**Tehnici de Optimizare**

*1) Concepte generale*

Amintim înainte câteva concepte de IA relevante lucrării:

* Seturi de date:
  + De antrenare (perechi de date problemă – rezultat folosite pentru a antrena un program de IA)
  + De testare (date pentru care vrem să aflăm rezultatele folosind programul de IA)
* Tipuri de algoritmi de IA:
  + De clasificare
  + Regresii (prezic o valoare fixă)
  + De detectare de anomalii
  + De clustering (gruparea seturilor de date conform unor pattern-uri găsite)
* Tipuri de învățare automată:
  + Supervizată
  + Nesupervizată
  + Învățarea prin feedback
* Variabile:
  + Independente
  + Dependente
* Rezultate:
  + Valori numerice
  + Categorii (valori dintr-un set anume care spun cărei clase de obiecte aparține obiectul analizat)

*2) Optimizări*

Algoritmii de IA trebuie să fie cât mai eficienți și să facă cât mai puține greșeli (adică să aibă acuratețe cât mai mare). Din acest motiv, algoritmii de IA sunt 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.

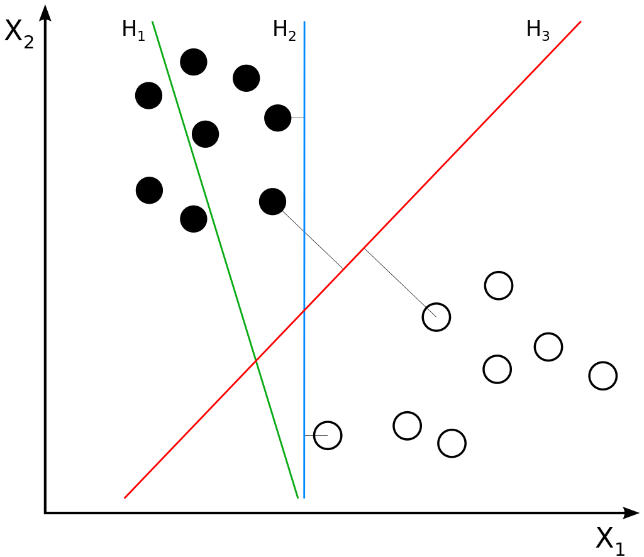
Lucrarea aleasă de noi face optimizări pe algoritmi de IA care folosesc SVM-uri așa că vom discuta SVM-uri și optimizările lor în cele ce urmează.

2.1) SVM

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 , 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.



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.

2.2) L1-SVM și L2-SVM

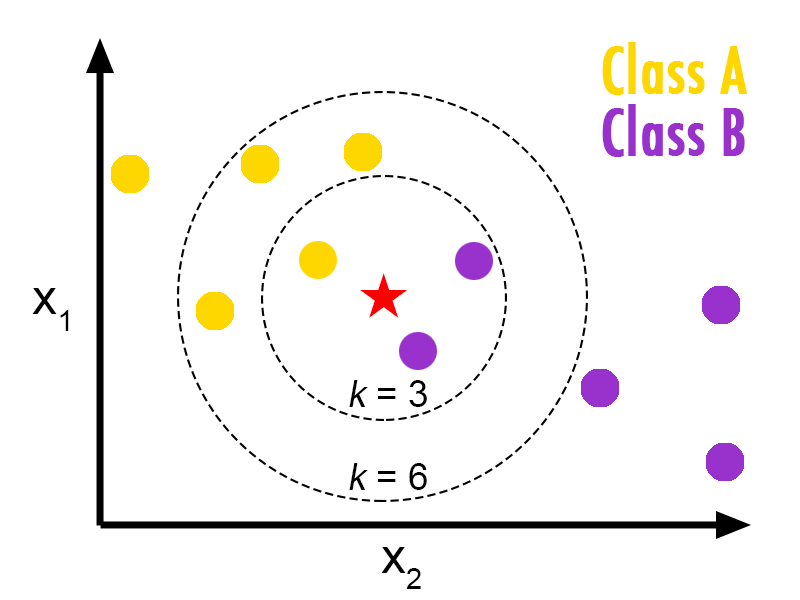
2.3) Gradienți

*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

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



Deși algoritmul este simplu de înțeles și implementat, exista 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, distanța Euclidiană fiind cea mai comună și cea aleasă de noi în implementări.

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 la finală este măsurată pe același set de date de testare (numit T).

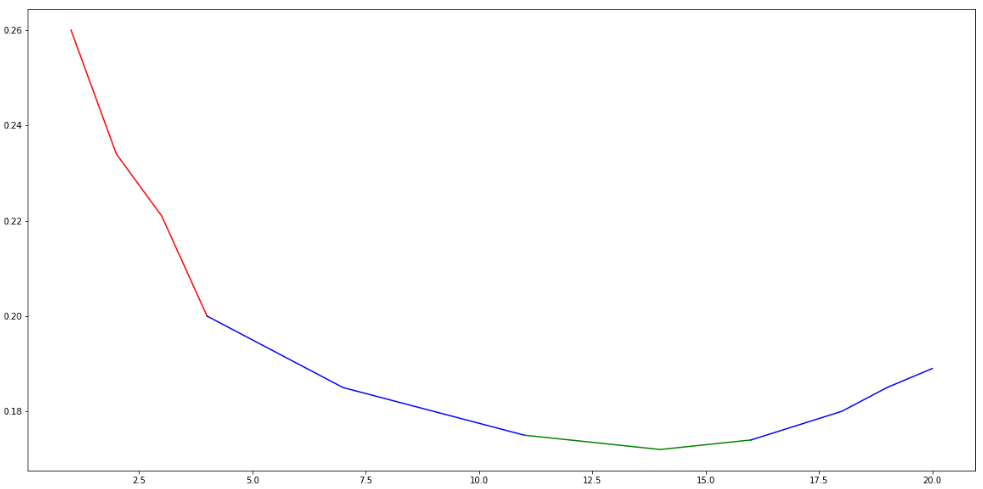
Prima implementare este un KNN simplu, fără optimizări, cu valoarea lui K aleasă ca 3 (majoritatea implementărilor aleg un K arbitrar, cel mai des 2 sau 3). Deși aceasta este cea mai rapidă dintre implementări, suferă din punct de vedere al acurateței.

Celelalte implementări împart setul de date de antrenare in 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 si 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).



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) Gradient Descent

3.3) Stochastic Gradient Descent

*4) Lucrarea „A Dual Coordinate Descent Method for Large-scale Linear SVM”*

Problema data este una de optimizare pentru un SVM de clasificare binara (adică doar 2 valori posibile de rezultat, -1 si 1). Se analizează N caracteristici (fiecare devine o dimensiune in SVM). Abordarea lucrării este de a caută caracteristicile cele mai importante ale căror valori determina aceasta clasificare. Unele date nu sunt importante pentru clasificare si pe ele se efectuează operații costisitoare care nu sunt necesare altfel.

Problema inițiala de creare a unui clasificator primește la început un set de date:

{(, ), . . . ,(, )}

care sunt date de antrenare si pe baza cărora se construiește clasificatorul.

reprezintă un punct in spațiul care definește unic o data de intrare. Datele propriu-zise pe care se va aplica acest algoritm de clasificare vor fi tot din spațiul .

reprezintă valori din {-1, 1} si sunt clasele din care aparțin aceste puncte xi.

Problema inițiala se concentrează pe construirea unui asemenea clasificator folosind un SVM, importanta fiind acuratețea si nu optimizarea algoritmului. Problema abordata de articol este optimizarea acestui clasificator.

Funcțiile SVM „loss function”, de exemplu L1-SVM și L2-SVM fac aceasta 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 descrisa in articol este o metoda aplicata pe L2-SVM de optimizare a acestui algoritm.

Metoda consta in aplicarea mai multor iterații asemenea celei descrise mai jos, pornind cu un :

Iterația :

pentru i = 2 la N:

=

, unde

// calculam pentru a fi folosit la pasul următor

Funcția f descrisa mai sus face calcule numai in cazul in care e nevoie. De fiecare data când se folosește acest f (o data per integrație mica, N ori per iterație mare) se calculează un gradient pe componenta curenta (i din iterația mica). Daca proiecția gradientului este 0 (dreapta d=0 este optimul ecuației ), rezulta ca nu mai trebuie updatata componenta i din si suntem scutiți de calcule adiționale pentru aceasta iterație, de aici optimizarea.

Proiecția gradientului se calculează astfel: