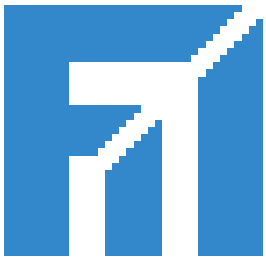
Universitatea ”Alexandru Ioan Cuza” din Iași

Facultatea de Informatică



Lucrare de Licență

PDF parser

Propusă de:

Silvia-Ana-Maria Ungureanu

Coordonator științific:

Lect. dr. Alex Moruz

Sesiunea: iulie, 2019

Universitatea ”Alexandru Ioan Cuza” din Iași

Facultatea de Informatică

PDF parser

Silvia-Ana-Maria Ungureanu

Coordonator științific:

Lect. dr. Alex Moruz

Sesiunea: iulie, 2019

Declarație privind originalitatea și respectarea drepturilor de autor

Prin prezenta declar că Lucrarea de licență cu titlul "PDF parser" este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- Toate fragmentele de text reproduse exact, chiar și în traducere proprie în altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- Reformularea în cuvinte proprii a textelor scrise de catre alți autori deține referința precisă a sursei;
- Codul sursă, imaginile etc. preluate din proiecte open-source sau alte surse, sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- Rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, iulie 2019

Absolvent

Silvia-Ana-Maria Ungureanu

_____

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul "PDF parser", codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea "Alexandru Ioan Cuza" Iași să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, iulie 2019

Absolvent

Silvia-Ana-Maria Ungureanu

_____

Cuprins

Introducere

Headerele și footerele reprezintă elemente de formatare de bază, întâlnite în aproape toate tipurile de documente. Pe lângă aspectul estetic pe care îl dau paginilor, aceste elemente cuprind și informații referitoare la numărul paginii, titlul cărții, autorul, numele și numărul capitolului curent precum și note de subsol. În situația în care se dorește extragerea textului, metodele clasice (Object character recognition) vor lua în calcul inclusiv aceste informații existente în headere și footere, ceea ce ar putea modifica informația transmisă de text.

În domenii precum NLP (Natural language processing) este necesar ca datele să fie extrase, fără a fi luate în calcul elementele regăsite în headere și footere. Deoarece aceste informații nu fac parte din conținutul paginii, în momentul extragerii, ele vor fragmenta si vor schimba sensul mesajelor, ceea ce nu este de dorit în situația în care se incearcă analiza unui text la nivel sintactic și semantic. (1)

Detectarea și eliminarea headerelor și footerelor din fișierele PDF de dimensiuni mari reprezintă o sarcină dificilă, mai ales atunci când utilizatorul este nevoit să apeleze la o metodă manuală de extragere a datelor necesare.

Pe lângă problema dimensiunii documentelor ce doresc a fi parsate, o alta problemă poate fi reprezentată de structura instabilă în ceea ce privește evidențierea headerelor și footerelor în PDF-uri.

Bibliotecile puse la dispoziție de actualele soft-uri sunt greu de gestionat pentru un utilizator obișnuit, iar de cele mai multe ori eficiența lor nu este cea dorită.

Motivație

În contextul în care domeniile care se ocupă de procesarea și de înțelegerea limbajului natural necesită o extragere cât mai corectă a datelor, lucrarea de față propune o soluție prin care să se extragă textul la nivel de frază, fără a fi luate în calcul și meta informațiile.

Ținând cont de problema dimensiunii documentelor cat și de cea a formatului instabil, lucrarea de față dorește să pună la dispoziția utilizatorului o modalitate practică de a elimina

automat headerele, footerele si notele de subsol din documente de tip PDF. Astfel, utilizatorul va avea o interacțiune minimă în ceea ce privește intregul proces de parsare.

Contribuții

Contribuțiile mele constau în punerea la dispoziția utilizatorului a unei aplicații ce procesează textele precum și investigarea si detectarea unor pattern-uri ce definesc prezența headerelor și footerelor într-un text.

Astfel lucrarea pune la dispozitie:

- ✓ O serie de pattern-uri ce ajută la identificarea headerelor si footerelor
- ✓ O listă de expresii regulate ce pune în aplicare pattern-urile identificate anterior
- ✓ Un algoritm care, în baza expresiilor regulate precum si a altor observații de comportament specifice headerelor și footerelor, le identifică și le elimină din text.

Organizare

În primul capitol se face o scurtă introducere în domeniul problemei. Voi discuta despre structura PDF ului, despre diferitele tipuri de headere și footere, precum și despre problema extragerii textului fără a elimina headerele si footerele.

În al doi-lea capitol se prezintă soluții similare, o descriere a funcționalităților puse la dispoziție de biblioteca PDF box, precum și dezavantajele și deficiențele întâlnite la soluțiile actuale.

Următorul capitol cuprinde o descriere detaliată a soluției propuse de mine, însoțită de exemple. Tot în acest capitol se descriu problemele întâmpinate pe parcursul cercetării și implementării soluției.

În capitolul 4 voi prezenta o serie de teste ce au fost efectuate asupra aplicației, în vederea determinării eficienței acesteia și a îmbunătățirii calității programului, inclusiv optimizării din punct de vedere a timpului de execuție.

Ultimul capitol cuprinde o serie de concluzii legate de obiectivele pe care am reușit să le ating, de cele pe care nu le-am atins, precum și o serie de modalități de îmbunătățire a aplicației pe viitor.

1 Problema extragerii textului din PDF

Pentru a înțelege mai bine problema eliminării headerelor și footerelor din PDF-uri voi face o scurtă introducere în ceea ce privește structura acestui tip de fișier.

De asemenea voi evidenția anumite caracteristici specifice diferitelor tipuri de headere și footere prin exemplificare și voi discuta mai pe larg despre impactul pe care îl poate avea extragerea necorespunzatoare a textului asupra sintaxei și semanticii.

1.1 Ce este un PDF și care este structura lui

Un PDF (Portable Document Format) este un tip de fișier dezvoltat de Adobe ce își propune prezentarea documentelor în mod sigur și independent față de componentele hard, soft și de sistemul de operare. Fișierele de tip PDF pot conține elemente audio, video, link-uri. (2)

Un PDF simplu, după cum se poate observa și în Fig. 1 conține 4 părți:

- Header - reprezintă prima linie dintr-un PDF și conține informații despre versiunile utilizate.
- Body - cuprinde toate obiectele (media, imagini, text) ce se regăsesc în document și sunt afișate utilizatorului.
- Tabel de referințe – salvează referințe către toate obiectele existente în fișier. Acest lucru facilitează accesarea independentă a fiecărui obiect.
- Trailer – specifică modul în care aplicația care citește documentul poate accesa obiectele și tabelul de referințe.

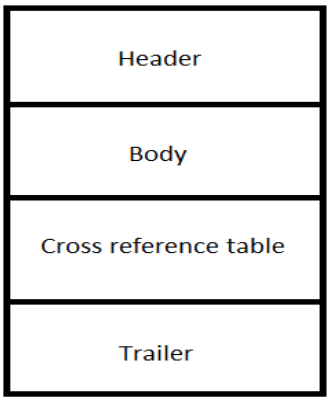


Fig. 1 Structura PDF

De asemenea este important de precizat faptul că orice fișier PDF se va termina cu stringul *%%EOF* (end of file). (3)

Din câte se poate observa, acest tip de document nu are o structura stabilă, un standard definit în ceea ce priveste modalitățile de evidențiere a componentelor regăsite în headere și footere de cele din conținutul efectiv al paginii. De aceea idetificarea lor pe baza structurii PDF-ului reprezintă o sarcină dificilă.

1.2 Tipuri de Headere și Footere

Desigur, există un număr mare de varietăți în ceea ce privește structura unui header sau a unui footer. Majoritatea headerelor pot conține următoarele tipuri de informații:

- Numărul paginii
- Titlul cărții
- Titlul cărții însoțit de numărul paginii
- Numele autorului
- Numele autorului însoțit de numărul paginii
- Titlul capitolului
- Titlul capitolului însoțit de numărul paginii

În ceea ce privește footerele, nu se pot extrage multe informatii întrucât structura acestora este foarte specifică. Astfel despre footere se poate spune că pot conține următoarele elemente:

- Numărul paginii
- Note de subsol

1.3 Sitaxa și semantica textului

Deoarece actualele aplicații open-source de procesare și extragere a textului din PDF –uri nu pun la dispoziție o modalitate de eliminare a headerelor și footerelor, acestea sunt extrase și interpretate ca făcând parte din corpul documentului.

Fig. 2 prezintă rezultatul parsării clasice al unui document PDF, utilizând biblioteca PDF box. Desigur, se poate observa cu ușurință că rândul subliniat reprezintă un header și că a fost și el extras odată cu întreg conținutul paginii.

În cazul în care se dorește analiza textului la nivel de frază, rândul subliniat va fi considerat ca făcând parte din ea, întrucât nu se regăsește în rândul anterior și nici în header un semn de punctuație, ce ar putea marca sfarșitul unei fraze.

Astfel, fraza extrasă va fi: *„The project managers look to us to help work out the schedule. We* ***6 Chapter 1: Clean Code*** *are deeply complicit in the planning of the project and share a great deal of the responsi- bility for any failures;”* în loc de *„The project managers look to us to help work out the schedule. We are deeply complicit in the planning of the project and share a great deal of the responsi- bility for any failures;”*

and telephone sanitizers. But the fault, dear Dilbert, is not in our stars, but in ourselves. We are unprofessional.

This may be a bitter pill to swallow. How could this mess be our fault? What about the requirements? What about the schedule? What about the stupid managers and the useless marketing types? Don't they bear some of the blame?

No. The managers and marketers look to us for the information they need to make promises and commitments; and even when they don't look to us, we should not be shy about telling them what we think. The users look to us to validate the way the requirements will fit into the system. The project managers look to us to help work out the schedule. We

are deeply complicit in the planning of the project and share a great deal of the responsibility for any failures; especially if those failures have to do with bad code!

"But wait!" you say. "If I don't do what my manager says, I'll be fired." Probably not. Most managers want the truth, even when they don't act like it. Most managers want good code, even when they are obsessing about the schedule. They may defend the schedule and requirements with passion; but that's their job. It's your job to defend the code with equal passion.

To drive this point home, what if you were a doctor and had a patient who demanded that you stop all the silly hand-washing in preparation for surgery because it was taking too much time?2 Clearly the patient is the boss; and yet the doctor should absolutely refuse to comply. Why? Because the doctor knows more than the patient about the risks of disease and infection. It would be unprofessional (never mind criminal) for the doctor to comply with the patient.

So too it is unprofessional for programmers to bend to the will of managers who don't understand the risks of making messes.

Fig. 2 – text extras cu PDF box

2 Soluții existente

Întrucât am decis să implementez aplicația în Java, aveam nevoie de o bibliotecă care să imi puna la dispozitie o serie de metode prin care să pot procesa documentele și să pot extrage doar *body*-ul din pagini.

Există mai multe biblioteci *open-source* implementate în Java, printre care cele mai importante sunt: PDFBox, iText, PDF JDK, PDF Clown.

2.1 PDFBox

Apache PDFBox a început dezvoltarea în anul 2002, în cadrul companiei Source Forge de către Ben Litchfield. Apache PDFBox este o bibliotecă open-source specifica limbajului Java care ofera utilizatorului un set de resurse ce își au ca scop crearea, manipularea, și conversia documentelor de tip PDF. (4)

O caracteristică importantă a bibliotecii menționate anterior este că extrage rapid textul dintr-o varietate mare de documente PDF. Apache PDFBox mai include și utilități pentru linia de comandă.

PDFBox poate fi împărțit în 4 sectiuni de bază:

- PDFBox – cuprinde clase de extragere și manipulare a datelor
- FontBox – cuprinde clase ce permit utilizatorului să modifice și să adauge fonturi noi
- XmpBox – cuprinde clase ce organizează metadatele XMP
- Preflight – verifică dacă fișierele corespund standardelor PDF

Întrucât în realizarea aplicației nu am avut nevoie decât de manipularea și extragerea datelor existente în acest tip de document, voi face o descriere succintă a componentei PDFBox.

Extragerea textului este una din funcționalitățile principale din PDFBox și se poate realiza prin două metode:

1. Folosind metoda getText() din clasa PDFTextStripper

2. Declarând o regiune de dimensiune dreptunghiulară folosind metoda AddRegions() și extragând textul cu ajutorul metodei extractRegions(). Amele metode se regăsesc în clasa TextStrippedByArea.

De asemenea este important de menționat faptul că biblioteca permite extragerea datelor dintr-o aumită pagina (specificând numărul acesteia), sau din mai multe pagini, specificând intervalul (pagina inițială și cea finală).

Mai multe detalii despre aceste două metode de extragere a textului cât și despre impactul și rolul pe care l-au avut în dezvoltarea aplicației voi explica în capitolele ce vor urma.

Un alt tool similar cu PDFBox, care este foarte popular în randul utilizatorilor este iText. Spre deosebire de PDFBox, iText pune la dispoziție un API.

Funcționalitățile puse la dispoziție sunt asemănatoare cu cele regăsite in PDFBox. Diferă însă modul în care sunt citite datele din PDF. Dacă PDFBox citește caracter cu caracter, iText ia în considerare mai multe caractere odată, ceea ce scade semnificativ timpul de procesare al documentului.

3 Aplicația

În acest capitol voi discuta despre modul în care a fost abordată problema, precum și despre pașii care au contribuit la realizarea solutiei finale. Desigur, voi prezenta și o serie de probleme ce au fost întâmpinate în parcursul dezvoltării acestei aplicații și modul în care au fost soluționate acestea.

3.1 Abordări inițiale

Initial am luat în cosiderare o abordare bazată pe OCR (Object Character Recognition) prin intermediul căreia să parcurg intreg documentul și cu ajutorul unor algoritmi de machine learning să pot să identific fiecare caracter și să îl scriu într-un fisier de tip txt. Desigur că această tehnica implica o complexitate mare din punct de vedere al impului necesar pentru efectuarea cercetărilor.

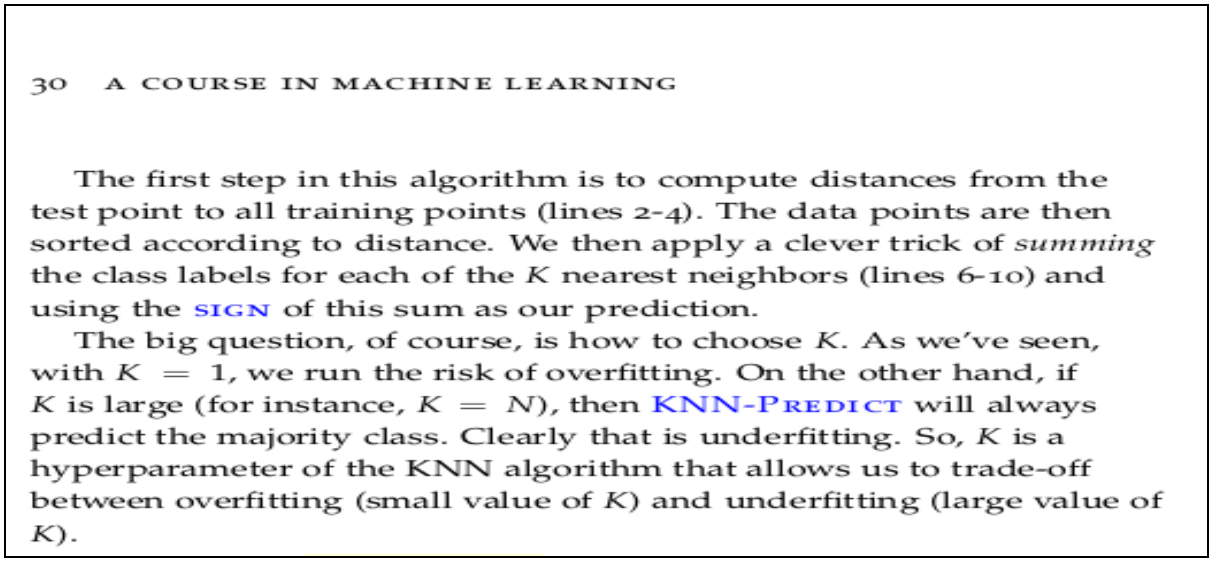
O altă abordare, a fost aceea de a încerca, tot prin intermediul algoritmilor de Machine Learning, în baza unor trăsaturi extrase, să creez date de antrenament și de test (folosind fragmente de text). Nici această abordare nu s-a dovedit a fi potrivită, întrucât necesita o cercetare detaliată și de lungă durată, precum și crearea unui număr suficient de mare de date de antrenament care să fie utilizate în cadrul algoritmului.

3.2 Soluția curentă

Pentru a realiza o soluție cât mai eficientă am decis să păstrez ideea de a extrage trăsături specifice headerelor și footerelor. Acestea au fost identificate în baza unor euristici și au fost reprezentate prin intermediul unor expresii regulate.

Astfel, pentru identificarea trăsăturilor, am analizat mai multe documente PDF. În urma analizei am descoperit că majoritatea PDF-urilor aveau urmatoarele structuri:

1. Numărul paginii urmat de titlul cărții (Fig. 3).



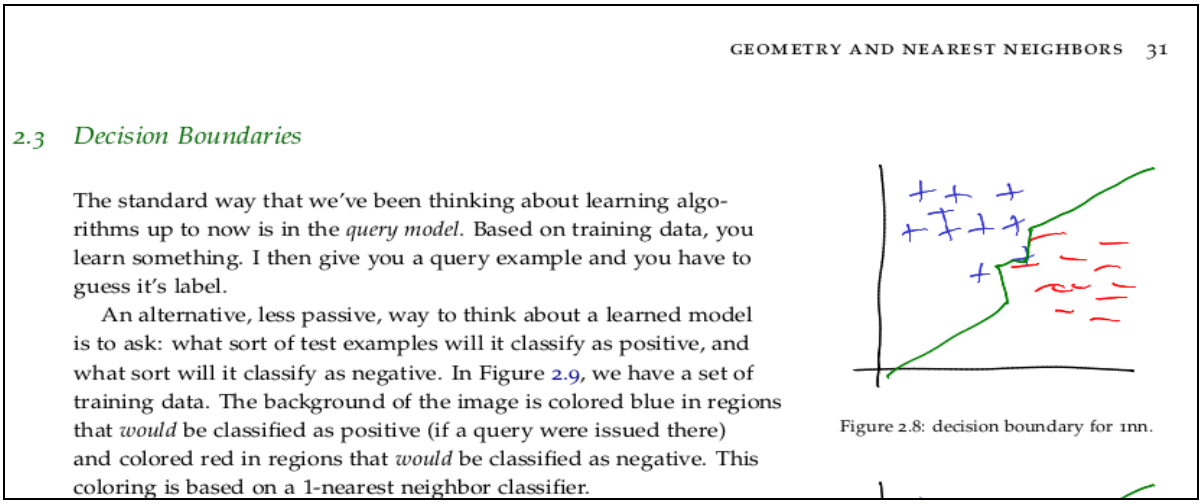
30 A COURSE IN MACHINE LEARNING

The first step in this algorithm is to compute distances from the test point to all training points (lines 2-4). The data points are then sorted according to distance. We then apply a clever trick of *summing* the class labels for each of the K nearest neighbors (lines 6-10) and using the SIGN of this sum as our prediction.

The big question, of course, is how to choose K. As we've seen, with $K = 1$, we run the risk of overfitting. On the other hand, if K is large (for instance, $K = N$), then KNN-PREDICT will always predict the majority class. Clearly that is underfitting. So, K is a hyperparameter of the KNN algorithm that allows us to trade-off between overfitting (small value of K) and underfitting (large value of K).

Fig. 3

2. Numele capitolului urmat de numărul paginii (Fig. 4).



GEOMETRY AND NEAREST NEIGHBORS 31

2.3 *Decision Boundaries*

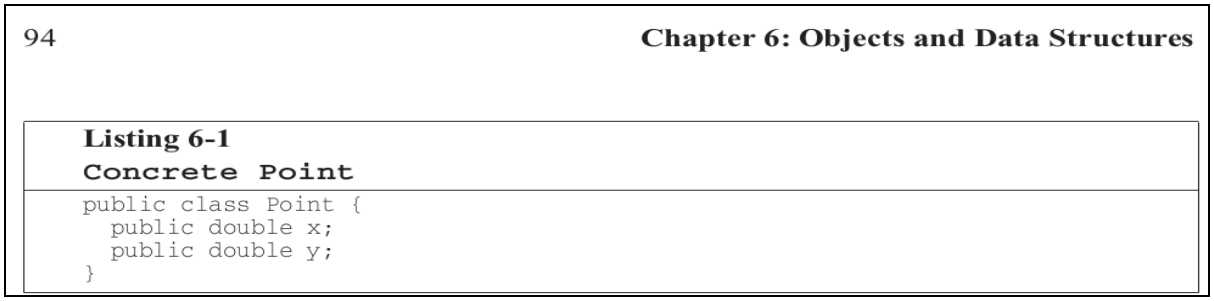
The standard way that we've been thinking about learning algorithms up to now is in the *query model*. Based on training data, you learn something. I then give you a query example and you have to guess it's label.

An alternative, less passive, way to think about a learned model is to ask: what sort of test examples will it classify as positive, and what sort will it classify as negative. In Figure 2.9, we have a set of training data. The background of the image is colored blue in regions that *would* be classified as positive (if a query were issued there) and colored red in regions that *would* be classified as negative. This coloring is based on a 1-nearest neighbor classifier.

Figure 2.8: decision boundary for 1nn.

Fig. 4

3. Numărul paginii, numărul capitolului insoțit de titlul acestuia (Fig. 5).



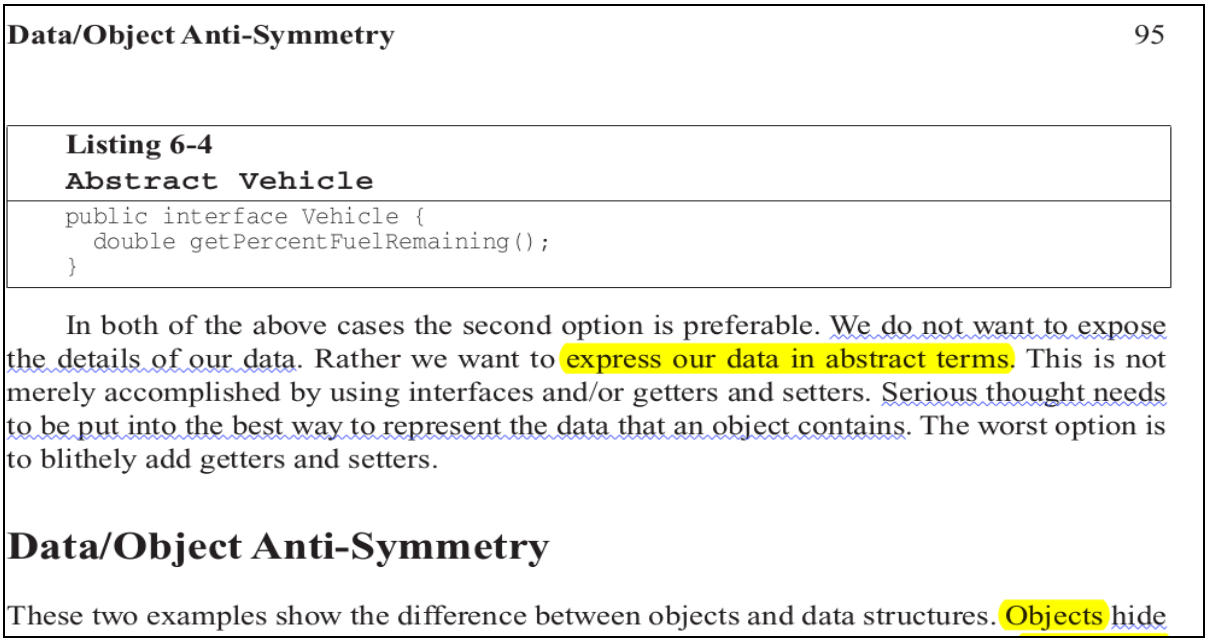
94 Chapter 6: Objects and Data Structures

Listing 6-1
Concrete Point

```
public class Point {
  public double x;
  public double y;
}
```

Fig. 5

4. Titlul subcapitolului urmat de numărul paginii (Fig. 6).



Data/Object Anti-Symmetry 95

Listing 6-4
Abstract Vehicle

```
public interface Vehicle {
  double getPercentFuelRemaining();
}
```

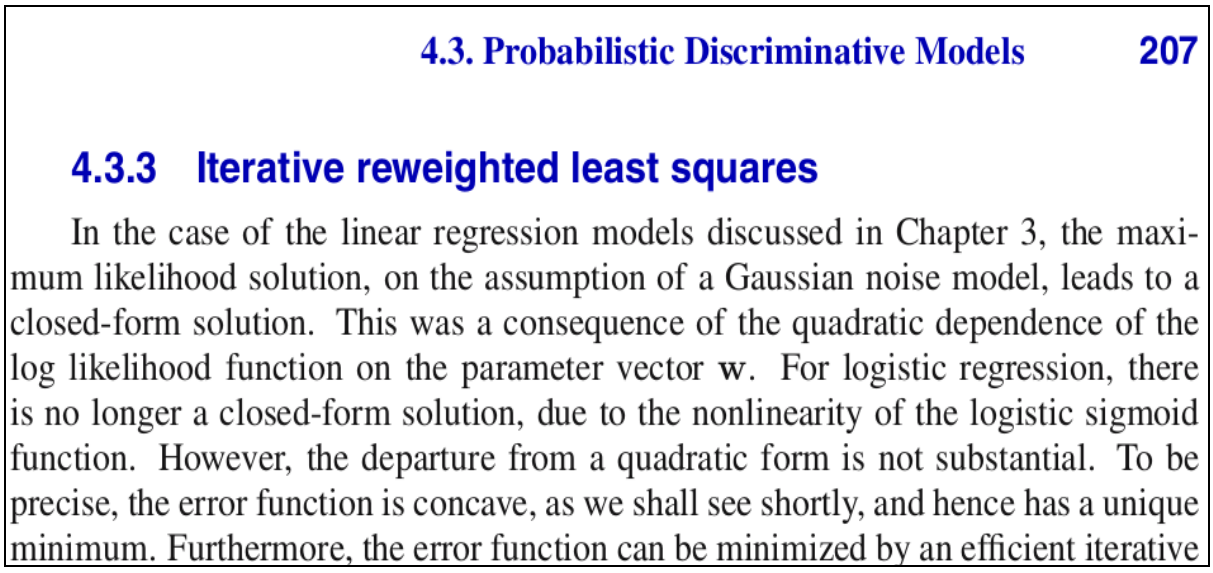
In both of the above cases the second option is preferable. We do not want to expose the details of our data. Rather we want to express our data in abstract terms. This is not merely accomplished by using interfaces and/or getters and setters. Serious thought needs to be put into the best way to represent the data that an object contains. The worst option is to blithely add getters and setters.

Data/Object Anti-Symmetry

These two examples show the difference between objects and data structures. Objects hide

Fig. 6

5. Numărul subcapitolului, numele acestuia urmat de numărul paginii (Fig. 7)



4.3. Probabilistic Discriminative Models 207

4.3.3 Iterative reweighted least squares

In the case of the linear regression models discussed in Chapter 3, the maximum likelihood solution, on the assumption of a Gaussian noise model, leads to a closed-form solution. This was a consequence of the quadratic dependence of the log likelihood function on the parameter vector $\mathbf{w}$. For logistic regression, there is no longer a closed-form solution, due to the nonlinearity of the logistic sigmoid function. However, the departure from a quadratic form is not substantial. To be precise, the error function is concave, as we shall see shortly, and hence has a unique minimum. Furthermore, the error function can be minimized by an efficient iterative

Fig. 7

6. Numărul paginii, numărul capitolului, urmat de numele acestuia (Fig. 8).

206 4. LINEAR MODELS FOR CLASSIFICATION

For a data set $\{\boldsymbol{\phi}_n, t_n\}$, where $t_n \in \{0, 1\}$ and $\boldsymbol{\phi}_n = \boldsymbol{\phi}(\mathbf{x}_n)$, with $n = 1, \ldots, N$, the likelihood function can be written

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n} \tag{4.89}$$

where $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$ and $y_n = p(\mathcal{C}_1|\boldsymbol{\phi}_n)$. As usual, we can define an error function by taking the negative logarithm of the likelihood, which gives the *cross-entropy* error function in the form

Fig. 8

O altă observație importantă referitoare la headere și footere este aceea că în unele pagini alternează trăsăturile. Mai exact, după cum se poate observa în figurile Fig. 3 și Fig. 4, pe paginile cu număr par se regasește numărul paginii urmat de titlul cărții, în timp ce pe paginile cu număr impar se regasește numele capitolului urmat de numărul paginii.

3.3 Rolul expresiilor regulate în identificarea trăsăturilor

După cum am amintit și mai sus, pentru a identifica trăsăturile specifice headerelor și footerelor am alcătuit o serie de expresii regulate.

Desigur, soluția fiind destul de complexă și expresiile regulate funcționând în baza unor algoritmi de tip greedy, am acordat o atenție deosebită asupra modului în care acestea au fost compuse.

Astfel, am creat urmatoarele expresii regulate:

- "([1-9][0-9]*)" – are rolul de a identifica număr de pagină.

- "(([A-Z][A-Za-z]*)+)([1-9][0-9]*)" – identifică mai multe cuvinte ce încep cu literă de tipar și se termină cu un număr.

- "([1-9][0-9]*)(([A-Z][A-Za-z]*)+)" – identifica un număr urmat de mai multe cuvinte ce încep cu literă de tipar.

- "([1-9]((\\.)[0-9]*)+)(([A-Z][A-Za-z]*)+)" – identifică un număr de subcapitole urmat de cuvinte ce încep cu literă de tipar.

- "([1-9][0-9]*)([1-9]((\\.)[0-9]*)+)(([A-Z][A-Za-z]*)+)" - identifică un număr de pagină urmat de un număr de subcapitole și de cuvinte ce încep cu literă de tipar.

- "([1-9]((\\.)[0-9]*)+)(([A-Z][A-Za-z]*)+)([1-9][0-9]*)" - identifică un număr de subcapitole și de cuvinte ce încep cu literă de tipar urmate de un număr de pagină.

- "([A-Z][a-z]* *)([a-zA-Z]*)" – primul cuvânt incepe cu literă de tipar obligatoriu și poate fi urmat de mai multe cuvinte ce incep sau nu cu majusculă.

- "([A-Z][a-z]* *)([a-zA-Z]*)([1-9][0-9]*)" - primul cuvânt incepe cu literă de tipar obligatoriu, poate fi urmat de mai multe cuvinte ce incep sau nu cu majusculă și conține la final un număr.

- "([1-9][0-9]* *)([A-Z][a-z]* *)([a-zA-Z]*)" – Rândul începe cu un număr, continuă obligatoriu cu un cuvânt ce începe cu literă de tipar și poate continua cu mai multe cuvinte ce pot începe sau nu cu majusculă.

- "([1-9]((\\.)[0-9]*)+)([A-Z][a-z]*)([a-zA-Z]*)" – Rândul începe cu un număr de capitol sau de subcapitol, continuă obligatoriu cu un cuvânt ce începe cu literă de tipar și poate continua cu mai multe cuvinte ce pot începe sau nu cu majusculă.

- "([1-9][0-9]* *)([1-9]((\\.)[0-9]*)+)([A-Z][a-z]*)([a-zA-Z]*)" - Rândul începe cu un număr de pagină urmat de un număr de capitol sau de subcapitol, continuă obligatoriu cu un cuvânt ce începe cu literă de tipar și poate continua cu mai multe cuvinte ce pot începe sau nu cu majusculă.

- "([1-9]((\\.)[0-9]*)+)([A-Z][a-z]* *)([a-zA-Z]*)([1-9][0-9]*)" - Rândul începe cu un număr de capitol sau de subcapitol, continuă obligatoriu cu un cuvânt ce începe cu literă de tipar, poate continua cu mai multe cuvinte ce pot începe sau nu cu majusculă și se termină cu un număr.

- "([1-9][0-9]*\\))(*[a-zA-Z]*)" – Rândul începe cu un număr urmat de o paranteză rotundă și de mai multe cuvinte ce pot începe sau nu cu majusculă. Această expresie regulată este folosită pentru identificarea notelor de subsol.

3.4 Soluții anterioare – complexitate

Inițial, algoritmul era structurat in 3 faze de bază:

1. Extragerea textului din document cu ajutorul metodelor puse la dispoziție de biblioteca PDFBox
2. Prelucrarea textului (în această fază se verifica fiecare linie extrasă daca se potrivește cu o listă de expresii regulate predefinite) și eliminarea headerului sau footerului în cazul în care linia corespunzătoare se potrivea cu expresia regulată.
3. Crearea noului fișier, de tip txt, avand headerele și footerele eliminate.

În a doua fază, datele erau procesate de mai multe ori. Prima data se construia un hash map ce reținea frecvența apariției fiecărei linii în text. În cazul în care linia respectiva reprezenta un header și conținea număr de pagină, trebuia salvat doar grupul ce nu conținea numărul paginii, intrucât acesta varia, iar în momentul în care se făcea comparație între șirurile de caractere, acestea ar fi fost diferite.

După ce se construia hash map-ul, în baza frecvenței apariției în text a fiecărei linii ce se potrivea cu lista de expresii regulate, textul era parcurs din nou și fiecare rând ce reprezenta cu probabilitate mare o trăsătură de header sau footer era eliminat.

Pentru că complexitatea procesării era destul de mare, cu atât mai mult cu cât și expresiile regulate măreau semnificativ complexitatea, am decis să păstrez ideile de bază și să încerc o nouă abordare, menită să reducă complexitatea.

3.5 Structura algoritmului

Cu ajutorul expresiilor regulate discutate anterior, folosind ideile principale regăsite în algoritmul inițial și în baza mai multor euristici, am compus un algoritm menit să ofere o soluție problemei.

Algoritmul se bazează în principal pe aplicarea unei liste de expresii regulate fiecărui rând în parte extras cu ajutorul metodei getText(). În cazul în care acest rând se potriveste cu expresia regulată și se regăsește cu o frecvență ridicată în cadrul textului, acesta va fi eliminat, în caz contrar, rândul va fi adăugat în fișierul text.

Desigur, ca acest lucru să fie posibil, am apelat la mai multe metode de stocare de tip hash map.

Citirea și procesarea textului se fac pagină cu pagină.

Expresiile regulate menite sa identifice headerele și footerele sunt salvate intr-o listă.

Inițial, se păsrează un map de liste, astfel: cheia este reprezentată de numărul paginii, iar valoarea este reprezentată de o listă ce cuprinde liniile existente în acea pagină.

În timp ce pagina este procesată, fiecare rând este adăugat in lista ce salvează liniile din pagină. Se parcurge lista de expresii regulate și se verifică dacă linia curentă se potrivește cu una dintre aceste expresii.

1. În cazul în care linia curentă se potrivește cu una dintre expresiile regulate, se construiește un obiect ce conține urmatoarele informații:

- Linia curentă
- Grupl pe care s-a făcut match
- Frecvența apariției liniei in text
- Numărul paginii

Obiectul creat in pasul anterior este adăugat intr-un map, unde cheia este linia din pagină, iar valoarea este obiectul. Acestea sunt utilizate ulterior pentru a elimina din map-ul initial liniile ce au o frecvență ridicată de apariție.

2. În cazul în care linia curentă nu se potrivește cu nici una dintre expresiile regulate, nu se mai construiește obiectul.

La final, se iterează prin lista de obiecte ce au făcut match și se elimină din paginile corespunzătoare fiecărui rând în parte (map-ul initial).

Se iterează din nou prin toate paginile filtrate și se scriu în fișier.

Se vor păstra două fișiere, cel initial și cel prelucrat, pentru o comparație ulterioară.

3.6 Probleme întâmpinate

Pe lângă problema complexității algoritmului care a fost discutată anterior, au mai existat o serie de impedimente, dintre care le voi aminti pe cele mai importante și le voi descrie succint.

O primă problemă întâmpinată a fost cea a caracterelor speciale, care nu pot fi descifrate cu ajutorul bliotecii PDFBox, astfel că soluția actuală nu permite eliminarea headerelor și footerelor din documente PDF ce conțin diacritice.

De asemenea în cadrul determinării headerelor și footerelor am avut o abordare puțin mai avansată, ce ar fi putut fi folosită de utilizatori mai experimentați, întrucât necesita interacțiune cu utilizatorul mai mare. Mai exact am încercat să elimin headerele/footerele folosind o metodă regasită în clasa PdfTextStripperByArea. Utilizatorul trebuia cu ajutorul unui tool să dea coordonatele (în pixeli) unui dreptughi ce reprezenta aria din care urma să fie extras textul (Fig. 9 - extract text by area și Fig. 10 - get text by area). Însă metodele puse la

dispoziție de PDFBox nu extrăgeau corect textul din aria descrisă de coordonatele date; se extrageau date în plus, ceea ce indica faptul că se depășea aria descrisă.

```
public String extractTextByArea(PDDocument pdDoc, int pageNumber) throws IOException {
    Rectangle2D region = new Rectangle2D.Double(0, 0, 791, 312);//header 70 height
    PDFTextStripperByArea stripper;
    PDPage page = pdDoc.getPage(pageNumber);
    stripper = new PDFTextStripperByArea();
    stripper.addRegion("region", region);
    stripper.extractRegions(page);
    String text = stripper.getTextForRegion("region").trim();
    return text;
}
```

Fig. 9 - extract text by area

```
public String getTextByArea()  {
    try(RandomAccessFile f = new RandomAccessFile(new File(filePath), "r")){
        PDFParser parser = new PDFParser(f);
        parser.parse();
        return extractTextByArea(new PDDocument(parser.getDocument()), 22);
    }catch (IOException e){
        e.printStackTrace();
        return null;
    }
}
```

Fig. 10 - get text by area

O altă problemă legată de extragerea textului folosind PDFBox a fost aceea că nu se pastrează formatarea, iar spațiile multiple sunt înlocuite cu unul singur.

O problemă întâmpinată în ceea ce privește expresiile regulate a fost aceea că nu funcționează metacaracterele în Java. Inițial foloseam „\\s” pentru a identifica orice tip de spațiu existent, însă am fost nevoită să rectific această eroare și să modific expresiile regulate, întrucât programul nu mai compila în cazul în care expresia regulată nu se potrivea cu linia din text.

4 Concluzii

Acest capitol cuprinde o descriere a obiectivelor atinse, precum și a celor neatinse. Deasemenea voi descrie și o serie de modalități prin care aplicația ar putea fi îmbunătățită pe viitor.

4.1 Obiective atinse

Lucrarea își propunea să elimine headerele și footerele din documentele de tip PDF folosind o listă de expresii regulate determinate prin itermedul extragerii caracteristicilor specifice headerelor si footerelor.

De asemenea, eficiența aplicației a fost cu mult crescută în urma modificării structurii expresiilor regulate.

Un alt obiectiv atins, la fel de important este acela ca am reușit să parcurg documentul în cât mai puțini pași posibili si de cât mai puține ori. Acest lucru a fost posibil datorită utilizării hash map-urilor în implementare, pentru a stoca informații despre fiecare pagină și linie în parte.

Pe lângă obiectivele atinse am reușit să le ating, aplicația mai dispune si de o metodă puțin mai avansată de extragere a textului. Astfel, utilizatorul trebuie să declare coordonatele X și Y, înălțimea și lungimea în pixeli. Aceste coordonate vor contura o zonă dreptungiulară din care poate fi extras textul.

4.2 Obiective neatinse

Desigur că nu toate obiectivele pe care mi le-am propus au fost atinse. Unul din principalele obiective pe care nu am putut să îl ating a fost acela de a elimina titlurile de capitole, cuprinsul si toate datele ce nu fac parte din conținutul textului.

De asemenea nu toate tipurile de headere și footere sunt identificate și eliminate. Pentru tipuri de headere cu conținut mai specific, acest lucru nu a fost posibil, întrucât nu am putut găsi o expresie regulată care să il diferențieze față de alte linii din text, iar expresiile regulate actuale nu îl pot identifica.

Același lucru este valabil și în ceea ce privește determinarea footerelor cu conținut specific. Înafară de cele care conțin numărul paginii sau note de subsol de forma număr pagina urmat de paranteză rotundă. O altă problemă o reprezintă și notele de subsol scrise pe mai multe rânduri.

Un alt tip de header ce nu poate fi identificat este cel care conține numărul paginii scris cu cifre romane.

4.3 Directive viitoare

Aplicația ar putea fi îmbunătățită, prin crearea unei interfețe menite să ușureze munca utilizatorului.

De asemenea, o soluție alternativă ar fi aceea de a implica algoritmi de Machine Learning în procesul de identificare a headerelor și footerelor. O abordare iteresanta ar putea fi utilizarea OCR pentru a recunoaște aceste informații.

Tot în cadrul aceleiași aplcații se poate realiza etichetarea porțiunilor de text cu tag-uri de tip HTML.

De asemenea, se poate încerca eliminaea headerelor și a footerelor din reviste, ziare sau articole, iar în cazul în care sunt scrise pe mai multe coloane să se ordoneze.

4.4 Opinia personală

Consider că această aplicație reprezintă o soluție simplă și eficientă pentru problema eliminării headerelor și footerelor din textele de bază și poate fi extinsă cu ușurință prin adaugarea de noi funcționalități ce țin de manipularea conținutului documentelor PDF.

5 Tabel de imagini

6 Bibliography

1. **Scott Vanderbeck, Joseph Bockhorst.** Header and Footer Extraction by Page-Association.

2. https://acrobat.adobe.com/us/en/acrobat/about-adobe-pdf.html. [Interactiv]

3. https://www.adobe.com/content/dam/acom/en/devnet/pdf/PDF32000_2008.pdf. [Interactiv]

4. https://pdfbox.apache.org. [Interactiv]