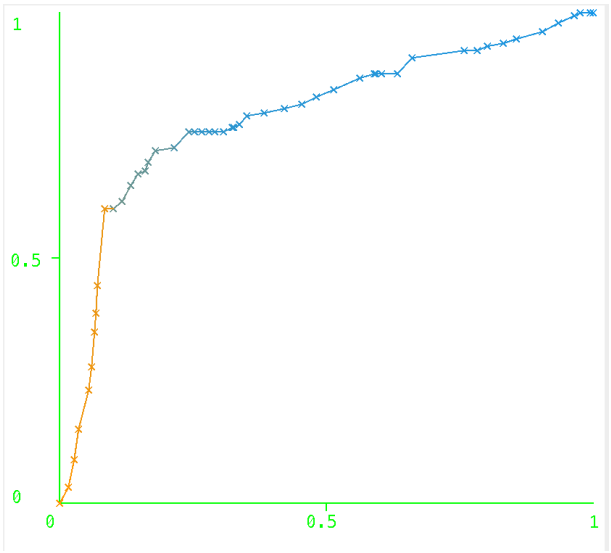
Tabela 8.4 prikazuje tabelo napačnih klasifikacij (*confusion matrix*), iz

Razred	TP	FP	Preciznost	Priklic	AUC
C1	0.661	0.106	0.603	0.598	0.81
C2	0.688	0.29	0.692	0.733	0.75
C3	0.671	0.133	0.721	0.658	0.85
Uteženo povprečje	0.678	0.207	0.686	0.685	0.793

Tabela 8.3: Primerjava klasifikacijskih točnosti pri odločitvenem drevesu po razredih. C1 - varčne strani, C2 - povprečne strani, C3 - požrešne strani.

Klasificiran kot →	C1	C2	C3
C1	84	32	11
C2	58	245	53
C3	4	71	153

Tabela 8.4: Tabela napačnih klasifikacij za odločitveno drevo. C1 - varčne strani, C2 - povprečne strani, C3 - požrešne strani.



Slika 8.2: Krivulja ROC za razred varčnih strani.

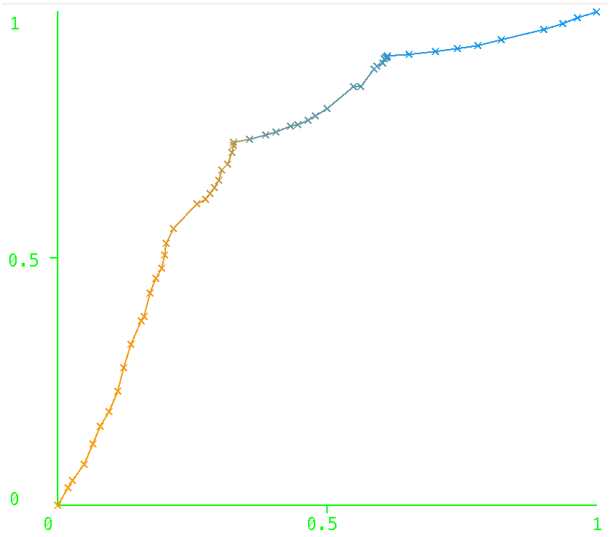
katere je razvidno, da so primeri večinoma klasificirani pravilno.

Slike 8.2, 8.3 in 8.4 prikazujejo krivulje ROC za posamezne razrede.

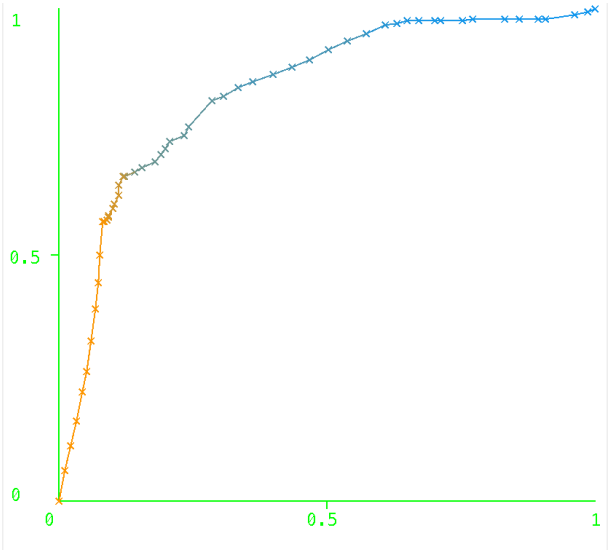
8.3 Razlogi za visoko porabo

Kot smo pojasnili v poglavju 8.2.1, je odločitveno drevo edina metoda, pri kateri lahko približno ocenimo, kateri atributi imajo na klasifikacijo največji vpliv. V ta namen smo analizirali graf na sliki 8.1.

Pri takšnem načinu analize se izkaže, da imajo največji vpliv na porabo



Slika 8.3: Krivulja ROC za razred povprečnih strani.



Slika 8.4: Krivulja ROC za razred požrešnih strani.

naslednji atributi:

- **Skupno število zahtevkov**.
- **Čas nalaganja strani** je očitno močno koreliran z energijsko porabo. Pri tem ocenjujemo, da daljši čas ni vzrok za večjo porabo, ampak da je oboje posledica nekega drugega vzroka.
- **Skupno število zahtevkov za JavaScript** in **skupna količina prenesenega JavaScripta**. Pri tem lahko špekuliramo, da na porabo baterije JavaScript vpliva predvsem zato, ker se mora na telefonu izvajati.
- **Število domen, na katere se stran povezuje**. Spletna stran lahko naloži resurse iz različnih domen. Zanimivo pri tem je, da nalaganje iz čim večjega števila domen velja za dobro prakso, s katero naj bi se povečala hitrost nalaganja. Razlog je v tem, da mnogi brskalniki omejujejo število sočasnih povezav na isto domeno, čemur se tako lahko izognemo.

Žal nam naša metodologija ne omogoča, da bi razloge za porabo določili bolj natančno.

8.4 Vpliv na okolje

Optimizacija spletnih strani za manjšo porabo energije je zaželjena iz dveh razlogov. Prvi je, kot že omenjeno, podaljšanje avtonomije baterije. Drugi razlog, za katerega bi lahko rekli, da je še bolj pomemben, pa je prihranek energije na svetovnem nivoju. Ta ima dobre ekološke posledice, saj z manjšo porabo energije zmanjšamo tudi izpust ogljikovega dioksida.

Zanima nas torej, kakšen vpliv ima optimizacija konkretne potratne strani na okolje v svetovnem merilu. Za izračun smo uporabili spletno stran amazon.com. Trenutno se ocenjuje, da gre za osmo najbolj obiskano spletno stran. Ker točno število ogledov strani (*page views*) ni znano, smo to v grobem ocenili s pomočjo podatkov, dobljenih v [31, 32]. Ti podatki kažejo, da

amazon.com na mesec obišče povprečno 164 milijonov obiskovalcev. Vsak obiskovalec naj bi si v povprečju ogledal okoli 10 strani, torej ocenjujemo, da se na amazon.com mesečno zgodi okoli milijarda in pol ogledov strani.

Upoštevati je treba tudi, da so to podatki za vse obiske na vseh napravah. Delež prometa z mobilnih in tabličnih naprav se ocenjuje na okoli 31% vsega prometa [34], kar je v tem primeru potrebno upoštevati pri ocenah. To pomeni, da lahko za izračun uporabimo oceno, da se je na amazonu zgodilo 465 milijonov ogledov strani.

V naših eksperimentih je amazon.com s porabo 9.4 mWh spadal v razred požrešnih strani. Če bi stran optimizirali tako, da bi se lahko uvrstila v srednji razred, bi ji morali porabo zmanjšati za vsaj 5.4 mWh. Če to pomnožimo s številom ogledov strani, bi tako skupaj prihranili okrog 2.5 MWh energije, kar ustreza približno 1.55 sodčkom nafte.

Če bi optimizirali tako, da bi stran padla v razred varčnih strani, bi ob vsakem ogledu v primerjavi s trenutnim stanjem prihranili 8 mWh. Preračunano to pomeni 3.72 MWh ali 2.33 sodčkov nafte.

Še bolj nas zanima ocena prihranka, če upoštevamo vse strani. Po metodi, opisani v [33], je bilo leta 2012 v grobem ocenjeno, da se mesečno na internetu zgodi 2300 milijard ogledov strani. Število mobilnih ogledov strani torej lahko ocenimo na 713 milijard. Če upoštevamo povprečno vrednost porabe energije, določeno v naši raziskavi, ki znaša 3.71 mWh, lahko izračunamo, da bi povprečna stran z optimizacijo lahko prihranila 2.25 mWh. Torej, mesečno bi tako prihranili več kot 1.55 GWh energije, letno pa kar 18.6 GWh, kar ustreza skoraj 12400 sodčkom nafte.

Pri tem je treba poudariti, da so to zelo grobe ocene, vseeno pa nam nudijo približen vpogled v stanje.

Poglavje 9

Sklepne ugotovitve

V svojem diplomskem delu smo preverili, ali je mogoče z metodami strojnega učenja poljubno spletno stran uvrstiti v ustrezen energijski razred glede na porabo energije na mobilnem telefonu. V ta namen smo razvili merilni sistem ter z njim zbrali množico učnih primerov, ki smo jo zatem uporabili za učenje klasifikatorjev. Pokazali smo, da izvedba sicer ni enostavna, je pa mogoča.

Te ugotovitve po našem mnenju predstavljajo pomemben korak v pravo smer. Dandanes se namreč vse bolj zavedamo, da je energija dragocena. To velja tudi za energijo, ki poganja naše mobilne telefone, in sicer iz več zornih kotov: dragocena je za uporabnika, ki od svoje naprave pričakuje čim večjo avtonomijo, in dragocena s stališča okolja. Morda se zdi poraba posamezne spletne strani zanemarljiva, toda ob upoštevanju velikega števila uporabnikov mobilnega interneta hitro vidimo, da temu ni tako.

Z našo raziskavo smo pokazali, da so med spletnimi stranmi velike razlike, predvsem pa, da je z metodami strojnega učenja možno vsaj približno oceniti, kako potratna je spletna stran. Z razvojem teh metod bi lahko izboljšali točnost ocen in tako razvijalcem ponudili orodje, ki bi jim omogočilo optimizacijo.

Analizirali smo tudi vpliv prihranka energije na okolje. To je sicer precej težko, saj je zelo težko oceniti natančno število obiskov spletne strani. Vseeno pa je očitno, da prihranki že pri eni sami strani niso zanemarljivi. Če bi lahko

optimizirali vse ali pa vsaj večino spletnih strani, bi lahko najverjetneje precej zmanjšali skupno porabo energije.

9.1 Nadaljnje delo

Da bi to dosegli, bi bilo potrebno implementirati nekatere izboljšave:

- Bolj natančno zajemanje podatkov o porabi. Ena od omejitev pri naši metodi je omejevanje nalaganja strani na 60 sekund, čeprav se nekatere strani naložijo veliko prej, nekatere pa potrebujejo še več časa.
- Boljši podatki o straneh. Idealno bi bilo zbiranje strukturnih podatkov o straneh med dejanskim merjenjem porabe, ali pa vsaj na istem telefonu in na isti geografski lokaciji.
- Večkratne ponovitve meritev, saj bi tako lahko zelo zmanjšali vpliv šuma.
- Brezhiben avtomatski sistem, s katerim bi lahko zbirali vzorce poljubno dolgo in tako zbrali veliko večjo množico učnih primerov.
- Analiza strani na podlagi sej - za mnoga spletna mesta obisk naslovne strani ni zelo reprezentativen.
- Pri meritvah bi lahko uporabili več različnih naprav, predvsem sodobnejših, ki bi bolje odražale trenutno stanje.

Literatura

[1] UIDevice Class Reference. Dostopno na: https://developer.apple.com/library/ios/documentation/uikit/reference/UIDevice_Class/Reference/UIDevice.html#//apple_ref/occ/instp/UIDevice/batteryLevel. Dostopano: april 2014.

[2] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, J. P. Singh. "Who Killed My Battery?: Analyzing Mobile Browser Energy Consumption", v zborniku: Proceedings of the 21st International Conference on World Wide Web, 2012, str. 41–50.

[3] Battery Manager. Dostopno na: http://developer.android.com/reference/android/os/BatteryManager.html. Dostopano: april 2014.

[4] Multimeter. Dostopno na: http://en.wikipedia.org/wiki/Multimeter. Dostopano: april 2014.

[5] A. Rice, S. Hay. "Decomposing power measurements for mobile devices", v zborniku: Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on, 2010, str. 70–78.

[6] (2013) Agilent 34410A and 34411A Multimeters Data Sheet. Dostopno na: http://cp.literature.agilent.com/litweb/pdf/5989-3738EN.pdf. Dostopano: april 2014.

[7] Digital Multimeter (DMM) Connectivity Utility Software. Dostopno na: http://www.home.agilent.com/en/pd-2307503/digital-multimeter-dmm-connectivity-utility-software. Dostopano: april 2014.

[8] Samsung Galaxy Ace Tech Specs. Dostopno na: http://www.samsung.com/galaxyace/ace_techspec.html. Dostopano: april 2014.

[9] Lithium-ion Battery. Dostopno na: http://en.wikipedia.org/wiki/Lithium-ion_battery. Dostopano: april 2014.

[10] Utility Frequency. Dostopno na: http://en.wikipedia.org/wiki/Utility_frequency. Dostopano: april 2014.

[11] What is Number of Power Line Cycle(NPLC) on a digital multimeter(DMM). Dostopno na: https://www.measuretechnic.com/?p=41. Dostopano: april 2014.

[12] Data Logging and Digitizing Using a Digital Multimeter. Dostopno na: http://cp.literature.agilent.com/litweb/pdf/5990-3220EN.pdf. Dostopano: april 2014.

[13] D. Dary, Selendroid. Dostopno na: http://selendroid.io/. Dostopano: april 2014.

[14] Testing Fundamentals - Instrumentation. Dostopno na: http://developer.android.com/tools/testing/testing_android.html#Instrumentation. Dostopano: april 2014.

[15] M. Lim Screen On Toggler. Dostopno na: https://play.google.com/store/apps/details?id=com.malcolm.screenontoggler. Dostopano: april 2014.

[16] B. Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications)*. Springer, 2011.

[17] HTTP Archive. Dostopno na:
http://httparchive.org/. Dostopano: maj 2014.

[18] WebPagetest. Dostopno na:
http://www.webpagetest.org/. Dostopano: maj 2014.

[19] I. Kononenko, M. Kukar. *Machine Learning and Data Mining*. Woodhead Publishing, 2007.

[20] C. Han, M. Kamber. *Data Mining*. Morgan Kaufmann Publishers Inc., 2006.

[21] Median Absolute Deviation. Dostopno na:
http://en.wikipedia.org/wiki/Median_absolute_deviation. Dostopano: maj 2014.

[22] I. Kononenko. "On Biases in Estimating Multi-valued Attributes", v zborniku: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, 1995, str. 1034–1040.

[23] N. M. Hewahi, M. K. Sadi. "Class Outliers Mining: Distance-Based Approach", v zborniku: International Journal of Intelligent Systems and Technologies, 2007, str. 55–68.

[24] Standard Score. Dostopno na:
http://en.wikipedia.org/wiki/Standard_score. Dostopano: maj 2014.

[25] I. H. Witten, E. Frank, M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques, Third Edition (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2011.

[26] R. Kohavi, G. H. John. "Wrappers for feature subset selection", v zborniku: Artificial Intelligence 97, 1997, str. 273–324.

[27] A. G. Karegowda, M. A. Jayaram, A. S. Manjunath. "Feature Subset Selection Problem using Wrapper Approach in Supervised Learning",

v zborniku: International Journal of Computer Applications, 2010, str. 13–17.

[28] T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, D. Haussler. "Support vector machine classification and validation of cancer tissue samples using microarray expression data", v zborniku: Bioinformatics, 2000, str. 906–914.

[29] Weka. Dostopno na: http://www.cs.waikato.ac.nz/ml/weka/. Dostopano: maj 2014.

[30] I. Kononenko. "On Biases in Estimating Multi-Valued Attributes", v zborniku: 14th International Joint Conference on Articial Intelligence, 1995, str. 1034–1040.

[31] Most Popular Retail Websites In He United States As Of 1st Quarter 2014, Ranked By Visitors. Dostopno na: http://www.statista.com/statistics/271450/monthly-unique-visitors-to-us-retail-websites/. Dostopano: junij 2014.

[32] Site Overview - amazon.com. Dostopno na: http://www.alexa.com/siteinfo/amazon.com. Dostopano: junij 2014.

[33] (2012) Calculate More Accurate Number Of Pageviews Of Any Website. Dostopno na: http://www.creasysolve.com/2012/10/calculate-more-accurate-number-of-page.html. Dostopano: junij 2014.

[34] StatCounter Global Stats - Browser, OS, Search Engine Including Mobile Market Share. Dostopno na: http://gs.statcounter.com/#all-comparison-ww-monthly-201305-201405. Dostopano: junij 2014.