Aplicarea tehnicilor de optimizare în alogritmi de inteligență artificială, cu accent pe SVM, KNN și Gradient Descent

Irimia Ioan-David
Grupa 341

Mitici Alexandru
Grupa 341

Axinte Sebastian
Grupa 341

May 17, 2020

Abstract

Majoritatea algoritmilor de inteligență artificială supervizată pun o problemă constrânsă de optimizare ce poate fi rezolvată cu algoritmi precum (Stochastic) Gradient Descent. Programele de clasificare (KNN, SVM) și de prezicere (regresie liniară) folosesc optimizări pentru a reduce timpul de procesare sau a îmbunătăți acuratețea. Această lucrare prezintă algoritmi de optimizare pentru probleme de inteligență artificială (Gradient Descent și Stochastic Gradient Descent în particular, pentru regresie liniară) și explică SVM-urile pe baza articolului "A Dual Coordinate Descent Method for Large-scale Linear SVM", propunând o rezolvare alternativă pentru acestea. Sunt prezentați și comparați, de asemenea, 3 algoritmi de rezolvare pentru K-Nearest Neighbors. Lucrarea conține o anexă în care toate problemele abordate sunt implementate cu exemple în Python 3.6.

1 Introducere

Algoritmii de inteligență artificială trebuie să fie cât mai eficienți și să facă cât mai puține greșeli (adică să aibă acurateţe cât mai mare). Din acest motiv, algoritmii de IA sunt mai întâi îmbunătățiți din punct de vedere al acurateței, de multe ori rezultând un algoritm complet nou, și apoi din punct

de vedere al complexității, ca să folosească cât mai puține resurse și cât mai puțin timp. Numim acest proces optimizarea algoritmilor de IA. Lucrarea aleasă de noi face optimizări pe algoritmi de IA care folosesc SVM-uri, așa că vom discuta SVM-uri și optimizările lor în cele ce urmează.

1.1 Anexă

Toate fișierele sursă Python sunt publice și se pot găsi aici: https://github.com/MiticiAlexandru/Tehnici-Optimizare---Proiect-2020

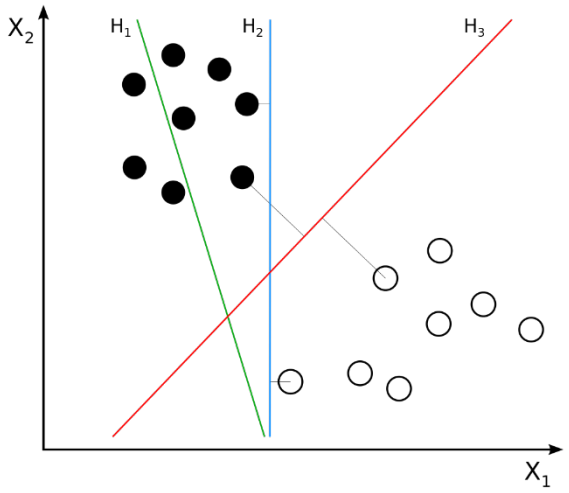
2 Noțiuni generale despre Support-Vector Machines

Un SVM (Support-Vector Machine) este un model pentru algoritmi de IA de învățare supervizată. De obicei, SVM-urile sunt folosite de algoritmi de IA de clasificare (de exemplu, KNN) sau de regresie (liniară, multiplă, logistică etc). Algoritmii care folosesc SVM-uri sunt non-probabilistici. Numele de SVM vine de la modul în care sunt reprezentate datele: având un set de date cu N caracteristici, SVM le reprezintă prin vectori din spațiul R^N, fiecare reprezentând o intrare în setul de date cu fiecare caracteristică măsurată pe o dimensiune a spațiului. Aceste intrări sunt mapate în spațiul SVM-ului astfel încât clasele de date să fie cât mai ușor de separat. Intrări noi sunt puse în aceeași categorie ca cele din subspațiul în care se află.

În cazul în care datele de antrenare nu au categoriile asignate, trebuie trecut de la un algoritm de învățare supervizată la unul de învățare nesupervizată. Algoritmii de acest fel care folosesc SVM-uri fac "clustering" pentru a determina singuri clasele pe baza caracteristicilor comune.

Formal, rolul unui SVM în algoritmii de clasificare este de a reprezenta datele într-un spațiu N-dimensional și de a face posibilă trasarea de hiperplanuri care separa datele în clase. Mai jos avem 3 separări diferite pentru un SVM cu 2 clase.

Figure 1: Exemplu de trasare e hiperplanelor



După cum se observă, H1 nu reușește să separe datele într-un mod corect. H2 reușește acest lucru dar distanța dintre el și date este mică și deci intrări viitoare pot fi clasificate greșit. H3 oferă separarea cea mai bună, cu distanța cea mai mare între date și hiperplan și cea mai bună acuratețe.

Un algoritm de optimizare pe SVM-uri este "Gradient Descent", aplicat pentru a găsi minimul local al unei funcții diferențiabile. Pentru a găsi minimul funcției cu "Gradient Descent", facem pași proporționali cu inversul gradientului aplicat funcției în punctul curent. Discutăm "Gradient Descent" și "Stochastic Gradient Descent" în detaliu în descrierea algoritmilor implementați în 3.3) și 3.4).

2.1 Kernel Tricks

De cele mai multe ori, separarea nu se poate face printr-un hiperplan drept. Spre exemplu, dacă setul nostru de date este pe o singură dimensiune (adică valorile lui x din setul de date au un singur element fiecare), putem avea situația următoare:

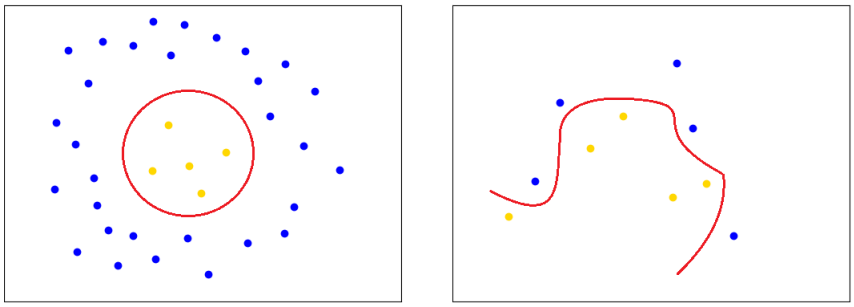
Figure 2: Graficul unui set de date unidimensional

În acest caz nu putem separa punctele cu un singur hiperplan unidimensional.

Putem folosi un "Kernel Trick" (o funcție kernel) care adaugă o dimensiune în plus setului de date pentru a reprezenta punctele într-un plan bidimensional pe o parabolă. Pentru fieacare punct pe coordonata x, calculăm kernel(x) pe axa OY. Astfel vom putea separa punctele printr-o dreaptă.

Alt caz pentru un Kernel Trick este când avem puncte în mijloc sau plasate "aleator" în două dimensiuni.

Figure 3: Seturi de date ce nu pot fi separate cu un hiperplan în două dimensiuni



2.2 L1-SVM și L2-SVM

L1-SVM și L2-SVM sunt optimizări pe SVM. L1-SVM și L2-SVM încearcă să reprezinte datele astfel încât distanțele dintre grupurile ("cluster") de date să fie cât mai mari și astfel distanța dintre date și hiperplanele care le separă să fie cât mai mare și eroarea să fie cât mai mică. Se încearcă aceasta fără o distorsionare prea mare a datelor prin rezolvarea problemei de optimizare:

$$\min_{w} \quad \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi(w; x_i, y_i)$$

Diferența dintre L1-SVM și L2-SVM este valoarea aleasă pentru funcția $\xi(w; x_i, y_i)$

- L1-SVM: $max(1 - y_i w^T x_i, 0)$
- L2-SVM: $max(1 - y_i w^T x_i, 0)^2$

3 Algoritmi Implementați

În cazul algoritmilor de IA, acuratețea și minimizarea numărului de operații complexe efectuate de aceștia sunt factorii cei mai importanți care descriu un algoritm. În implementările noastre am încercat să creștem semnificativ acuratețea unor algoritmi de IA și apoi să îi optimizăm.

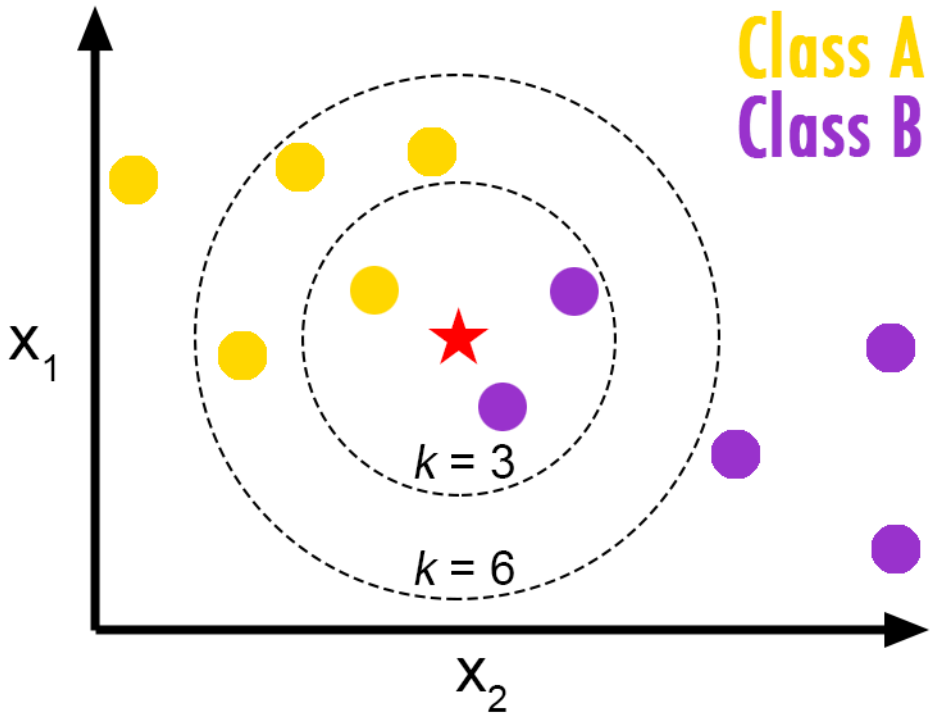
3.1 Optimizări pe KNN

Problema Abordată: În cele ce urmează în această secțiune, încercăm să optimizăm algoritmul KNN prin găsirea unui K optim.

Rezolvarea problemei: Algoritmul KNN este unul din cei mai simpli algoritmi de clasificare. KNN (K-Nearest Neighbours) decide clasa unei intrări astfel: algoritmul caută cei mai apropiați K vecini ai intrării și îi atribuie clasa cea mai comună dintre acești vecini.

Deși algoritmul este simplu de înțeles și implementat, există 2 factori importanți care pot influența acuratețea și performanța algoritmului: funcția aleasă pentru calcularea distanței și valoarea aleasă pentru K. Există multe funcții propuse pentru calcularea distanței, cea Euclidiană fiind mai comună și cea aleasă de noi în implementări.

Figure 4: Graficul KNN



Avem trei implementări de KNN care rulează concomitent, cu optimizări alese pentru găsirea unei valori pentru K. Aceștia sunt antrenați pe același set de date de antrenare (numit A) și acuratețea lor finală este măsurată pe același set de date de testare (numit T).

Prima implementare este un KNN simplu, fără optimizări, cu valoarea lui K aleasă ca 3 (majoritatea implementărilor aleg un K arbitrar, cel mai des 2 sau 3). Deși aceasta este cea mai rapidă dintre implementări, suferă din punct de vedere al acurateței.

Celelalte implementări împart setul de date de antrenare în 2: un set pe care se face antrenarea efectivă (90% din datele totale ale setului de antrenare, pe care îl numim α) și un set de testare pentru K (restul datelor de antrenare, pe care îl numim β).

După selectarea unui K și antrenarea datelor pe α, se testează acuratețea algoritmului pe setul β.

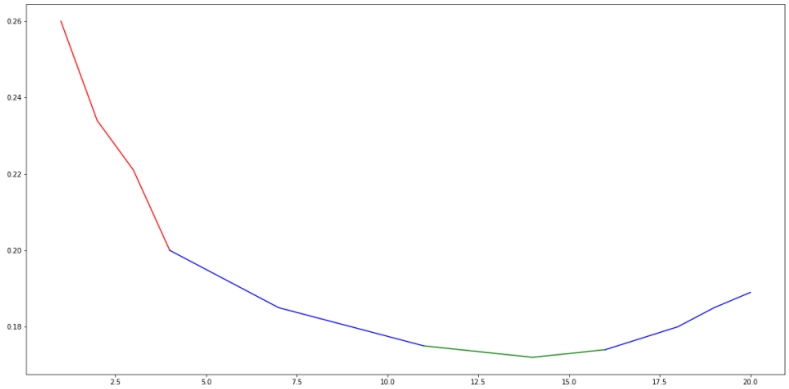
Al doilea algoritm măsoară acuratețea modelului pentru toate valorile posibile ale lui K (de la 1 la N, unde N este numărul intrărilor din α) și alege K cu cea mai bună acuratețe. Această implementare devine foarte lentă și ineficientă pentru un set de date foarte mare.

Al treilea algoritm aplică o metodă "Divide et Impera" pentru a-l alege pe K, folosind tot acuratețea pentru alegerea sa. Se pornește de la intervalul 1, ..., N și se aplică recursiv până se găsește K optim. Pentru un interval ales, se măsoară acuratețea pentru valorile din capetele sale, intervalul restrângându-se la jumătate în funcție de valorile acestea. Algoritmul este mult mai eficient ca cel precedent și știm că găsește aceeași valoare optimă, cu excepția cazurilor în care seturile de date sunt foarte mici și acuratețea pe valorile lui K nu este stabilă. Noi am găsit un set de date relativ mic pentru antrenare și testare (300 de intrări în total, antrenarea efectivă fiind făcută pe 67 de intrări), totuși se observă că algoritmul funcționează și pe acesta, algoritmii 2 și 3 având aceeași acuratețe. Numărul mic de date de intrare pe care îl avem explică și acuratețea slabă în general a modelului nostru (57.70 pentru algoritmul 1 și 62.55 pentru algoritmii 2 și 3) deoarece KNN are nevoie de seturi de date mai mari decât alți algoritmi de clasificare pentru a ajunge la o acuratețe bună. Totuși, și în aceste condiții, se observă o creștere semnificativă a acurateței măsurate pe setul T după aplicarea algoritmilor.

Al treilea algoritm este optimizat și are cea mai bună acuratețe. Mai jos avem un grafic, reprezentativ KNN aplicat pentru seturi mari de date. Pe axa OX sunt valorile alese pentru K și pe OY este eroarea de clasificare

(invers proporțională cu acuratețea).

Figure 5: Graficul optimizării KNN



După cum se observă, pentru valori mici ale lui K (primele 2 secțiuni) eroarea este foarte mare dar scade cu cât K crește, ajungând până la un interval de stabilizare (secțiunea verde) în care se află K-ul optim (cu eroarea cea mai mică) și apoi începe iar să crească (ultima secțiune).

3.2 Hiperplane în SVM

Problema abordată: În cele ce urmează în această secțiune, presupunem că lucrăm cu un SVM cu 2 clase, notate cu -1 și 1, și ne propunem să rezolvăm problema găsirii hiperplanului care le separă.

Rezolvarea problemei: Numim un Hiperplan (vizualizat pe un grafic în 2 dimensiuni) dreapta care separă setul de date în două categorii. Convențional, cele două categorii sunt etichetate cu -1 și +1 pentru a facilita formula cu care putem face clasificarea.

De asemenea, numim Support Vectors dreptele paralele cu hiperplanul de separare care trec prin cele mai apropiate două puncte din setul de date: unul pe stânga (pentru -1) și unul pe dreapta (pentru +1).

Odată ce avem hiperplanul trasat, putem foarte ușor să prezicem din ce categorie face parte un nou punct ce nu aparține setului de date. Dacă este la stânga, atunci va avea eticheta -1, altfel, +1.

Numim margini de separare (Street Widths) distanțele de la support vectors la hiperplanul de separare, și sunt notate convențional D- și D+. Ideal, un SVM încearcă să obțină cea mai mare distanță între cele două seturi de date, adică să maximizeze marginile D- și D+.

3.3 Descrierea funcționalității unui SVM

Un SVM pune o problemă constrânsă de optimizare convexă: încercarea de a maximiza distanțele D- și D+.

Dacă ecuația unei drepte este dată de:

$$y = wx + b$$

Ecuația unui hiperplan este dată de formula:

$$y = w^T x + b \qquad sau \qquad \sum_{i=1}^{n} w_i x_i + b$$

Astfel, problema de optimizare a unui SVM constă în doi factori:

- minimizarea normei lui w
- maximizarea lui b

Notă: Norma lui w se definește astfel:

$$||w|| := \sqrt{\sum_{i=1}^{n} w_i^2}$$

Pentru toate punctele de forma (x, y) din setul de date, constrângerea este dată de formula:

$$y(w^T x + b) \geq 1$$

Important: y = 1 sau -1 și se numește clasă (reprezintă eticheta). y NU este coordonata de pe axa OY. Dacă x-urile sunt pe două coordonate, atunci x va conține o coordonată pe axa OX și una pe axa OY.

Există multe valori w și b care satisfac condiția de mai sus. Vectorii suport vor satisface ecuația:

$$y(w^T x + b) \cong 1$$

Un SVM va aproxima valorile respective și vor rezulta valori foarte aproape de soluția exactă. De obicei, o aproximare bună este îndeajuns

Minimizarea normei lui w este o problemă de optimizare convexă deoarece valoarea sa poate fi reprezentată pe un grafic convex; punctul minim de pe grafic va fi punctul optim. În acest caz, pentru că graficul este convex, $\| w \|$ minim se numește punctul de minim global.

Rezolvarea banală a problemei unui SVM este prin forța brută: se încearcă "toate" opțiunile făcând pași constanți până găsim cel mai mic $\| w \|$.

La fiecare pas, se testează condiția pentru toate punctele (x, y). Deoarece semnele coeficienților lui w contează, când încercăm un anumit w, trebuie să încercăm toate combinațiile posibile de semne ale elementelor sale. Spre exemplu: dacă w este (5, 5), va trebui să încercăm și (-5, 5), (-5, -5) și (5, -5).

Complexitatea acestui algoritm este: $O(w, sw, dataset, b) = n(w) \times n(sw) \times n(dataset) \times n(b)$

Unde:

- n(w) := numărul de iterații prin w
- n(sw) := numărul posibilităților semnelor lui w
- n(dataset) := numărul de puncte din setul de date
- n(b) := numărul de iterații prin care trecem ca să îl aflăm pe b

Notă: testarea semnelor elementelor lui w se poate face prin generarea tuturor vectorilor de aceeași dimensiune ca w de forma (+/-1, +/-1, ... +/-1); înmulțirea lui w transpus cu un astfel de vector va rezulta într-o versiune a lui w în care termenii săi au semnele elementelor din vectorul respectiv.

În anexa lucrării se poate găsi un exemplu simplu de implementare a unui SVM în limbajul Python (Python 3.6); programul este bine documentat în codul sursă. Algoritmul abordat este puțin mai eficient decât forța brută. La fiecare pas pentru $\| w \|$ se verifică dacă valoarea curentă este mai mică decât cea precedentă. Dacă da, atunci continuăm. Dacă nu, atunci schimbăm sensul parcurgerii (pentru că înseamnă că am trecut de punctul de minim) și micșorăm drastic mărimea pasului. În algoritm avem un număr arbitrar/euristic de epoci prin care trecem - repetăm de câteva ori tot procesul (de 3 ori în implementare) și rămânem cu o valoare aproximativă decentă pentru w și b.

O metodă de optimizare și mai eficientă este Gradient Descent, aplicat pentru a găsi minimul local al unei funcții diferențiabile. Pentru a găsi minimul funcției cu Gradient Descent, facem pași proporționali cu inversul gradientului aplicat funcției în punctul curent. Discutăm Gradient Descent și Stochastic Gradient Descent în descriere algoritmilor implementați în 3.2) și 3.3). Anexa conține implementarea celor două tipuri de Gradient Descent asupra unei regresii liniare.

3.4 Gradient Descent

Problema abordată: În cele ce urmează în această secțiune, ne propunem să aplicăm metoda de optimizare "Gradient Descent" pe o regresie liniară.

Rezolvarea problemei: Gradient Descent este o tehnică de optimizare pentru găsirea punctului de minim al unei funcții convexe. Această tehnică se poate aplica la mulți algoritmi de IA, precum Regresia Liniară, Regresia Logistică sau Support Vector Machine.

Dacă o dreaptă are o pantă, un hiperplan are un gradient.

Algoritmul se bazează pe repetarea a 3 pași:

1. Alegerea unei direcții de parcurgere. Se analizează panta în punctul curent din iterație pe baza derivatei funcției

2. Se alege o dimensiune a pasului. Dacă iterația este departe de punctul minim, se vor face pași mai mari. Cu cât se apropie de punctul minim, cu atât pasul este mai mic, și deci are o acuratețe mai bună.

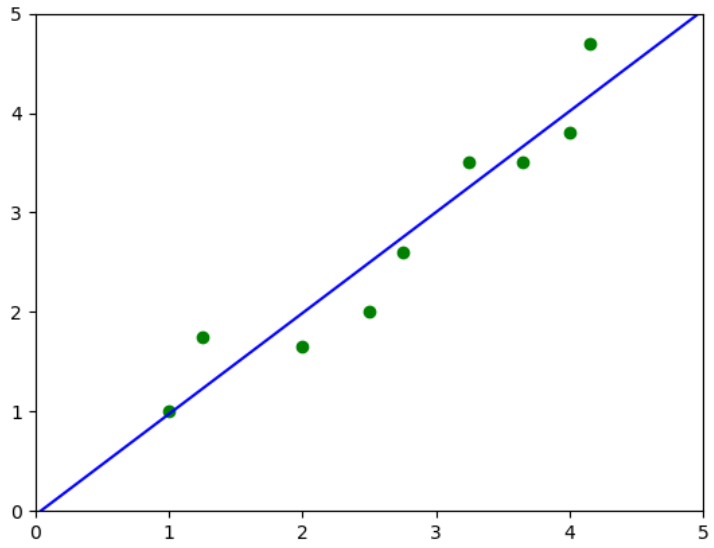
3. Se face verificarea convergenței. Se verifică dacă s-a găsit punctul de minim sau dacă s-a apropiat destul de mult de el.

3.5 Implementarea Gradient Descent

În continuare, se va descrie metoda Gradient Descent aplicată asupra unei regresii liniare.

O regresie liniară trasează o linie pe graficul setului de date care să fie cât mai aproape pe axa OY de toate punctele de pe grafic. Astfel se poate prezice o coordonată a unui nou punct (cu o marjă de eroare care trebuie luată în calcul).

Figure 6: Graficul unei regresii liniare simple



Această marjă de eroare este utilă și se numește eroare reziduală și se calculează cu formula

$$Res(punct) = punct.y - c - m \times punct.x$$

Unde m și c sunt panta și interceptul ecuației dreptei regresiei $y = mx+c$.

Deși panta m poate fi aflată eficient cu Gradient Descent, vom folosi metoda Least Squares pentru aflarea acesteia. Gradient Descent intervine pentru aflarea valorii c.

Pentru aflarea pantei dreptei, întâi se calculeaza media valorilor x (notată mediax) și media valorilor y (notată mediay).

Pentru fiecare punct (x, y):

- $fie \quad x' = x - mediax$
- $fie \quad y' = y - mediay$

Panta m se află din următoarea formulă:

$$\frac{\sum_{i=1}^{n} x'_i * y'_i}{\sum_{i=1}^{n} x'^2_i}$$

Se pornește cu c la o valoare euristică în "stânga" punctului de minim.

Fie Loss(c) funcția de loss, și se calculează cu formula:

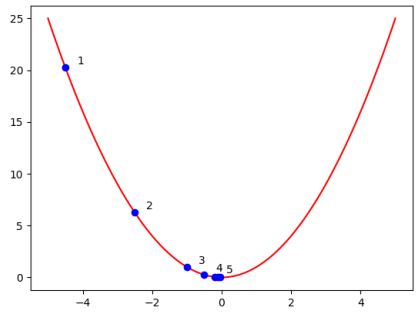
$$Loss(c) = \sum_{i=1}^{n} Res(point_i)^2$$

Funcția obiectiv a algoritmului de optimizare este funcția Loss.

Notă: există multe variații a funcției Loss.

Scopul algoritmului este, deci, să minimizeze funcția Loss. Din moment ce punctele se știu și m se știe, singura variabilă a funcției Loss este c. Deoarece c poate crește liniar, iar Loss(c) este o parabolă, punctele de forma (c, Loss(c)) formează un grafic convex (c este pe axa OX iar Loss(c) pe axa OY).

Figure 7: Graficul funcției Loss(c)



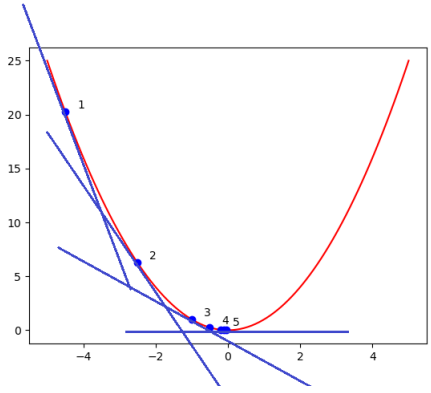
Cu cât linia regresiei se apropie de c-ul perfect, cu atât se apropie c-ul de punctul de minim de pe graficul de mai sus.

În continuare trebuie calculată derivata funcției Loss.

$$Loss(c) = \sum_{i=1}^{n}(-2)Res(point_i)$$

Aceasta este utilă pentru aflarea pantei într-un anumit punct. Cu cât iterația se apropie de punctul de minim al funcției Loss, cu atât panta derivatei devine mai aproape de 0.

Figure 8: Graficul funcției Loss(c) și derivata în câteva puncte



Tot ce rămâne este să definim incrementarea lui c în funcție de o rată de învățare aleasă euristic, anume 0.05:

$$c \leftarrow c + abs(Loss'(c)) * 0.05$$

Algoritmul se încheie când Loss'(c) se apropie suficient de mult de 0. După algoritm, rămânem cu ecuația unei drepte y = mx + c care trasează o linie aproximativ egal îndepărtată de toate punctele pe axa OY.

3.6 Stochastic Gradient Descent

Problema abordată: În cele ce urmează în această secțiune, ne propunem să rezolvăm aceeași problemă ca în secțiunea precedentă dar folosind metoda "Stochastic Gradient Descent".

Rezolvarea problemei: Stochastic Gradient Descent este o metodă de optimizare bazată pe Gradient Descent și aproximează minimul unei funcții.

În Gradient Descent obișnuit se calculează derivata funcției Loss, care are ca punct de minim valoarea optimă pentru c (intercept). Panta se află prin derivata funcției Loss într-un punct c; cu cât se apropie de panta egală cu 0, cu atât se apropie de soluție.

La fiecare iterație, Gradient Descent trebuie să itereze prin toate punctele din setul de date. Această metodă este foarte înceată când avem seturi mari de date. Spre exemplu, 1.000.000 de date înseamnă că algoritmul este de 1.000.000 de ori mai încet decât Stochastic Gradient Descent. Iată de ce:

În Stochastic Gradient Descent, funcția Loss nu iterează prin toate punctele, ci printr-un singur punct luat aleator. La fiecare apelare a funcției Loss, se ia un alt punct aleator din setul de date și se folosește formula

$$StocLoss(c) = (r.y - m \times r.x - c)^2$$

Unde r este un punct aleator din setul de date. Ca urmare, funcția Loss este mai imprecisă, dar mult mai rapidă pentru seturi mari de date.

Derivata funcției StocLoss este:

$$StocLoss'(c) = (-2)(r.y - m \times r.x - c)$$

Calculând derivata acesteia într-un punct aleator, ne vom apropia încet de punctul c optim, adică de punctul minim al funcției StocLoss(c). Anexa lucrării conține un exemplu simplu de Stochastic Gradient Descent pentru regresia liniară.

4 Lucrarea "A Dual Coordinate Descent Method for Large-scale Linear SVM"

Problema dată este una de optimizare pentru un SVM de clasificare binară (adică doar 2 valori posibile de rezultat, -1 și 1). Se analizează N caracteristici (fiecare devine o dimensiune în SVM). Abordarea lucrării este de a căuta caracteristicile cele mai importante ale căror valori determină această clasificare. Unele date nu sunt importante pentru clasificare și pe ele se efectuează operații costisitoare care nu sunt necesare. Problema inițială de creare a unui clasificator primește la început un set de date de antrenare și pe baza cărora se construiește clasificatorul:

$$\{(x_1, y_1), ..., (x_L, y_L)\}$$

x_i reprezintă un punct în spațiul care definește unic o intrare din datele de intrare. Datele propriu-zise pe care se va aplica acest algoritm de clasificare vor fi tot din spațiul R^N

y_i reprezintă valori din -1, 1 și sunt clasele din care aparțin aceste puncte xi. Problema inițială se concentrează pe construirea unui asemenea clasificator folosind un SVM, importantă fiind acuratețea și nu optimizarea algoritmului. Problema abordată de articol este optimizarea acestui clasificator.

Funcțiile SVM "loss function", de exemplu L1-SVM și L2-SVM fac această optimizare, însă algoritmii sunt costisitori, mai ales pe seturi de date foarte mari. Se încearcă deci mai multe metode de modificare a acestor algoritmi pentru a reduce numărul de calcule efectuate. Metoda descrisă în articol este o metodă aplicată pe L2-SVM de optimizare a acestui algoritm.

Metoda constă în aplicarea mai multor iterații asemenea celei descrise mai jos, pornind cu un b^0:

Iterația b^{k+1}
$a^{k,1} = b^k$
$a^{k,N+1} = b^{k+1}$
pentru $i = 2$ la N :
$a_{k,i} = a^{k,i} = [b_1^{k+1}, ..., b_{i-1}^{k+1}, b_i^k, ..., b_N^k]$
$E* : \min_d f(a^{k,i} + de_i)$, unde $e_i = [0, 0, ..., 0, 1, 0, ..., 0]^T$
//calculăm b_{k+1}^i pentru a fi folosit la pasul următor

Funcția f descrisă mai sus face calcule numai în cazul în care este nevoie. De fiecare dată când se folosește acest f (o dată per iterație mică, N ori per iterație mare) se calculează un gradient pe componenta curentă (i din iterația mica). Daca proiecția gradientului este 0 (dreapta d=0 este optimul ecuației E*), rezultă ca nu mai trebuie actualizată componenta i din $a^{k,i}$ și suntem scutiți de calcule adiționale pentru această iterație, de aici optimizarea.

Proiecția gradientului se calculează astfel:

$$\bigtriangledown_i^P f(a) \begin{cases} \bigtriangledown_i f(a), & daca 0 < a_i < U \\ min(0, \bigtriangledown_i f(a)), & daca a_i = 0 \\ max(0, \bigtriangledown_i f(a)), & daca a_i = U \end{cases}$$

References

[1] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press.

[2] G. Yuan, C. Ho and C. Lin *Recent Advances of Large-Scale Linear Classification* in Proceedings of the IEEE, vol. 100, no. 9, pp. 2584-2603, Sept. 2012, doi: 10.1109/JPROC.2012.2188013.

[3] R. Berwick *An Idiot's guide to Support vector machines (SVMs).*

[4] sentdex. *Completing SVM from Scratch - Practical Machine Learning Tutorial with Python*
https://www.youtube.com/channel/UCfzlCWGWYyIQ0aLC5w48gBQ

[5] Artificial Intelligence - All in One, Andrew Ng: Support Vector Machines
https://www.youtube.com/watch?v=hCOIMkcsm_-g&list=PLNeKWBMsAzboNdqcm4YY9x7Z2s9n9q_Tb

[6] How Support Vector Machines work – an example
https://www.machinelearningtutorial.net/2016/12/18/svm-example/

[7] Sunil Ray. *Understanding Support Vector Machine(SVM) algorithm from examples (along with code)*
https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

[8] edureka! *Machine Learning Full Course - Learn Machine Learning 10 Hours*
https://www.youtube.com/watch?v=GwIo3gDZCVQ

[9] el mustapha ben bihi. *Machine Learning: Support Vector Machine - Kernel Trick*
https://www.youtube.com/watch?v=vMmG_7JcfIc

[10] Machine Learning Crash Course with TensorFlow APIs
https://developers.google.com/machine-learning/crash-course

[11] Kayla Matthews, Productivity Bytes. *Data Mapping Using Machine Learning*
https://www.kdnuggets.com/2019/09/data-mapping-using-machine-learning.html

[12] Davesh Poojari. *K-Nearest Neighbors and its Optimization*
https://towardsdatascience.com/k-nearest-neighbors-and-its-optimization-2e3f6797af04

[13] i-king-of-ml (Medium). *KNN(K-Nearest Neighbour) algorithm, maths behind it and how to find the best value for K*

https://medium.com/@rdhawan201455/knn-k-nearest-neighbour-algorithm-maths-behind-it-and-how-to-find-the-best-value-for-k-6ff5b0955e3d

[14] Linear Regression: Simple Steps, Video. Find Equation, Coefficient, Slope
https://www.statisticshowto.com/probability-and-statistics/regression-analysis/find-a-linear-regression-equation/

[15] Sushant Patrikar. *Batch, Mini Batch and Stochastic Gradient Descent*
https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a

[16] Jason Brownlee. *A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size*
https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

[17] Gu. *An Introduction to Support Vector Machines (SVM): Gradient Descent Solution*
https://nianlonggu.com/2019/05/24/tutorial-on-SVM/

[18] Reducing Loss Gradient Descent
https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent

[19] StatQuest with Josh Starmer
https://www.youtube.com/channel/UCtYLUTtgS3k1Fg4y5tAhLbw

[20] Reddit
https://www.reddit.com

[21] Wikipedia
https://www.wikipedia.org/