Universitat Oberta de Catalunya

Segon cicle d'enginyeria informàtica

Projecte fi de carrera

Algorisme de hashing basat en hiperplans separadors localment òptims

Autor:
Joan Camps Morey

Consultor:
David Masip Rodo

4 de gener de 2015

Índex

Índex de figures

Índex de taules

1 Introducció

Des de que vaig iniciar el segon cicle en Enginyeria d'informàtica vaig decidir clar que el projecte fi de carrera el volia orientar cap a la recerca. Els motius que vaig tenir en compte a l'hora de prendre aquesta decisió foren, per una banda, que al haver de compaginar els estudis amb el món laboral resultaria molt pesat acabar la jornada laboral i arribar a casa per a continuar desenvolupant quelcom que ja he fet vuit hores aquell mateix dia, mentre que dedicar les hores destinades a acabar el projecte fi de carrera a aprendre coses noves seria un al·licient afegit. Per altra banda, donat que la finalitat dels treballs de recerca és la concepció o creació de nous coneixements, productes, processos o mètodes, sempre m'ha atret la idea de poder-hi participar, per tal d'aportar el meu granet de sorra, per petit que aquest sigui.

Un cop presa aquesta decisió era necessari escollir un camp i un projecte en particular. L'elecció del camp va aparèixer després de cursar les dues assignatures d'intel·ligència artificial del pla d'estudis d'enginyeria d'informàtica. Vaig passar-ho molt be mentre estudiava i feia les pràctiques d'aquestes dues assignatures. L'elecció del projecte en particular va venir després de parlar amb el David Masip, qui em va donar l'opció de continuar un article[1] que havia publicat juntament amb l'Oriol Pujol temps enrere, que té per títol *Conjunts basats en geometria: Cap a una caracterització estructural de la frontera de classificació**, i a continuació faré un resum dels punts més importants que en ell s'expliquen.

1.1 Conjunts basats en geometria: Cap a una caracterització estructural de la frontera de classificació

L'article presenta un algorisme d'aprenentatge supervisat per a la resolució de problemes de classificació binària. Aquest algorisme tracta d'aproximar una frontera de decisió no lineal mitjançant un model additiu lineal per trams, on la frontera de decisió, frontera a partir de la quals es classifiquen els elements en les dues classes, és definida geomètricament per mitjà dels *punts característics de la frontera*†. Hi ha diversos algorismes de classificació amb un component eminentment geomètric a darrera, com per exemple, les màquines de vectors de suport. Aquesta tècnica *classifica els objectes basant-se en la construcció d'un hiperplà en l'espai de les dades que separa els objectes que pertanyen a una classe dels que pertanyen a l'altra*[2]. Tot i que aquest concepte és molt senzill d'entendre en el cas on la frontera és un únic hiperplà, es torna molt més complexa quan la frontera és no lineal‡. En aquests casos, l'estratègia més coneguda és la de *transformar l'espai de les dades original (un espai M dimensional, atès que considerem M atributs numèrics) en un nou espai diferent i més gran, i en*

* *Geometry-Based Ensembles: Toward a Structural Characterization of the Classification Boundary* , en anglès.

† En anglès, *Characterizing Boundary Points*, CBP's

‡ com és el cas de les màquines de vectors de suport no lineals

la separació lineal dels elements en aquest nou espai[2]. Aquesta tècnica no és suficient per a definir una frontera de classificació no lineal de forma exacta, atès que, degut als punts que ens serveixen per a definir la frontera però que en realitat estan lluny d'aquesta, s'introdueix un grau d'incertesa al model molt elevat. A la figura 1a es poden veure aquests punts, i la incertesa que s'introdueix al model, representada per la zona ombrejada. Tots els elements que es trobin a dins aquesta àrea ombrejada seran susceptibles de ser classificats erròniament.

(a) Amb àrea d'incertesa (b) Sense àrea d'incertesa

Figura 1: Fronteres no lineals

Per tal d'eliminar aquest problema, a l'article s'introdueix el concepte de CBP. Els CBP són, conceptualment, punts que pertanyen a la frontera només sota unes certes condicions de robustesa i marge. Això vol dir, que per a cada punt tenim un marge òptim localment. La unió d'aquests CBP ens donen una frontera sense àrea d'incertesa, tal i com es pot veure a la figura 1b. A l'article, aquesta frontera s'aproxima mitjançant l'ús d'una funció lineal per trams, és a dir, cadascun d'aquests trams ens defineix un hiperplà, que és localment òptim, donat que s'aconsegueix a partir del marge localment òptim, que podrem usar per a classificar els punts del conjunt de dades a classificar.

Punts característics de la frontera

Abans de veure com podem obtenir els CBP els definirem formalment. Per això, suposarem que tenim un conjunt d'entrenament S, de N elements x_i, etiquetats amb la classe l_i a la que pertanyen, i ho definirem formalment de la següent forma

$$S = \{(x_i, l_i)\},\ \text{on } x_i \in \mathbb{R}^d,\ \text{i } l_i = l(x_i) \in \{-1, +1\} \text{ amb } i = 1..N$$

llavors, un *punt característic de la frontera* $x_{cbp}^{i,j} \in \mathbb{R}^d$ queda definit per dos punts del conjunt d'entrenament (x_i, x_j) sota les següents condicions

- **Condició necessària**. Els dos punts pertanyen a classes diferents

$$x_i \in S \mid l_i = +1 \text{ mentre que } x_j \in S \mid l_j = -1$$

- **Condició de proximitat a la frontera òptima**. No existeix cap altre punt més proper al candidat a CBP, $x_(cbp)$ que aquells dos que l'han definit, és a dir (X_i, X_j).

  Formalment, $\forall p : \{p \in \mathbb{R}^d \mid (p, l) \in S\}$, es compleix que

$$\left\|x_i - x_{cbp}\right\| \leq \left\|p - x_{cbp}\right\|, \tag{1}$$
$$\left\|x_j - x_{cbp}\right\| \leq \left\|p - x_{cbp}\right\|. \tag{2}$$

- **Condició de robustesa al marge**. Suposant que no disposem de cap tipus d'informació geomètrica sobre el marge, el candidat a CBP, x_{cbp}, ha d'estar situat a la màxima distància dels dos punts que el defineixen, X_i i x_j, és a dir, és el punt mig entre X_i i x_j.

$$x_{cbp}^{i,j} = \frac{(x_i + x_j)}{2} \tag{3}$$

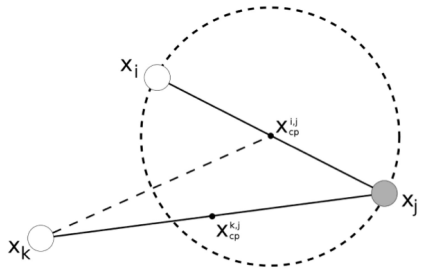

Figura 2: Imatge de la definició de CBP

La *condició de robustesa al marge* és la que assegura que els CBP acabin definint hiperplans que són localment òptims. Així mateix, defineixen una àrea de influència a cadascun dels dos punts que defineixen el CBP. Això es veu més intuitivament a la figura 2. La parella $(x_i + x_j)$ defineix $x_{cbp}^{i,j}$, donat que no existeix cap altre punt a dins l'hiperesfera definida per $x_{cbp}^{i,j}$ amb un ràdius inferior a $\left\|x_i - x_j\right\|/2$. Definits formalment els CBP, podem passar a veure quin és l'algorisme proposat a l'article per a calcular-los.

Algorisme per al càlcul dels CBP

A la secció anterior hem vist com definir formalment els CBP, ara veurem com els podem obtenir a partir del conjunt de dades original. Suposem que tenim $S = \{(x_i, l_i)\}$, on $x_i \in \mathbb{R}^d$, i $l_i = l(x_i) \in \{-1, +1\}$ amb $i = 1..N$, és a dir, un conjunt de punts etiquetats amb la classe a la qual pertanyen. El que volem és trobar els parells d'elements formats per punts d'una classe i l'altra que ens defineixen un CBP, és a dir, un conjunt $P = \{i, j\}$ on i i j corresponen als

índexs dels elements de S, on $x_i \in l(+1)$ i $x_j \in l(-1)$.

El mètode per obtenir els índexs seria el següent. Primer, i per tal d'evitar haver de calcular la distància entre tots els punts a cada iteració, podem calcular la matriu de distàncies Euclidianes entre tots els elements de S. Després, avaluem totes les parelles formades pels elements x_i i x_j d'una classe i l'altra, cercant si hi ha algun punt x_k més proper al punt central del segment que uneix x_i i x_j. L'algorisme per fer això es proporciona a l'article original en pseudocodi, mentre que la versió ja codificada en llenguatge Matlab© també esta disponible per a baixar[§], i es pot consultar a l'apèndix i. La versió en pseudocodi és la següent.

Input: $S = \{(x_i, l_i)\}$, where $x_i \in \mathbb{R}^d$, and $l_i = l(x_i) \in \{-1, +1\}$ with $i = 1..N$
Output: $P = \{i, j\}$ where (x_i, x_j) defines a CBP
    $D(SxS) \leftarrow$ Distances within S elements
    $P \leftarrow \varnothing$
    for all $x_i | l(x_i) = +1$ **do**
        for all $x_j | l(x_j) = -1$ **do**
            for all $x_k | x_k \in S$ **do**
                $d_{k,m} \leftarrow$ distance from x_k to central point $x_m = \frac{(x_i + x_j)}{2}$
                if $\nexists k | d_{k,m} < \frac{D(i,j)}{2}$ **then**
                    $P \leftarrow P \cup (i, j)$
                end if
            end for
        end for
    end for

L'algorisme anterior ens retorna un conjunt d'índexs i i j tals que x_i, x_j defineix $x_{cbp}^{i,j}$ que és un CBP. A més, cada $x_{cbp}^{i,j}$ ens defineix un hiperplà que separa de forma localment òptima els elements propers, i, com ja he dit, l'article usa aquests hiperplans per a classificar els elements. A la següent secció explicaré com.

Classificació dels elements

Per a saber la classe a la que pertany un element el que es fa és aplicar un model aditiu per trams. Anem a veure com es fa. Primer, tenim els CBP, que hem definit de la següent forma

$$x_{cbp}^{i,j} = \frac{x_i + x_j}{2}$$

a més, sabem que cada $x_{cbp}^{i,j}$ ens defineix un hiperplà, del quan en podem calcular el vector normal, $\vec{n}_{x_{cbp}^{i,j}}$, així

$$\vec{n}_{x_{cbp}^{i,j}} = \frac{x_i - x_j}{||x_i - x_j||}$$

[§]Es pot baixar des de la web *http://www.maia.ub.es/∼oriol/Personal/downloads.html*

atès que per definició el vector normal té un valor positiu, podem definir un classificador local de la següent forma

$$\pi_{x_{cbp}^{i,j}}(x) = (x - x_{cbp}^{i,j})\vec{n}_{i,j}$$

un valor positiu de $\pi_{x_{cbp}^{i,j}}$ ens indicarà que, per a l'hiperplà definir per $x_{cbp}^{i,j}$ la l'element x pertany a la classe $+1$, mentre que un valor negatiu ens indicarà que pertany a la classe -1. Però això només és per a un hiperplà en particular, i nosaltres necessitem el model complet, per tant, el que es fa és calcular $\prod(x)$, de la següent forma,

$$\prod(x) = \sum_{k=1}^{N} \alpha_k sign(\pi_k(x))$$

així apliquem de forma additiva els valors que ens retornen tots els hiperplans, i per tant, la classe $\hat{l}$ definitiva que assignarem al nostre element es pot calcular com

$$\hat{l} = sign(\prod(x) - \alpha_0)$$

els valors α_0 i α_k son valors que ens ajuden a mesurar el pes que ha de tenir cada hiperplà en el resultat local, i en el final. Es calcula usant la *regularització de Tikhonov*, però per a l'objectiu d'aquest projecte no és necessari veure com es calculen aquests dos valors, donat que no els usaré.

Amb això finalitza l'explicació del funcionament de l'algorisme de l'article en que es basa aquest projecte. Ja sabem com es classifiquen els elements. Totes les passes del procés es poden resumir de la següent forma:

- Calculem els CBP sobre el conjunt d'entrenament,

$$\{x_{cbp}^{i,j}\}, \forall i, j \in 1..M, i \neq j$$

- Calculem el conjunt dels classificadors en base als CBP

$$\pi_{x_{cbp}^{i,j}}(x) = (x - x_{cbp}^{i,j})\vec{n}_{i,j}, x_{cbp}^{i,j} = \frac{x_i + x_j}{2}, \vec{n}_{x_{cbp}^{i,j}} = \frac{x_i - x_j}{||x_i - x_j||}$$

- Avaluem els classificadors base amb el conjunt de dades per a trobar la matriu A

$$A(k,i) = sign(\pi_k(x_i)), k \in \{1..|\{x_{cbp}\}|\}, i \in \{1..M\}$$

- Finalment es troba la solució λ per a trobar les α_k i α_0 de les que he parlat anteriorment, resolent aquesta equació

$$\alpha_\lambda = arg_\alpha min(||A\alpha - l||_2^2 + \lambda^2||(\alpha - \alpha^*)||_2^2)$$

1.2 Objectiu del projecte

Fins ara hem vist què són els CBP, els hem definit formalment, hem vist com obtenir-los algorítmicament i com usar-los per a construir un algorisme d'aprenentatge supervisat. Ara, en base a tots aquests conceptes definirem l'objectiu principal d'aquest projecte fi de carrera. Suposem que tenim un conjunt S de s valors $(s_1, s_2, ..., s_m)$, etiquetats amb la classe binària a la que pertanyen $(-1, +1)$. Suposem ara, que hi apliquem l'algorisme de càlcul dels CBP, i ens retorna un conjunt P de p punts que satisfan les condicions per a ser CBP. Aquests p punts ens defineixen, a l'hora, p hiperplans, que anomenarem $H = \{h_1..h_p\}$. Llavors, podem usar la projecció de cadascun dels nostres s punts del conjunt d'entrada S sobre cadascun dels p hiperplans de H, i usar la mateixa funció de signe per a comprovar a quin costat de l'hiperplà queda el punt que estem tractant. Si assignem els valors 1 i 0 als valors positius i negatius de la funció de signe, el que obtindrem és una matriu binària $M(s \cdot p)$ on cada valor ens indicarà si aquell punt queda a un costat o l'altra de l'hiperplà, és a dir, per a cada punt obtindrem una cadena binària de p bits que ens classifiquen aquell valor per a cada hiperplà. L'objectiu principal d'aquest projecte és *dissenyar un algorisme de hashing que ens permeti escollir el conjunt N de n hiperplans que millor ens classifiquin el conjunt de dades original* minimitzant la pèrdua de capacitat de classificació.

Algorisme de hashing basat en hiperplans separadors localment òptims

Suposem que tenim un conjunt de dades format per quatre punts (A, B, C, D). Suposem que apliquem l'algorisme de càlcul dels CBP a aquest conjunt i que ens dóna com a resultat dos únics CBP, que ens donen a la vegada dos hiperplans h_0 i h_1, tal i com es mostra a la figura 3.

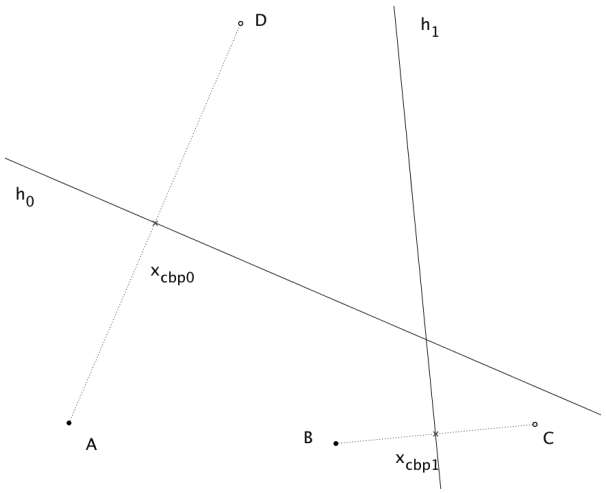

Figura 3: Representació del conjunt S d'entrada

Després, tal i com es veu a les figures 4,5,6 i 7, projectem els punts que tenim sobre els hiperplans obtinguts, i apliquem la funció de signe sobre el vector normal resultant, codificant els valors positius i negatius com a 0 i 1 respectivament (o a l'inrevés, la codificació escollida no és rellevant).

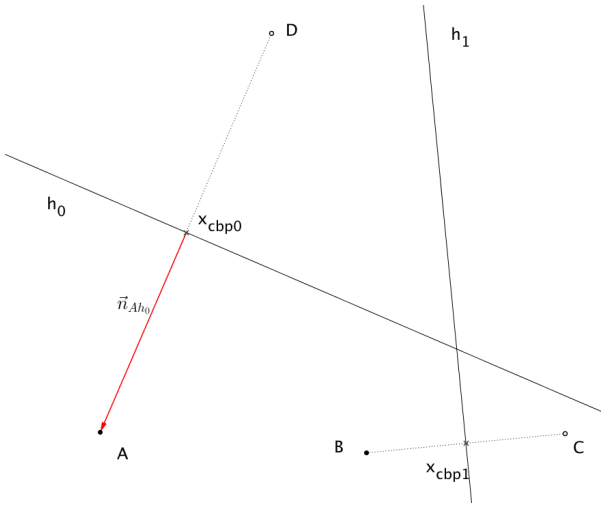

Figura 4: Projecció del punt A sobre l'hiperplà h_0

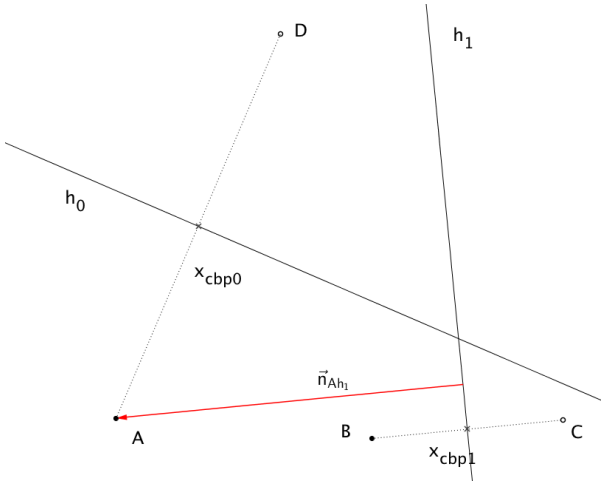

Figura 5: Projecció del punt A sobre l'hiperplà h_1

Com es pot veure pel cas del punt D projectat sobre l'hiperplà h_1, no tots els punts es classifiquen correctament, atès que els hiperplans sòn òptims de forma local, no global. Podem fer una taula amb els resultats obtinguts per a cada cas, i podem comparar-lo amb la seva classe, que coneixem.

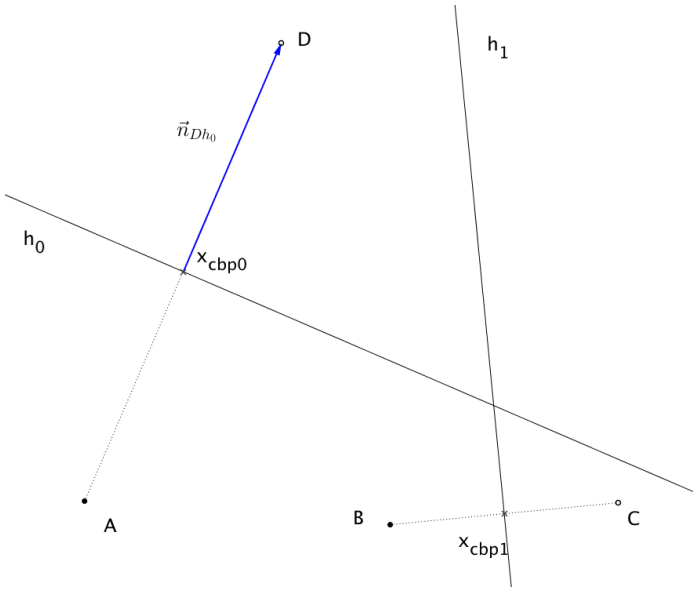

Figura 6: Projecció del punt D sobre l'hiperplà h_0

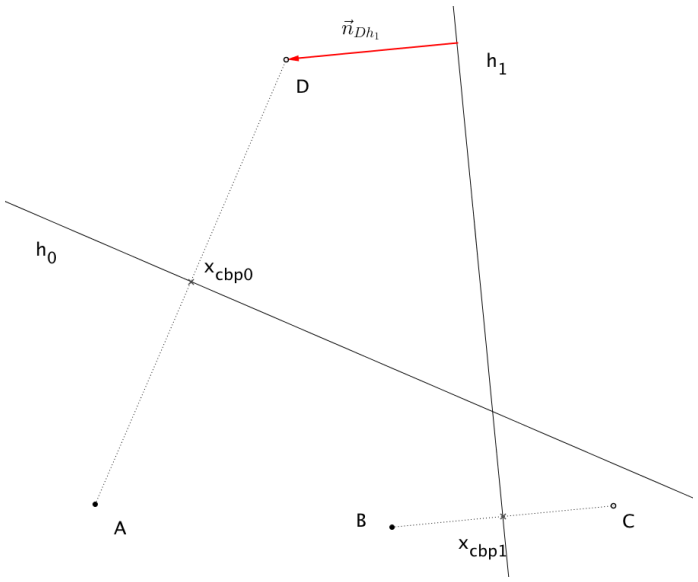

Figura 7: Projecció del punt D sobre l'hiperplà h_1

Es pot veure com en aquest cas, molt senzill, si ens fixem a la Taula 1, es veu com obtenim un vector de 2 bits per a cada valor del conjunt d'entrada que és el resultat de la classificació d'aquest envers cada hiperplà, si ho mirem per files, o

vist des d'un altre punt de vista, obtenim un vector de 4 bits, com a resultat de classificar[¶] tots els punts amb un mateix hiperplà, si ho mirem per columnes.

	h_0	h_1	Classe coneguda
A	1	1	1
B	1	1	1
C	1	0	0
D	0	1	0

Taula 1: Exemple simple de vectors locals de classificació

Doncs bé, si agafem aquest segon punt de vista, podem esbrinar la tasa d'encert que té un hiperplà, comparant els valors de classificació obtinguts, representat a les columnes, amb els valors reals de cadascun dels punts, valor del qual disposem donat que és una condició d'entrada. Per exemple, la tasa d'encert dels hiperplans h_0 i h_1 és de 3/4. La primera tasca a realitzar serà la de dissenyar un algorisme de classificació unificat, és a dir, com usar els dos hiperplans per a millorar la tasa d'encert. Aquest punt es resoldrà molt probablement amb una funció que ens retorni la distància entre dos vectors binaris, per exemple, la *distància Hamming*[‖], tot i que encara està per decidir.
Fins aquí hem vist un cas molt senzill per tal de facilitar la tasca de representar-ho gràficament. Ara podem veure un cas més complexe. Suposem que tenim un cas, on el conjunt d'entrada està format per n valors $(v_1, ..., v_n)$. Suposem que l'algorisme d'obtenció dels CBP ens retorna un conjunt de p valors, on cadascun defineix un hiperplà localment òptim $(h_1, ..., h_p)$. Podem obtenir igualment la matriu binària i obtindrem una taula com la següent.

	h_1	h_2	h_3	...	h_p	Classe coneguda
v_1	1	0	1	...	1	1
v_2	1	0	1	...	1	0
v_3	0	1	1	...	1	0
...	...	...	...	...	1	1
v_n	1	1	1	...	1	0

Taula 2: Valors de les projeccions de cada punt sobre cada hiperplà

Un cop arribats a aquest punt, la segona cosa que hem de fer, i és la que en realitat dóna nom al projecte, és escollir els n hiperplans, la combinació dels quals ens doni la major tasa d'encert de classificació. És a dir, aplicar hashing als vectors obtinguts a les files. És evident que, al ser n definit per l'usuari, és variable. A més, també es fàcil veure que si $n = 1$ el que obtindrem és un classificador lineal, i senzillament haurem de retorar l'hiperplà amb una tasa d'encert major. Igualment, si $n \geq p$ el que haurem de retornar són tots els

[¶]Per a simplificar les taules, uso la codificació de classe +1 amb l'1 binari i la classe -1 amb el 0 binari.

[‖]Per a més informació, es pot visitar *http://en.wikipedia.org/wiki/Hamming_distance*

hiperplans, i la tasa d'encert no tindrà cap pèrdua respecte l'original, atès que no es perd cap classificador local. És a dir, suposem que donada la taula 2, hem de triar els 4 hiperplans que combinats millor resultat de classificació ens donin. Per exemple, podríem escollir els resaltats a la taula 3

	h_1	h_2	h_3	h_4	...	h_{p-1}	h_p	Classe coneguda
v_1	1	0	1	1	...	1	1	1
v_2	1	0	1	1	...	1	1	0
v_3	0	1	1	0	...	1	0	0
...	...	...	...	...	...	...	1	
v_n	1	1	1	0	...	1	0	0

Taula 3: Hiperplans escollits com a millors classificadors combinats

En el cas anterior, el nostre algorisme haurà de retornar el conjunt

$$P = \{h_1, h_2, h_4, h_{p-1}\}$$

Per tant, haurem de proposar un algorisme per a realitzar aquesta tria. Existeixen diverses opcions i les haurem d'avaluar totes, des d'usar la tècnica del veï més proper passant per la cerca exhaustiva, o fins i tot, podem pensar un nou algorisme per a fer aquesta tria que no impliqui haver de calcular els resultats de classificació de totes les combinacions possibles.

1.3 Justificació del projecte

Tot i que els resultats de les classificacions de conjunts de dades amb el mètode anterior són molt bons, en la majoria dels casos el nombre de CBP's generats és mot alt, i això és deu bàsicament a dos problemes.

- **CBPs redundants**. Anomenaré així a tots aquells CBP's que es creen però que en realitat no aporten informació addicional a l'hora de classificar les dades. Un exemple és el següent conjunt de dades. Si suposem un conjunt de dades format per elements de dues classes D i I, enumerats com $D_0, ..., D_3$ i $I_0, ..., I_3$, respectivament, podrem veure com, aplicant l'algorisme d'obtenició dels CBP's original, obtenim els CBPs que es mostren a la figura 8. Tots aquests CBPs s'usaràn posteriorment per a realitzar les classificacions, quan en realitat, amb un únic CBP (amb qualsevol d'ells), com es pot veure a la figura 9 obtindrem el mateix grau d'eficiència en la classificació, tot i que amb l'algorisme original el càlcul es realitzarà amb tots els punts obtinguts, amb la càrrega addicional de càlcul que això suposa.

- **Outliers**. Anomenaré així a tots els punts que tot i ser d'una classe, es troben immersos enmig d'un núvol de punts de l'altre classe. Suposem un altre conjunt de dades com el de la figura 10. Si calculem els CBPs amb el mètode explicat anteriorment, els CBPs obtinguts quedaríen representats

a la figura 11. Com es pot veure el punt I_1 ens genera molts de CBPs que en realitat empitjoraran la capacitat de classificació del model.

Per tant hi ha la necessitat de reduïr el nombre de CBPs del model, i que aquesta reducció afecti, majoritàriament, a punts d'aquests dos tipus.

1.4 Descripció de les tasques a realitzar

A més de les tasques que s'han de dur a terme per tal de desenvolupar l'algorisme de hashing, també s'hauran de desenvolupar tasques per a l'avaluació dels resultats obtinguts. Per tant, el llistat sencer de les tasques a realitzar, incloses les que ja s'han dut a terme, són les següents.

1. *Comprensió de l'article original i de la feina a realitzar.* Abans de poder realitzar una descripció detallada tant de les tasques a realitzar així com fer una estimació temporal del temps que dedicaré a cadascuna, hauré d'entendre l'article original, per tant, a aquesta serà la primera tasca.
2. *Tasques prèvies.* Aquesta tasca està destinada a prendre decisions del tipus d'entorn que usaré pel desenvolupament o a triar el llenguatge de programació escollit per al desenvolupament del projecte. 3. *Definir l'algorisme.* En aquest punt haurem de definir les entrades i sortides de l'algorisme, calcular els CBP's, calcular la binarització per a cada mostra d'entrenament, i definir l'algorisme que cerqui la millor combinació de CBP's per a fer una binarització a partir de l'entrenament.
4. *Definir un protocol per a validar la proposta.* Cercaré bases de dades per a provar l'eficiència de l'algorisme, i definir un protocol de test.
5. *Redactar el projecte i preparar tots els medis a presentar.* A aquesta darrera tasca hauria de posar en clar totes les conclusions obtingudes per tal de fer la presentació final.

1.5 Planificació de les tasques a realitzar

Tenint en compte les tasques a realitzar i les dates de les entregues de la segona i tercera prova d'avaluació continuades, proposo la següent planificació.

1. *Comprensió de l'article original i de la feina a realitzar.* Aquesta tasca hauria d'estar acabada abans del 20 d'octubre.
2. *Tasques prèvies.* Aquesta tasca és ràpida, i espero no dedicar-hi més d'una setmana. 3. *Definir l'algorisme.* Crec que junt amb l'anàlisi dels resultats, la tasca més complexa és aquesta, per tant, espero dedicar-hi fins a la tercera entrega, que és a finals de novembre.
4. *Definir un protocol per a validar la proposta.* Per a aquesta tasca quinze dies haurien de ser suficients.
5. *Redactar el projecte i preparar tots els medis a presentar.* Finalment, fins a la darrera entrega quedaran 30 dies, que els dedicaré a la redacció del projecte i

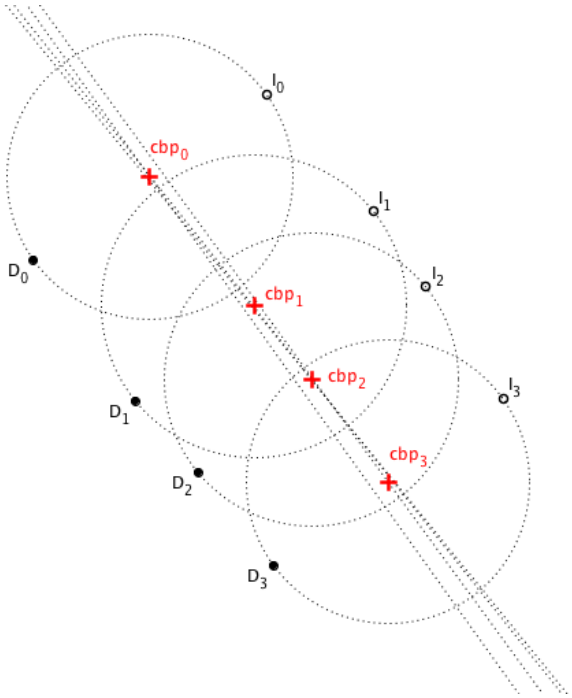

Figura 8: CBPs obtinguts pel conjunt de dades d'exemple

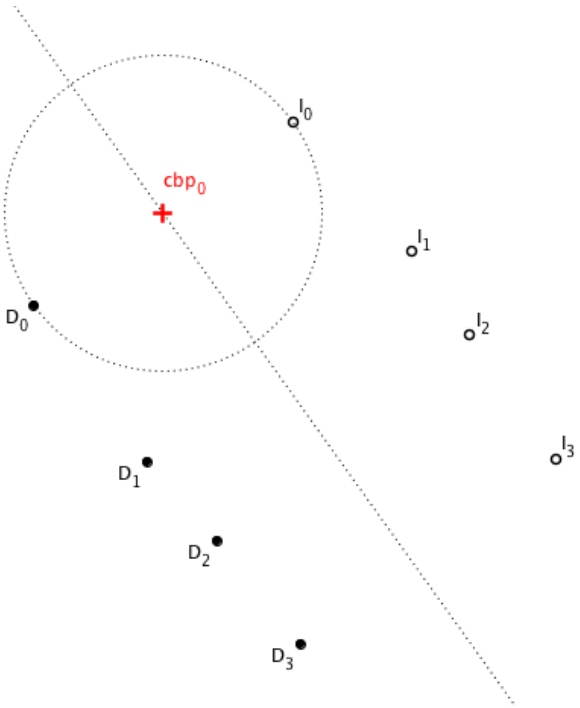

Figura 9: Amb un únic CBP obtindríem la mateixa eficiència

Figura 10: Conjunt de dades d'exemple

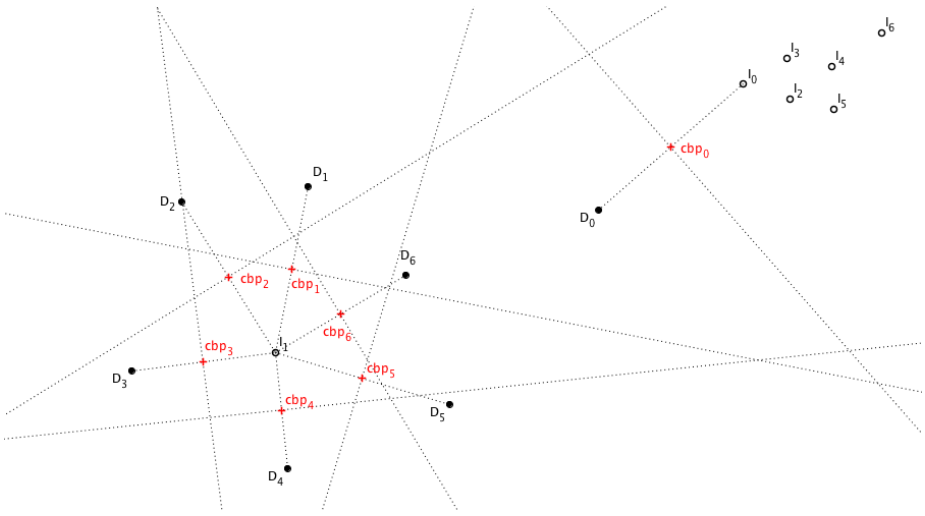

Figura 11: CBPs obtinguts amb el conjunt de dades

a preparar els medis necessaris per a la presentació final. Aquesta darrera fase també haurà d'absorbir els possibles retards en les altres fases.

A la figura 12 podem veure un diagrama Gantt amb aquesta programació.

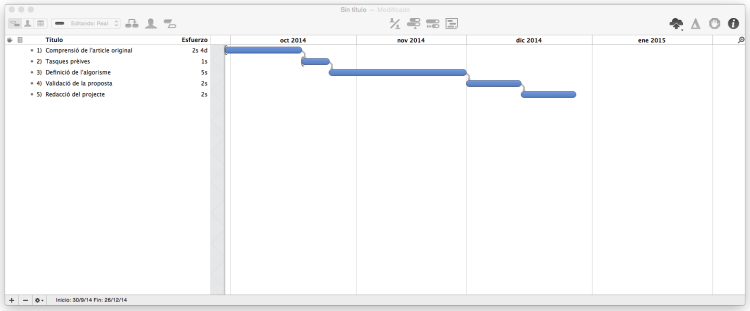

Figura 12: Diagrama Gantt amb la planificació temporal del desenvolupament del projecte

2 Desenvolupament del projecte

Al llarg d'aquesta secció aniré explicant les tasques que s'han dut a terme per tal d'assolir de forma satisfactòria els objectius del projecte definits a la secció anterior. Totes les decisions preses així com la seva justificació, els esdeveniments que s'han anat produint i els problemes que han sorgit, així com la solució que s'ha proposat, es troben a aquest apartat.

2.1 Tasques prèvies

Abans d'iniciar el desenvolupament del projecte he de prendre algunes decisions importants. A aquest apartat es detallen aquestes tasques.

Elecció del llenguatge de desenvolupament

Tot i que, com ja he comentat abans, una part important del codi esta disponible* en format Matlab® sota llincència GPL†, i per tant en podria fer ús, he decidit implementar de zero els algorismes amb el llenguatge Python®, usant els originals com a base. Avui en dia és una pràctica generalitzada entre els científics fer aquesta transició entre els dos llenguatges, i es pot trobar molta informació‡§¶‖ de per quins motius ho fan. En el meu cas els motius pels quals m'he decidit per usar Python® són els següents,

- Tot i que Matlab® és un llenguatge interpretat d'alt nivell fàcil de llegir i entendre, i resulta relativament senzill escriure-hi programes, no tinc gaire experiència en el seu ús, mentre que Python® és, igualment, un llenguatge interpretat d'alt nivell, fàcil de llegir i entendre, però amb el que tinc experiència, atès que l'uso cada dia durant l'exercici de la meva vida professional.
- Matlab® és de pagament. Això fa que per a desenvolupar-hi es necessita un intèrpret que és de pagament, i a més no és barat. Per contra, Python® i el seu intèrpret són completament gratuïts.
- El punt anterior complica la difusió del codi Matlab® atès que no tothom en té una llicència per a poder executar-lo, per contra, això no passa en Python®.
- El fet que Python® sigui de codi obert fa que en cas d'haver-hi un error d'execució o rendiment en algun dels seus mòduls, podem revisar-ne el codi per tal d'arreglar-ho sense necessitar d'un support tècnic que ho revisi per nosaltres. Això no passa amb Matlab®.

*Es pot baixar des de la web *http://www.maia.ub.es/∼oriol/Personal/downloads.html*

†Per a més detalls sobre aquesta llicència es pot visitar la web *http://ca.wikipedia.org/wiki/GNU_General_Public_License*

‡Per exemple, *https://stevetjoa.com/305/*

§Per exemple, *https://sites.google.com/site/pythonforscientists/python-vs-matlab*

¶Per exemple, *http://phillipmfeldman.org/Python/Advantages_of_Python_Over_Matlab.html*

‖Per exemple, *http://www.stat.washington.edu/ hoytak/blog/whypython.html*

- Moltes de les funcionalitats que usaré al llarg del projecte estan disponibles via llibreria de codi ober, en canvi, tot i que Matlab® també disposa de llibreries molt similars, i amb les que es pot fer pràcticament el mateix que amb les de Python®, moltes d'elles han d'adquirir-se com una llicència a part, amb el consegüent cost afegit.

Com a únics punts positius a favor de Matlab® podria dir que,

- Ja disposo de part del codi escrit en Matlab®, i això implica que l'hauré de tornar a escriure, amb la despesa de recursos temporals que això suposa.
- La instal·lació de l'entorn de Matlab® és més senzilla que la de Python®.

Exposats els arguments, és evident que els motius per a triar Python® són de pes i, a més, més nombrosos. Per tant, Python® és l'elegit.

Elecció de la plataforma de desenvolupament

Com a únic equip personal disposo d'un *Apple® MacBook Pro®*, per tant, és el que he triat, juntament amb el seu sistema operatiu, l'*OSX® Yosemite*. En quan a l'entorn de desenvolupament integrat tan sols es requereix un editor de text, però, tot i així, he escollit el *PyCharm® Community Edition*. Aquest entorn integrat de desenvolupament és gratuït i presenta les funcionalitats típiques dels entorns de desenvolupament, com puguin ser les funcions d'autocompletat o les d'execució i depuració des del propi entorn, cosa que amb els editors de text no passa.

2.2 Definició de l'algorisme

Com ja he explicat al llarg d'aquest document, la finalitat d'aquest projecte és la de dissenyar un algorisme de hashing per tal de reduïr el nombre de CBPs, atès que el cost computacional de la classificació amb tots els CBPs que s'obtenen amb l'algorisme original és molt elevat. Per tant, és una bona idea intentar reduïr aquest nombre de CBP's per tal que, un cop obtingut el model de classificació, el fet de classificar els nous punts sigui el més ràpid possible. Com ja hem vist abans, un cop reduït el nombre de CBPs, obtindrem un matriu de característiques que usarem per a classificar els nous elements usant el mètode del veï més proper (*knn*), és a dir, obtindrem les característiques del nou punt, calcularem la distancia als punts de la matriu de classificació i obtindrem per *knn* la classe de l'element.

A més, hem de tenir el compte que, com que el nombre de CBPs és molt elevat, cercar la combinació més eficient entre tots ells també és un problema amb un cost computacional molt elevat, i per tant, haurem de descartar la cerca exhaustiva com a mètode de cerca. Per a fer-ho per cerca exhaustiva hauriem de calcular, primer, totes les combinacions possibles, i, posteriorment, calcular l'eficiència de cadascun d'ells, per a finalment agafar aquesta millor combinació.

Ara presentaré l'algorisme, i posteriorment en calcularé el cost computacional i el compararé amb el cost computacional que tendria fer-ho per cerca exhaustiva.

Disseny de l'algorisme

En realitat per a classificar nous elements necessitem no només els CBPs, sino que també necessitem els vector normals de cadascun d'ells, i a més, la matriu de característiques. Amb això, un cop volem classificar un punt, primer en calcularem les seves característiques amb els CBPs i els seus vectors normals, és a dir, obtindrem un vector binari on cada bit ens indicarà si el punt queda a un costat o l'altra de cada hiperplà definit per cada DBP, i, posteriorment, usant *knn* farem la predicció de la classe a la que pertany, això és, cercarem els k elements més propers de la matriu de característiques al vector obtingut anteriorment, i assignarem al nou element la classe de la majoria dels k. Per a calcular l'algorisme *knn* s'usa la classe *NearestNeighbors* de la llibreria *SciKit learn*[**], tant a la fase d'entrenament (per a generar el model) com a la fase de classificació, amb la diferència que a la fase d'entrenament s'ha de calcular *knn* amb tots els valors del conjunt d'entrenament exceptuant aquell que estem validant, com és evident.

Així, tenim que en realitat haurem de fer la reducció no tan sols sobre els CBPs, sino que també sobre la matriu de característiques i sobre els vectors normals. De totes formes el que faig per a simplificar els procés, com explicaré més endavant, és fer una llista dels punts que vull mantenir, i retornar els valors de la matriu de característiques, dels CBPs i dels vectors normals, només dels elements d'aquesta llista.

[**]Per a consultar la documentació completa de la llibreria es pot fer a http://scikit-learn.org/stable/documentation.html

Suposem que tenim un conjunt de N punts característics de la frontera o CBPs que voler reduïr a un màxim de h. El pseudocodi de l'algorisme dissenyat seria el següent.

Input: $CBPs = \{set\ of\ CBPs\}$
Input: $h = desired\ maximum\ hash\ size$
$Hash \leftarrow$ CBP with maximum individual accuracy
while True **do**

if $CheckExitConditions()$ **then**
$exit()$
end if
# addition stage
while True **do**
for all $x_i \in CBPs$ **do**
$tmp\ accuracy \leftarrow accuracy(Hash \cup x_i)$
if $tmp\ accuracy > max\ accuracy$ **then**
$max\ accuracy = tmp\ accuracy$
$max\ item = x_i$
end if
end for
if $max\ accuracy > accuracy(Hash)$ **then**
$Hash = Hash \cup x_i$
end if
if $CheckAdditionStageExitConditions()$ **then**
$exit()$
end if
end while
# deletion stage
while True **do**
for all $x_i \in Hash$ **do**
$tmp\ accuracy \leftarrow accuracy(Hash - x_i)$
if $tmp\ accuracy > max\ accuracy$ **then**
$max\ accuracy = tmp\ accuracy$
$max\ item = x_i$
end if
end for
if $max\ accuracy > accuracy(Hash)$ **then**
$Hash = Hash - x_i$
end if
if $CheckDeletionStageExitConditions()$ **then**
$exit()$
end if
end while
end while

L'algorisme, com es pot veure al pseudocodi anterior, es descomposa en dues fases, una primera d'addició i una segona d'eliminació, que es repeteixen a dins un bucle, K vegades. A la fase d'addició, el que faig és agafar el CBP que té un grau de classificació major. A partir d'aquest vaig testejant totes les combinacions possibles amb cadascun dels altres punts d'N, i a cada iteració afegeixo al hash aquell CBP que millora el grau de classificació del conjunt, sempre que existeixi un CBP que en millori el grau, o sempre que no arribem al màxim nombre de CBPs que es desitja (i s'indica per paràmetre). Cada cop que s'afegeix un element al conjunt, es torna a reiniciar el procés amb el primer element del conjunt, és a dir, si tenim un conjunt

$$Hash = \{1, 3, 4\}$$

i afegim l'element $\{8\}$, el bucle torna a començar i comprova el nou conjunt

$$Hash = \{1, 3, 4, 8\}$$

amb tots els elements del conjunt de CBPs que no estan a dins $Hash$, és a dir, que acabada aquesta nova iteració podríem tenir el nou conjunt

$$Hash = \{1, 3, 4, 8, 2\}$$

on es veu que s'ha afefit l'element $\{2\}$. Al final d'aquesta primera fase, dels N CBPs ja n'hem redüit el nombre a h CBPs, com a màxim. A la fase d'eliminació faig l'invers de la fase anterior, és a dir, vaig llevant CBPs del conjunt de Hash creat a la fase d'addició, i comprovo si eliminant aquest CBP l'eficiència del conjunt augmenta. A cada iteració elimino el CBP amb l'eliminació del qual més s'incrementa el grau de classificació. Aquestes dues fases es repeteixen K vegades. Tot i que els bucles s'executen de forma contínua (hi ha el *While True*), les condicions de sortida es comproven a les funcions següents,

- **CheckExitConditions()**
- **CheckAdditionStageExitConditions()**
- **CheckDeletionStageExitConditions()**

Aquestes condicions es veuen més clares al codi font del programa, disponible a l'apèndix iii, i, a més, totes les condicions s'expliquen una mica més endavant. Tanmateix, amb un exemple quedarà més clar.

Suposem que tenim un conjunt de dades amb el que hem generat el següent conjunt de CBPs,

$$CBPS = \{1, 3, 4, 7, 9, 13, 17, 21, 45, 55, 64\}$$

en realitat, els CBPs són punts N-dimensionals, el conjunt CBP anterior només és per a fer la demostració. Suposem que $\alpha(H)$ és una funció que ens retorna l'eficiència de classificació d'un conjunt de CBPs. Llavors, inicialitzem el conjunt de CBPs reduït H amb l'element c de CBPS tal que es maximitza $\alpha(H)$. Suposem que aquest element és el 4, així que

$$H = \{4\}$$

ara, hem d'iterar tots els elements de N, afegint cada element de N a H, i comprovar si el nou conjunt té un grau de classificació major que H sense aquest element. En aquesta iteració no es té en compte els elements que ja formen part de H, evidentment. Al final d'aquesta iteració, afegim l'element de N que maximitzi $\alpha(H)$, i obtenim un nou conjunt H. Suposem que aquest nou conjunt és

$$H = \{4, 17\}$$

Ara es tornen a afegir tots els elements de N, un a un, a H, i es comprova $\alpha(H)$ cada vegada. Al final tornem a afegir el valor que millora aquesta combinació. Després de repetir aquest procés vàries vegades, suposem que arribem a

$$H = \{4, 17, 45, 55\}$$

Aquesta primera fase s'acaba quan es compleixen alguna de les següents condicions

- **Condició de màxim grau de classificació**. Si l'algorisme troba un H tal que $\alpha(H) = 1.0$ està clar que no podem millorar el grau de classificació, per tant, podem finalitzar el procés. Si es dóna aquesta condició l'algorisme finalitza fins i tot si H no té la mida que s'ha passat per paràmetre, és a dir, es prioritza el grau d'eficiència per damunt de la mida del hash.
- **Condició de mida de hash màxim**. Si arribem a la mida màxima de hash sol·licitada, l'algorisme surt d'aquesta primera fase. És possible que a la segona fase s'eliminin elements, per tant, aquesta és condició només per sortir d'aquesta fase, no de l'algorisme.
- **Condició de nombre d'elements d'**N. Sortim d'aquesta fase en haver comprovat tots els elements.
- **Condició de millora de d'**$\alpha(H)$. Sortim d'aquesta fase si no trobem cap element que millori $\alpha(H)$.

Un cop hem sortit de la fase d'addició, iniciem la fase d'eliminació, per tant, del conjunt

$$H = \{4, 17, 45, 55\}$$

llevem el primer element, i en calculem $\alpha(H)$, és a dir,

$$H = \{17, 45, 55\}$$

fem el mateix pel següent element, ara

$$H = \{4, 45, 55\}$$

al final d'aquesta iteració, si existeix un element de H tal que si l'eliminem del conjunt ens dóna que $\alpha(H') > \alpha(H)$, l'elimininem del conjunt i iniciem el procés pel nou conjunt, amb l'element anterior eliminat. Suposem que al finalitzar aquesta fase ens queda el següent conjunt

$$H = \{4, 17, 55\}$$

La fase d'eliminació acaba quan es donen una de les següents condicions,

- **Condició de nombre d'elements d'**N. Sortim d'aquesta fase en haver comprovat tots els elements.
- **Condició de millora de d'**$\alpha(H)$. Sortim d'aquesta fase si no trobem cap element que millori $\alpha(H)$.

Aquest procés (addició + eliminació) es repeteixen k vegades, de forma que al finalitzar cada iteració el conjunt obtingut ha de complir que

$$\alpha(H') > \alpha(H)$$

on H' representa el nou conjunt. Aquest és l'algorisme disenyat, i tot i que el seu cost computacional, com veurem més endavant és elevat, és menys costos que la cerca exhaustiva.

Implementació en Python® de l'algorisme

A continuació, i tot i que està també a l'apèndix iii, mostro el codi Python de l'algorisme.

```
def hash_features(features,
                  cbps,
                  nvects,
                  labels,
                  hash_max_size):
    """
    This function returns a reduced set of cbps, a set of features
    and its normals vectors, based on the originals sets computed
    by compute_cbps function. hash_max_size only works as a limiter
    of the hash size. If the algorithm find a hashed set lower in
    size but with a better classification index, it returns the
    small one. The accuracy has priority on hash size.
    :param features: array representing features for each sample
    computed with compute_oge function
    :param cbps: array of cbps computed by compute_cbps function
    :param nvects: array of normal vectors, one for each feature
    :param labels: classes of each sample
    :param hash_max_size: maximum size of the hash
    :return: returns a reduced set of cbps, a set of features and
    its normals vectors, based on the originals sets computed by
    compute_cbps function
    """

    #Initial setup of local variables
    max_iterations = 10
    iterations = 0
    features = features.transpose()
    n_cbps = features.shape[0]
    last_iteration_accuracy = -1
    item_deleted = True

    # compute accuraccies of each cbp to get the maximum value
    accuracies = [compute_accuracy(features[i, :],
                  labels) for i in range(0, n_cbps)]
    i = accuracies.index(max(accuracies))

    # initialize hash with the cbp that has the maximum
  #individual accuracy
    hashed_cbps, hashed_accuracy = [i], accuracies[i]

    while True:

        """
        Check exit conditions
        """
        # maximum number of iterations reached
        if iterations == max_iterations:
            break
        # accuracy of 1.0 reached
        if hashed_accuracy == 1.0:
            break
        # check if accuracy has improved
        if hashed_accuracy > last_iteration_accuracy:
```

```
        last_iteration_accuracy = hashed_accuracy
    else:
        break
    # check if some item has been deleted in last iteration
    if not item_deleted:
        #print "Element not deleted. Break"
        break

    """
    Stage 1. Add only cbp's that improves the prediction accuracy
    """
    while True:
        max_accuracy = -1.
        max_i = -1

        for i in range(n_cbps):
            if i not in hashed_cbps:
                tmp_hashed_cbps = hashed_cbps + [i]
                prediction = compute_knn_classification(
             features[tmp_hashed_cbps].transpose(),
             labels)
                tmp_accuracy = compute_accuracy(prediction,
                       labels)

                if tmp_accuracy > max_accuracy:
                    max_accuracy = tmp_accuracy
                    max_i = i

        if max_accuracy > hashed_accuracy:
            hashed_cbps = hashed_cbps + [max_i]
            hashed_accuracy = max_accuracy

        # exit condition 1 for stage 1
        else:
            break
        # exit condition 2 for stage 1
        if len(hashed_cbps) >= hash_max_size:
            break

    """
    Stage 2. Remove cbps only if deletion improves
    the prediction accuracy
    """
    while True:

        # exit condition 1 for stage 2
        if len(hashed_cbps) < 2:
            break

        max_accuracy = -1.
        max_i = -1
        item_deleted = False

        for i in hashed_cbps:
            tmp_hashed_cbps = hashed_cbps[:]
            tmp_hashed_cbps.remove(i)
```

```
                prediction = compute_knn_classification(
                                features[tmp_hashed_cbps].transpose(),
            labels)
                tmp_accuracy = compute_accuracy(prediction, labels)

                if tmp_accuracy > max_accuracy:
                        max_accuracy = tmp_accuracy
                        max_i = i

            if max_accuracy > hashed_accuracy:
                item_deleted = True
                hashed_cbps.remove(max_i)
                hashed_accuracy = max_accuracy

             # exit condition 2 for stage 2
            else:
                break

        iterations += 1

    return features[hashed_cbps].transpose(),
           cbps[hashed_cbps],
           nvects[hashed_cbps]
```

Codi font de l'algorisme de hash dels CBP

Cost computacional de l'algorisme

El cost computacional de l'algorisme dissenyat, sobre un conjunt de N elements, i amb una mida de hash màxima de h elements, és pot expressar com

$$\zeta(N,h) = k(N^2 + h^2)$$

de la qual cosa, podem deduïr que per a valors molt grans de h, o fins i tot iguals a N, el cost seria

$$\zeta(N,h) = 2kN^2$$

mentre que per a valors petits de h, que serà el cas més habitual, el cost es podria expressar com a

$$\zeta(N,h) = kN^2$$

tot i així s'ha de tenir en compte que el grau d'eficiència millorarà per a valors majors de h. Per a comparar el cost amb el de la cerca exhaustiva agafaré el pitjos dels dos casos, el cas on $h = N$. Per aquest cas, hem vist que

$$\zeta(N,h) = 2kN^2$$

En el cas de la cerca exhaustiva, es tracta d'un problema de combinatòria, per tant, tenim combinacions de h elements sobre un conjunt de N elements, sense repeticions, així que el cost d'aquest algorisme el podríem expressar com

$$\zeta(N,h) = \frac{N!}{(N-p)!}$$

per tant, per a un cas concret on les dades inicials ens ha donat un conjunt de 360 CBPs (un cas real dels conjunts de dades que he usat) i tenint en compte que volem reduir a 8 valors aquests CBPs, tenim $N = 350$ i $h = 8$, per tant, per a la cerca exhaustiva, tenim

$$\zeta(N,h) = \frac{N!}{(N-p)!} = \frac{360!}{(360-8)!} = 260858210422628246400$$

per tant, obtenim 260858210422628246400 iteracions, mentre que pel pitjor cas d'hcbp, suposant que $k = 10$, tenim que

$$\zeta(N,h) = 2kN^2 = 20 * 360^2 = 2592000$$

és a dir, 2592000 iteracions. Es pot veure que, tot i que el cost computacional és molt elevat, és molts ordres de magnitut més petits que en el cas de la cerca exhaustiva.

2.3 Traducció dels algorismes de Matlab® a Python®

Obtenció dels CBPs

Recordem quin era l'algorisme per a obtenir els CBPs.

Input: $S = \{(x_i, l_i)\}$, where $x_i \in \mathbb{R}^d$, and $l_i = l(x_i) \in \{-1, +1\}$ with $i = 1..N$
Output: $P = \{i, j\}$ where (x_i, x_j) defines a CBP

```
D(SxS) ← Distances within S elements
P ← ∅
for all x_i|l(x_i) = +1 do
    for all x_j|l(x_j) = −1 do
        for all x_k|x_k ∈ S do
            d_{k,m} ← distance from x_k to central point x_m = (x_i + x_j)/2
            if ∄k|d_{k,m} < D(i,j)/2 then
                P ← P ∪ (i, j)
            end if
        end for
    end for
end for
```

La implementació d'aquest en Python® queda de la següent forma,

```
def compute_cbps(data, labels):
    """
    Returns a set of characterizing boundary points computed
  using Gabriel neighborhood rule. This is an adaptation to
  python language of the beta version of the Optimized
  Geometry-based Ensemble basic classifier, copyright 2008,
  2009 by Oriol Pujol and David Masip.

    :arg ds: Dataset containing the data
    :return: List of pair of characterizing boundary points
    and its normal vector
    """
    idxs, cbps, nvects, ones, zeros = [], [], [], [], []
    d = compute_distance_matrix(data, data)
    d2 = d ** 2

    for i in range(0, labels.shape[0]):
        if labels[i] == 1:
            ones.append(i)
        else:
            zeros.append(i)

    for i in ones:
        for j in zeros:
            for k in range(0, labels.shape[0]):

                if (k != i) & (k != j):

                    a = d[k, i]
                    c = d[j, i]
                    a2 = d2[k, i]
                    b2 = d2[k, j]
                    c2 = d2[j, i]
                    if (a * c) != 0:
                        tmp = ((b2-a2-c2)/(-2*a*c))
                    dist = a2*(1-tmp*tmp)+(a*tmp-c/2)**2

                    if dist < (c/2)**2:
                        break

                if k == data.shape[0] - 1:
                    xi = np.array(data[i])
                    xj = np.array(data[j])
                    cbp = np.true_divide(xi + xj, 2)
                    op = xi - xj
                    op_mod = np.sqrt((op * op).sum())
                    nvect = np.true_divide(op, op_mod)
                    cbps.append(cbp)
                    nvects.append(nvect)

    return np.array(cbps), np.array(nvects)
```

Codi font de l'algorisme d'obtenció dels CBP

Punt	X_i	X_j	Classe
A	1.49	4.79	1
B	2.82	3.16	1
C	2.58	5.36	1
D	1.79	6.53	1
E	4.28	4.12	1
F	3.68	3.19	1
G	1.61	3.23	1
H	1.87	3.79	1
I	2.09	7.17	0
J	2.84	6.1	0
K	3.51	6.89	0
L	2.82	7.69	0
M	4.78	4.64	0
N	3.56	7.59	0
O	5.22	5.89	0

Taula 4: Conjunt de dades bidemensional per a demostració de cerca dels CBPs

sobre aquesta traducció de Matlab® a Python® no hi ha molt que comentar, així que senzillament oferiré unes dades d'exemple, com les que es veuen a la taula 4 i mostraré el resultats que l'algorisme ens retorna, que pel cas que he presentat anteriorment és de 4 CBPs.

$$[array([2.71, 5.73]), array([1.94, 6.85]), array([2.315, 6.315]), array([4.53, 4.38])]$$

i he utilitzat el programari *Geogebra 5*[††] per a fer la representació gràfica d'aquest cas en particular. Com es pot comprovar a la figura 13 tots els punts compleixen les característiques per a ser CBPs, i per tant, podem donar l'algorisme com a vàlid.
Com es veu a la taula 5 els valors que s'han obtingut amb el *Geogebra 5*, que corresponen als punts P, Q, R i S, són els mateixos que ens retorna l'algorisme traduït.

	GeoGebra 5		compute_cbps	
	X_i	X_j	X_i	X_j
cbp_0 (R)	2.71	5.73	2.71	5.73
cbp_1 (Q)	1.94	6.85	1.94	6.85
cbp_2 (P)	2.31	6.31	2.315	6.315
cbp_3 (S)	4.53	4.38	4.53	4.38

Taula 5: Comparació dels valors obtinguts

[††]Programari de codi obert disponible a http://www.geogebra.org/

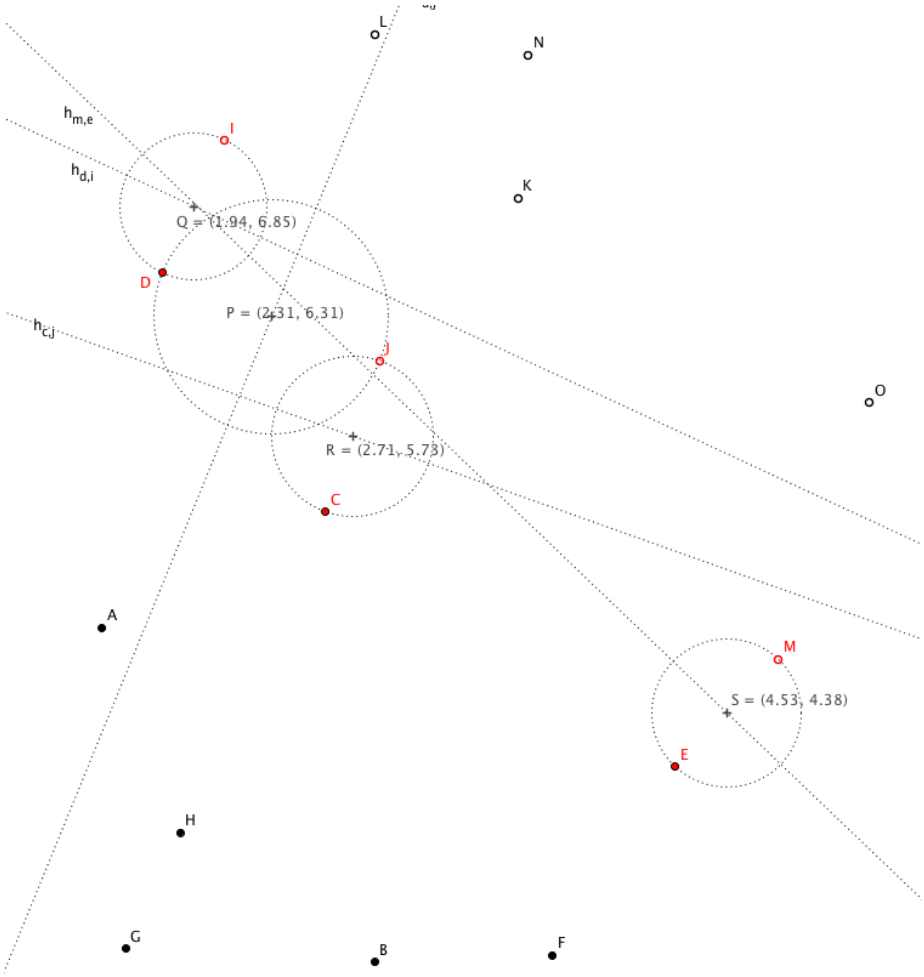

Figura 13: Representació gràfica dels 4 CBPs retornats.

3 Anàlisi de resultats

Al llarg d'aquesta secció explicaré els mètodes i els conjunts de dades que he usat per a mesurar els resultats obtinguts a la secció anterior, i, a més, en faré una valoració.

3.1 Elecció dels mètodes per a la validació

Per a validar l'algorisme he triat el mètode de validació creuada*. Aquest mètode consisteix a dividir el conjunt de dades d'entrada en K particions, després usem $K-1$ d'aquestes particions per a entrenar l'algorisme, i la que queda per a avaluar-ne l'eficiència. Aquest procés es repeteix K vegades, de forma que la eficiència s'avalua per a totes les particions, usant cada cop la resta per a entrenar el model. Al final de les K iteracions es pot obtenir el grau d'eficiència final senzillament fent la mitjana aritmètica entre els K resultats obtinguts, un per a cada iteració i avaluació. A més de calcular l'eficiència del model, calcularé també la desviació típica, que ens donarà el grau de dispersió respecte al valor mig calculat anteriorment, amb la qual cosa obtindrem una idea de l'error de càlcul en el grau d'eficiència.

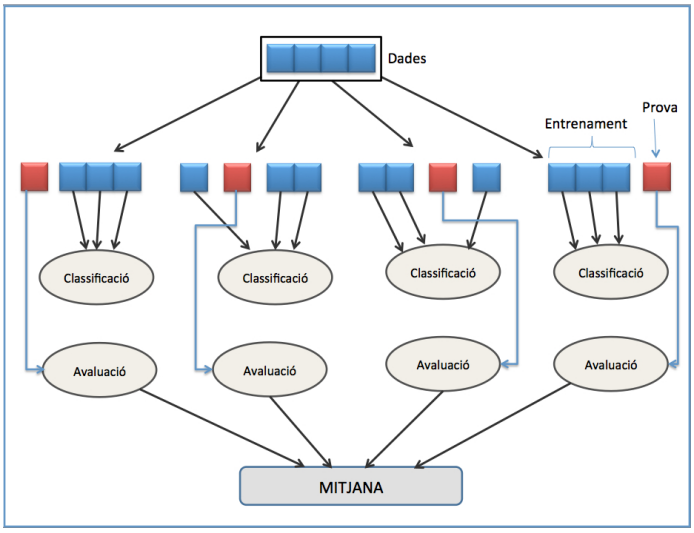

Figura 14: Esquema k-fold cross validation, amb $K = 4$ i amb un sol classificador.

A la figura 15 es pot veure gràficament una representació de l'algorisme de validació creuada. Per a la implementació d'aquest mètode de validació he usat la classe *KFold* de la llibreria *SciKit learn*†. La llibreria *SciKit learn* implementa

*K-Fold cross validation, en anglès.

†Per a consultar la documentació completa de la llibreria es pot fer a http://scikit-learn.org/stable/documentation.html

una sèrie de mètodes orientats a facilitar el desenvolupament d'algorismes relacionats amb l'aprenentatge computacional, i de fet, també implementen molts dels algorismes més coneguts i usats. Aquesta llibreria, i la possibilitat que ofereix Python® d'indexar matrius mitjançant llistes m'ha facilitat molt la tasca, atès que per a particionar les dades d'entrada es pot fer senzillament amb el codi següent.

```
kf = KFold(data.shape[0],
           n_folds=kfolds,
           shuffle=False,
           random_state=None)

for train_indexes, test_indexes in kf:

    features,
    cbps,
    nvects,
    ori_cbps = fit(data[train_indexes],
         labels[train_indexes],
         hash_max_size)

    prediction = predict(features,
          cbps,
          nvects,
          labels[train_indexes],
          data[test_indexes])

    accuracies[i] = compute_accuracy(prediction,
                    labels[test_indexes])
```

La classe KFold ens inicialitza dues llistes de llistes, cadascuna d'elles amb els índexos per a cada conjunt en cada partició, i com que a més és iterable, i Python® ens permet indexar matrius amb llistes, podem fer crides a les funcions d'entrenament i de prova passant com a paràmetres les dades i les etiquetes, però filtrades per les llistes que corresponguin en cada ocasió, facilitant molt així la tasca de controlar els índexs que s'han de tractar a cada iteració. Aquesta mateixa llibreria l'he usada per a calcular els valors calculats amb el mètode del veí més proper.

3.2 Elecció de les dades per a la validació

Per a provar l'algorisme he escollit els mateixos conjunts de dades que s'usaren a l'article original, de forma que podré comparar els resultats per mesurar la pèrdua d'eficiència amb la reducció del nombre de CBP's usats per a la classificació. De totes formes, com que l'article original usava la regulació de Tikhonov per tal d'aconseguir un resultat més acurat, i aquesta part no s'implementava a aquest projecte, la comparació no podrà ser comparable al cent per cent.

Els conjunts de dades escollits són de domini públic, i tots es poden aconseguir

Codi	Nom del conjunt de dades	Inst.	Atr.
bcw	Breast Cancer Wisconsin (Original) Data Set	699	10
bld	Liver Disorders Data Set	345	7
ion	Ionosphere Data Set	351	34
bre	Breast Cancer Wisconsin (Diagnostic) Data Set	569	32
snr	Connectionist Bench (Sonar, Mines vs. Rocks) Data Set	208	60
vot	Congressional Voting Records Data Set	435	16
cre	Credit Approval Data Set	690	15

Taula 6: Conjunt de dades escollit, nombre d'instàncies i atributs del conjunt

Codi	Nom del conjunt de dades	Efi.	D.E.
bcw	Breast Cancer Wisconsin (Original) Data Set	97,09	0,41
bld	Liver Disorders Data Set	69,74	1,31
ion	Ionosphere Data Set	90,92	0,90
bre	Breast Cancer Wisconsin (Diagnostic) Data Set	97,69	0,40
snr	Connectionist Bench (Sonar, Mines vs. Rocks) Data Set	78,68	1,63
vot	Congressional Voting Records Data Set	95,25	0,63
cre	Credit Approval Data Set	86,97	0,79

Taula 7: Conjunt de dades escollit, amb l'eficiència que s'obtenia amb l'algorisme original.

a la web del repositori de l'UCI[‡]. A la taula 6 es mostren els conjunts de dades escollits, amb el nombre d'atributs i d'instàncies de cadascun d'ells.

A la taula 7, de la pàgina 35, es mostren els mateixos conjunts de dades amb la eficiència (Efi.) i la desviació estàndar (D.E.) obtingudes a l'article original. Les dades de la taula 7 són les que usaré per a comparar amb les dades obtingudes amb l'algorisme de hashing.

[‡]Repository del departament d'aprenentage computacional de la universitat de Irvine (https://archive.ics.uci.edu/ml/datasets.html)

Preparació de les dades per a la validació

Tot i que les dades són fàcils de descarregar des de la web de la UCI, aquestes no sempre estan disponibles per a usar directament, és a dir, no estàn normalitzades, per tant, serà necessari un tractament previ per tal de normalitzar-les. A més, com a requeriment de l'algorisme he fet que les dades es llegeixin d'un fitxer on els atributs es representen a les columnes, i les instàncies a les files, i on, a més, la darrera columna es correspon a la classe coneguda de cadascuna de les instàncies. Amb això, les dades que es poden descarregar de la web han de ser tractades.

A l'assignatura d'*Intel·ligència artificial II*, al mòdul d'aprenentatge, se'ns va explicar que, segons Sneath i Sokal [4] *suggereixen que quan els atributs binaris són majoria i es fan servir amb coeficients de correlació, s'han de codificar mitjançant els valors 0 i 1 i no s'han d'estandarditzar; si en aquest cas hi ha atributs ordinals es recomana codi- ficar-los amb codificació additiva. Per a situacions en què hi ha atributs binaris i ordinals o nominals es proposa el ranging. Quan els atributs són majoritàriament numèrics o categòrics, es recomana, en general, l'estandarditzaci*ó. Per tant, això és el que s'ha fet. Com a mètode de normalització he escollit, en els casos en que així ha estat necessari, l'estandarització. A l'apèndix iii es presenta una relació de les accions que s'han dut a terme per a cada conjunt de dades.

3.3 Anàlisi dels resultats obtinguts

Per tal de validar els resultats obtinguts no he pogut fer ús dels resultats publicats a l'article original, atès que, tot i usar els mateixos conjunts de dades, no conec el tractament previ que es va aplicar al procés original, i aquest pot ser diferent al que jo he aplicat. Per tant, el que he fet ha estat usar els meus orígens de dades amb algorismes de classificació ben coneguts[§], i comparar els resultats obtinguts amb tots ells amb els obtinguts amb l'algorisme de hashing basat en hiperplans localment òptims (HCBP). A més, per tal d'avaluar la pèrdua d'eficiència respecte a l'algorisme original, també he usat les mateixes dades amb l'algorisme original, i els he comparat amb els obtinguts reduïnt el nombre de CBPs a 4 (HCBP4) i 8 (HCBP8) bits, respectivament[¶].
A l'apèndix IV s'inclouen tots els resultats obtinguts per a cada joc de proves, i d'ells en podem treure les següents conclusions.

- Passat un cert nombre de bits de hash, diferent per a cada joc de proves, augmentar la mida del hash no implica una millora en el grau de classificació. Fins i tot, pot suposar-ne un empitjorament.

[§]S'han usat diverses variacions de Knn (ball tree, kd tree i brute force), dos implementacions diferents de les màquines de vectors de suport (rbf i lin), l'AdaBoost i el Random Forest. Tots ells amb les implementacions disponibles al paquet SciKit-Learn per a Python©

[¶]A partir d'ara usaré HCBP4 i HCBP8 per a referir-me a l'algorisme de hashing que redueix a 4 i 8 punts respectivament.

- En cap dels casos provats per a mides de hash iguals o majors a 8 bits, mai s'ha assolit el límit de la mida del hash.
- La pèrdua d'eficiència envers l'algorisme original és molt petita.
- Els graus d'eficiència en la classificació de l'HCBP4 i l'HCBP8 són perfectament comparables als dels altres algorismes, i fins i tot en alguns casos és millor.

Analitzem les conclusions una per una. El fet que passat un cert nombre de bits no s'augmenti el grau de classificació és lògic, de fet és un dels avantatges de la reducció de la dimensionalitat, és a dir, quedar-se només amb la informació rellevant per a classificar, i eliminar la redundant o fins i tot, la que et causa perjudici. Si pensem en el que he explicat a 1.3 sobre els CBPs redundants i els Outliers, això és precisament el que hem aconseguit. La segona conclusió s'explica també pel mateix motiu que la primera, però m'ha sorprés que, donat l'elevat nombre de CBPs que es calculen per alguns dels conjunts de dades, la reducció sigui tan gran. La tercera conclusió és un dels objectius inicials del projecte, és a dir, la de reduïr el nombre de CBPs sense afectar notablement l'eficiència en la classificació de l'algorisme. Si es comparen els resultats obtinguts, a excepció de dos casos, el grau de classificació tant de HCBP4 i HCBP8 és molt similar al de l'OGE, i, a més, he de dir que les proves fetes amb l'OGE no estàn fetes amb k-Fold Cross Validation, atès que no es tenia la implementació en Matlab© del k-Fold Cross Validation, i per tant, s'han usat les dades d'entrenament com a dades de test. En canvi, pels dos casos en els que la diferència és gran, el grau de classificació dels altres algorismes també cau molt, i en molts de casos cau més del que ho fa amb l'HCBP. Això em serveix per a explicar la tercera conclusió, i és que els resultats obtinguts amb l'HCBP4 i l'HCBP8 estan a l'alçada dels obtinguts amb els diferents algorismes provats. Aquesta conclusió es veu millor a la figura 15.A les taules 8 i 9 es pot veure l'eficiència obtinguda en cada cas, així com la desviació estàndar, representada per σ, que ens dóna una idea de la qualitat d'aquesta eficiència.

	VOT		BRE		ION		BLD	
	Eficiència	σ	Eficiència	σ	Eficiència	σ	Eficiència	σ
knn (ball_tree)	0,9334	0,0426	0,9560	0,0227	0,8294	0,1106	0,5366	0,1244
knn (kd_tree)	0,9311	0,0397	0,9560	0,0227	0,8294	0,1106	0,5366	0,1244
knn (brute)	0,9241	0,0474	0,9560	0,0227	0,8294	0,1106	0,5366	0,1244
SVM (rbf)	0,9562	0,0337	0,9666	0,0242	0,9260	0,0602	0,6320	0,0784
SVM (lin)	0,9539	0,0345	0,9648	0,0222	0,8692	0,0739	0,6552	0,1302
Adaboost	0,9585	0,0289	0,9736	0,0162	0,9233	0,0568	0,6902	0,0899
Random Forest	0,9539	0,0330	0,9491	0,0346	0,9258	0,0575	0,6549	0,0836
OGE	0,9218	0,0000	0,9877	0,0000	0,9231	0,0000	0,8029	0,0000
HCBP (4 bits)	0,8660	0,0419	0,9613	0,0293	0,8660	0,0849	0,4911	0,1347
HCBP (8 bits)	0,9423	0,0414	0,9595	0,0295	0,8860	0,0829	0,4992	0,1118

Taula 8: Taules comparatives dels resultats obtinguts pels conjunts VOT, BRE, ION i BLD.

	BCW		SNR		CRE	
	Eficiència	σ	Eficiència	σ	Eficiència	σ
knn (ball_tree)	0,9634	0,0358	0,4036	0,1659	0,8098	0,1394
knn (kd_tree)	0,9634	0,0358	0,4036	0,1659	0,8098	0,1394
knn (brute)	0,9634	0,0358	0,4036	0,1659	0,8098	0,1394
SVM (rbf)	0,9679	0,0304	0,2188	0,2272	0,8393	0,1818
SVM (lin)	0,9663	0,0285	0,5345	0,1714	0,8611	0,1961
Adaboost	0,9605	0,0351	0,7081	0,2027	0,8410	0,1555
Random Forest	0,9693	0,0303	0,5731	0,1396	0,8209	0,1477
OGE	0,9751	0,0000	0,9038	0,0000	0,8721	0,0000
HCBP (4 bits)	0,9664	0,0332	0,5552	0,2901	0,8329	0,1756
HCBP (8 bits)	0,9664	0,0332	0,5681	0,2482	0,8298	0,1827

Taula 9: Taules comparatives dels resultats obtinguts pels conjunts BCW, SNR i CRE.

La figura 15 és una representació en format circular de les dades de les taules 8 i 9, aqui es pot apreciar com el comportament de l'HCBP4 i l'HCBP8 és molt similar al dels altres algorismes.

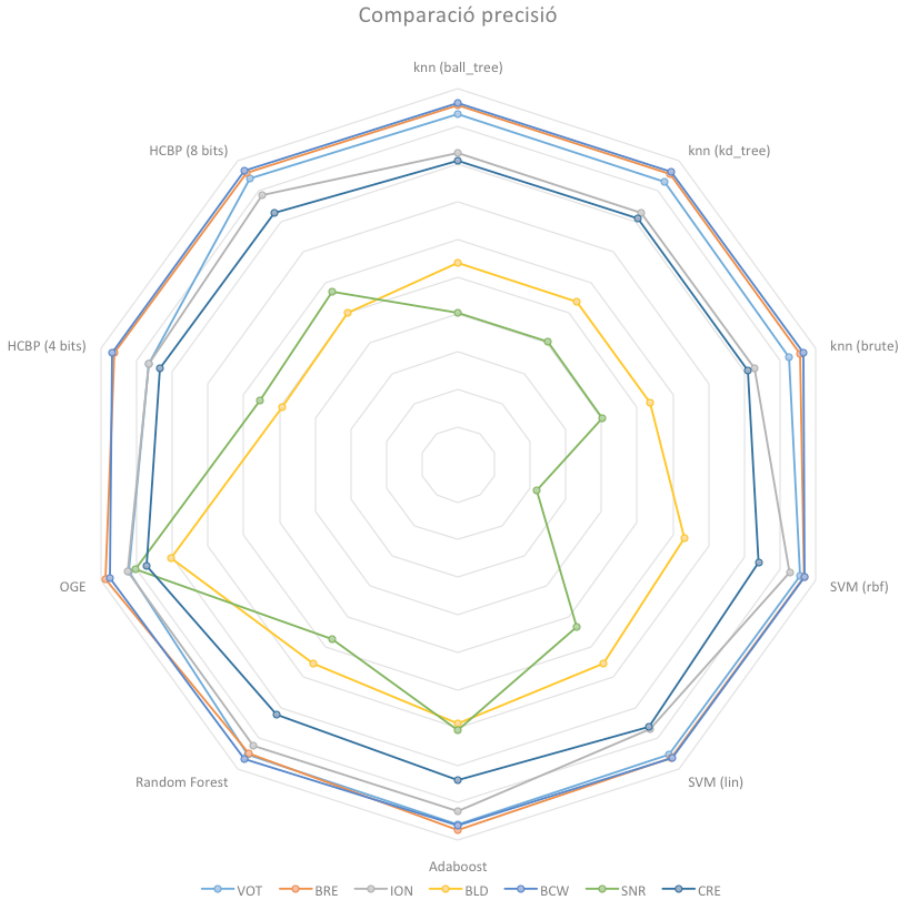

Figura 15: Comparació dels resultats obtinguts amb tots els algorismes.

Al gràfic de la figura 16 es pot veure com la diferència entre l'OGE i l'HCBP és mínima excepte pels dos casos esmentats anteriorment (jocs de proves SNR i BLD). Com ja he explicat, crec que aquests dos jocs de proves s'han tractat de forma incorrecta prèviament, atès que els altres algorismes pateixen una reducció del seu grau de precisió usant les mateixes dades. El fet de que amb l'OGE això no passi crec que es degut al fet de no usar les dades d'entrenament per a test, i que amb l'OGE no s'ha usat el k-Fold Cross Validation.

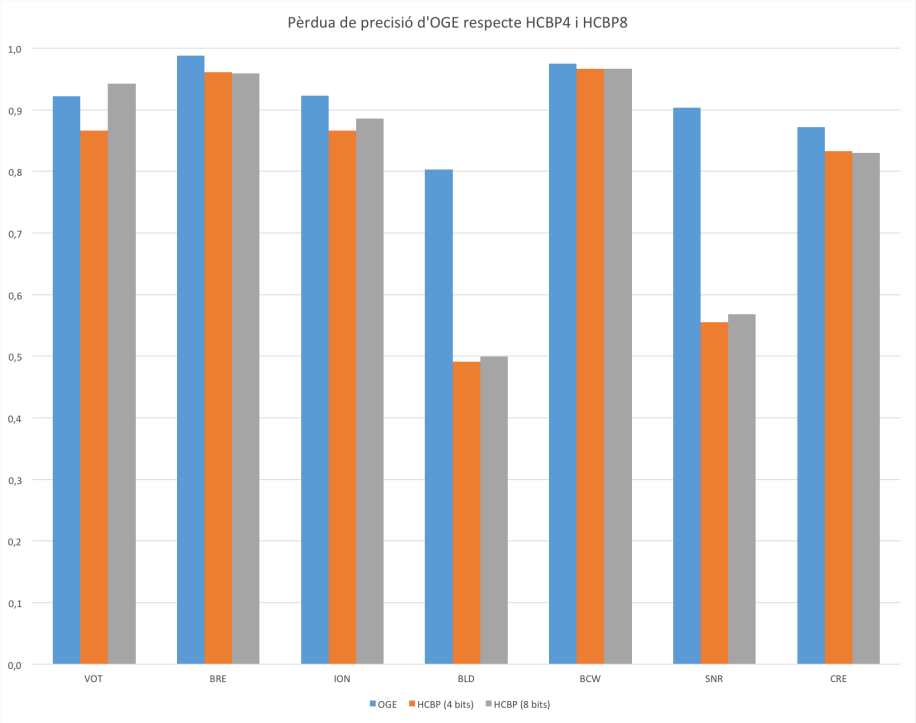

Figura 16: Comparació de OGE envers HCBP4 i HCBP8.

3.4 Problemes sorgits al llarg del desenvolupament del projecte

Al llarg d'aquest apartat explicaré els problemes més importants que anat trobant i solucionant.

Retrasos en la planificació

Tot i aue no han suposat un problema real en el desenvolupament general del projecte, els problemes de comprensió del problema inicial em van suposar un retràs en la fase inicial del projecte que he anat arrossegant fins al final. La comprensió de l'article inicial va ser complexe, amb una alta càrrega de continguts matemàtics que volia entendre, com per exemple, la regularització de Tikhonov, i que, en realitat, no era necessari entendre atès que el projecte no havia d'implementar res relacionat amb aquesta part de l'article original. Entendre per què i com havia de reduïr el nombre de CBPs va dur més temps del desitjat.

Problemes en la implementació

La implementació de l'algorisme, sobre els papers, era senzilla, però a mesura que avançava el projecte he anat veient com sorgien problemes d'eficiència, degut al temps de computació requerit per a realitzar les fases d'entrenament i prova. S'han hagut de modificar moltes de les parts de l'algorisme inicial per tal de baixar el temps de computació. Especialment interessant ha estat el veure com els bucles *for* penalitzen computacionalment, i s'han hagut de substituir, sempre que ha estat possible, per altres solucions, com càlculs matricials, per tal de millorar el temps d'execució total. Tot i així, crec que encara es pot millorar, i, com que és un tema que m'interessa, seguiré treballant-hi per tal de millorar-lo.

Problemes en la documentació

Tot i que estic content d'haver pres la decisió inicial de redactar la memòria en LaTeX, el no estar habituat a usar-lo com a editor de texts, en algunes fases del projectes he dedicat més temps del desitjat en la recerca de com realitzar algunes tasques, com per exemple, insertar i aliniar imatges, o com incloure taules.

4 Conclusions

Les conclusions de l'estudi realitzat s'han explicat a la secció 3.3, i per tant, aquesta secció tractarà de les conclusions que es poden obtindre del desenvolupament de les diferents fases d'aquest projecte.

Al pla docent de l'assignatura se'ns donen dues idees clares del que ha de ser el projecte final de carrera. Se'ns diu que *el projecte fi de carrera és una assignatura que està pensada per a realitzar un treball de síntesi dels coneixements adquirits en altres assignatures de la carrera i que requereixi posar-los en pràctica conjuntament en un treball concret.* Aquesta part crec que queda perfectament coberta amb el treball realitzat. Una gran part dels coneixements requerits pel seu desenvolupament han estat adquirits prèviament tant al primer cicle com al segon cicle de l'enginyeria informàtica, com per exemple, els algorismes de classificació basats en màquines de vectors de suport, els del veï més proper o l'AdaBoost, així com conceptes estadístics usats per a obtenir-ne les conclusions. Dels coneixements necessaris adquirits al primer cicle podria destacar tota la part de programació.

També se'ns explica que *normalment el PFC és un treball eminentment pràctic i vinculat a l'exercici professional de la informàtica, tot i que, i sense que hagi de ser un treball de recerca, sí que hauria d'incloure certa dosi de coneixement addicional.* Per una banda, tot i que atesa la naturalesa del projecte aquest no era un treball eminenment pràctic, sí que he hagut de realitzar treballs pràctics, i en són una mostra la traducció del codi en Matlab a Python, o la implementació de l'HCBP en Python. Per altra banda, he hagut d'aprendre un nou algorisme de classificació que no havíem vist al llarg dels anys acadèmics, l'OGE, i fins i tot, n'he dissenyat un altre. l'HCBP. A més, he aprofundit en el coneixement del llenguatge Python, i he aprés a usar Latex per a la redacció d'aquesta documentació.

Diverses són les formes en que aquest projecte es podria continuar. Es podríen desenvolupar els paquets necessaris per tal d'incloure tant l'OGE com l'HCBP al repositori de l'SciKit Learning, per tal de que aquests estiguin a l'abast de tothom. També es podria intentar fer un tractament inicial als CBPs, per tal d'eliminar la majoria de CBPs redundants o de Outliers prèviament a la reducció del nombre de CBPs per tal d'agilitzar el procés. Alguns d'aqusts objectius espero poder assolir-los en el futur, però quedaven fòra de l'abast dels objectius inicials, i per això no s'han duit a terme.

Per tant, crec que es pot afirmar que s'han assolit els propòsits que persegueix el projecte final de carrera.

Apèndixs

i Apèndix I: Codi font de l'article original

i.1 Codi font per al càlcul dels CBP

```
% This is a beta version of the Optimized Geometry-based
% Ensemble basic classifier.
% Copyright 2008, 2009 Oriol Pujol and David Masip.
% This software is distributed under the terms of the
% GNU General Public License

function [NodeA, NodeB, log, D, D2] = edgesGN(X,labels)

[D N] = size(X);
labelval=unique(labels);
label1 = find(labels == labelval(1));
label2 = find(labels == labelval(2));
N1 = size(label1,2);
N2 = size(label2,2);

%precomputing the distances
D = L2_distance(X,X);
D2 = D.*D;

NodeA = [];
NodeB = [];

for i = 1:N1
    for j= 1:N2
        %Check the GN condition
        for k = 1:N
            if and((k ~= i),(k~=j))
                A = D(k,label1(i));
                B = D(k,label2(j));
                C = D(label2(j),label1(i));
                A2 = D2(k,label1(i));
                B2 = D2(k,label2(j));
                C2 = D2(label2(j),label1(i));
                tmp = ((B2-A2-C2)/(-2*A*C));
                dist = A2*(1-tmp*tmp)+(A*tmp-C/2).^2;
                if dist < (C/2).^2
                   break;
                end
            end
        end
        log(i,j) = k;
        if k == N
            NodeA = [NodeA i];
            NodeB = [NodeB j];
        end
    end
end
NodeA=label1(NodeA);
NodeB=label2(NodeB);
```

Codi font de l'algorisme d'obtenció dels CBP

ii Apèndix II: Codi font del projecte

ii.1 Codi font del projecte

```
from tabulate import  tabulate
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import KFold
from datetime import datetime
import numpy as np
import scipy.spatial
import os


def load_data(csv):
    """
    Load data from a comma separated values file. Attributes
  must be represented in columns and samples in rows. Last
  column is the class that represent to each sample.
    :param csv: path to the csv file
    :return: a numpy array representing the data and a numpy
  array representing the classes
    """
    if os.path.isfile(csv):
        data = np.genfromtxt(csv, delimiter=';')
        size = data.shape
        n_samples = size[0]
        n_attributes = size[1] - 1
        labels = np.zeros(n_samples)

        for i in range(0, n_samples):
            labels[i] = data[i, n_attributes]

        data = np.delete(data, n_attributes, 1)
        return data, labels
    else:
        return None


def compute_accuracy(prediction, labels):
    """
    Returns the accuracy of a particular classification against
    the vector of known values of its class.

    :arg prediction: Boolean vector. Each bit corresponding to
    classification values of a set of points
    :arg labels: Boolean vector. Each bit corresponding to real
    class of that point

    :return: Accuracy of 'classification'. A real value between
    0 and 1 where 0 is the worst classification accuracy and 1
    is the best.
    """
    return 1 - np.true_divide(
        np.logical_xor(prediction, labels).sum(),
        labels.shape[0])
```

```
def hash_features(features,
                  cbps,
                  nvects,
                  labels,
                  hash_max_size):
    """
    This function returns a reduced set of cbps, a set of features
    and its normals vectors, based on the originals sets computed
    by compute_cbps function. hash_max_size only works as a limiter
    of the hash size. If the algorithm find a hashed set lower in
    size but with a better classification index, it returns the
    small one. The accuracy has priority on hash size.
    :param features: array representing features for each sample
    computed with compute_oge function
    :param cbps: array of cbps computed by compute_cbps function
    :param nvects: array of normal vectors, one for each feature
    :param labels: classes of each sample
    :param hash_max_size: maximum size of the hash
    :return: returns a reduced set of cbps, a set of features and
    its normals vectors, based on the originals sets computed by
    compute_cbps function
    """

    #Initial setup of local variables
    max_iterations = 10
    iterations = 0
    features = features.transpose()
    n_cbps = features.shape[0]
    last_iteration_accuracy = -1
    item_deleted = True

    # compute accuraccies of each cbp to get the maximum value
    accuracies = [compute_accuracy(features[i, :],
                  labels) for i in range(0, n_cbps)]
    i = accuracies.index(max(accuracies))

    # initialize hash with the cbp that has the maximum
  #individual accuracy
    hashed_cbps, hashed_accuracy = [i], accuracies[i]

    while True:

        """
        Check exit conditions
        """
        # maximum number of iterations reached
        if iterations == max_iterations:
            break
        # accuracy of 1.0 reached
        if hashed_accuracy == 1.0:
            break
        # check if accuracy has improved
        if hashed_accuracy > last_iteration_accuracy:
            last_iteration_accuracy = hashed_accuracy
        else:
            break
```

```
# check if some item has been deleted in last iteration
if not item_deleted:
    #print "Element not deleted. Break"
    break

"""
Stage 1. Add only cbp's that improves the prediction accuracy
"""
while True:
    max_accuracy = -1.
    max_i = -1

    for i in range(n_cbps):
        if i not in hashed_cbps:
            tmp_hashed_cbps = hashed_cbps + [i]
            prediction = compute_knn_classification(
         features[tmp_hashed_cbps].transpose(),
         labels)
            tmp_accuracy = compute_accuracy(prediction, labels)

            if tmp_accuracy > max_accuracy:
                max_accuracy = tmp_accuracy
                max_i = i

    if max_accuracy > hashed_accuracy:
        hashed_cbps = hashed_cbps + [max_i]
        hashed_accuracy = max_accuracy

    # exit condition 1 for stage 1
    else:
        break
    # exit condition 2 for stage 1
    if len(hashed_cbps) >= hash_max_size:
        break

"""
Stage 2. Remove cbps only if deletion improves
the prediction accuracy
"""
while True:

    # exit condition 1 for stage 2
    if len(hashed_cbps) < 2:
        break

    max_accuracy = -1.
    max_i = -1
    item_deleted = False

    for i in hashed_cbps:
        tmp_hashed_cbps = hashed_cbps[:]
        tmp_hashed_cbps.remove(i)

        prediction = compute_knn_classification(
                     features[tmp_hashed_cbps].transpose(),
       labels)
        tmp_accuracy = compute_accuracy(prediction, labels)
```

```
                if tmp_accuracy > max_accuracy:
                        max_accuracy = tmp_accuracy
                        max_i = i

            if max_accuracy > hashed_accuracy:
                item_deleted = True
                hashed_cbps.remove(max_i)
                hashed_accuracy = max_accuracy

             # exit condition 2 for stage 2
            else:
                break

        iterations += 1

    return features[hashed_cbps].transpose(),
           cbps[hashed_cbps],
           nvects[hashed_cbps]


def compute_knn_classification(data, labels):
    """
    Compute classification based on k neighbor
  implemented in the scikit library.
    :param data: data to which apply knn algorithm
    :param labels: labels of data samples
    :return: boolean array corresponding to the
  classification
    """

    results = np.zeros(data.shape[0])

    for i in range(data.shape[0]):
        # removes the sample before fit the trainig set
        # we don't have to take care of the sample itself
        temp_data = np.delete(data, i, 0)
        temp_labels = np.delete(labels, i, 0)
        knn = KNeighborsClassifier()
        knn.fit(temp_data, temp_labels)
        results[i] = knn.predict(data[i])

    return results


def compute_oge_classification(point, cbp, nvect):
    """
    Returns a boolean indicating if point 'p' is at either
  one side or another of the hyperplane defined by 'cbp',
  depending on the normal vector that connects them.

    :arg point: point to classify based on the hyperplane
  defined by cbp
    :arg cbp: cbp defining the hyperplane
    :arg nvect: normal vector of hyperplane defined by cbp

    :return: boolean indicating if 'point' is at either one side
```

```
    or another of the hyperplane defined by 'cbp', depending on
    the normal vector that connects them. 1 for possitive value
    of classifier, 0 otherwise

    """
    return 0 if (np.dot((point - cbp), nvect) < 0) else 1


def compute_features(data, cbps, nvects):
    """
    Compute features of each sample based on oge classification
    :param data: data to which apply algorithm
    :param cbps: cbp set
    :param nvects: normal vectors of that cbps
    :return: array containing features
    """

    n_features = len(cbps)
    n_samples = data.shape[0]

    features = np.zeros((n_samples, n_features))

    for i in range(0, n_features):
        for j in range(0, n_samples):
            features[j][i] = compute_oge_classification(np.array(data[j]),
                             cbps[i],
                             nvects[i])

    return features


def compute_cbps(data, labels):
    """
    Returns a set of characterizing boundary points computed
    using Gabriel neighborhood rule. This is an adaptation to
    python language of the beta version of the Optimized
    Geometry-based Ensemble basic classifier, copyright 2008,
    2009 by Oriol Pujol and David Masip.

    :arg ds: Dataset containing the data
    :return: List of pair of characterizing boundary points
    and its normal vector
    """
    idxs, cbps, nvects, ones, zeros = [], [], [], [], []
    d = compute_distance_matrix(data, data)
    d2 = d ** 2

    for i in range(0, labels.shape[0]):
        if labels[i] == 1:
            ones.append(i)
        else:
            zeros.append(i)

    for i in ones:
        for j in zeros:
            for k in range(0, labels.shape[0]):
```

```
                if (k != i) & (k != j):

                    a = d[k, i]
                    c = d[j, i]
                    a2 = d2[k, i]
                    b2 = d2[k, j]
                    c2 = d2[j, i]
                    if (a * c) != 0:
                        tmp = ((b2-a2-c2)/(-2*a*c))
                    dist = a2*(1-tmp*tmp)+(a*tmp-c/2)**2

                    if dist < (c/2)**2:
                        break

                if k == data.shape[0] - 1:
                    xi = np.array(data[i])
                    xj = np.array(data[j])
                    cbp = np.true_divide(xi + xj, 2)
                    op = xi - xj
                    op_mod = np.sqrt((op * op).sum())
                    nvect = np.true_divide(op, op_mod)
                    cbps.append(cbp)
                    nvects.append(nvect)

    return np.array(cbps), np.array(nvects)


def compute_distance_matrix(u, v):
    """
    Compute euclidean distances between to unidimensional vectors
    :param u: first unidimensional vector (n elements)
    :param v: second unidimensional vector (m elements)
    :return: array (n * m) containing distances between
    elements of u and v
    """
    distances = np.zeros((len(u), len(v)))
    for x in range(len(u)):
        for y in range(len(v)):
            if (x != y) & (y < len(v)):
                distances[x][y] = scipy.spatial.distance.euclidean(v[x],
                                                                   u[y])

    return distances


def fit(data, labels, hash_max_size):
    """
    This functions computes the training of the classification model.
  It returns a reduced set of cbps, array of features and normal
  vectors of the cbps. These three elements are necessary to perform
  a classification
    :param data: input data
    :param labels: input labels
    :param hash_max_size: maximum hash size
    :return:
    """
    cbps, nvects = compute_cbps(data, labels)
```

```
    features = compute_features(data, cbps, nvects)

    features, hcbps, nvects = hash_features(features,
                                            cbps,
                                            nvects,
                                            labels,
                                            hash_max_size)

    return features, hcbps, nvects, cbps


def predict(model_features,
      model_cbps,
      model_nvects,
      model_labels,
      data):
    """
    This function gets the elements of the model and performs a
  prediction of classes on 'data' dataset
    :param model_features: array of features computed in 'fit' function
    :param model_cbps: array of cbps computed in 'fit' function
    :param model_nvects: array of normal vectors computed in 'fit'
  function
    :param model_labels: known labels of the model
    :param data: data to which perform the prediction
    :return: boolean array containing prediction
    """
    features = compute_features(data, model_cbps, model_nvects)
    prediction = np.zeros(data.shape[0])

    # perform training on known data and labels
    knn = KNeighborsClassifier()
    knn.fit(model_features, model_labels)

    # perform knn prediction on data samples
    for i in range(data.shape[0]):
        prediction[i] = knn.predict(features[i, :])

    return prediction


def hcbp(csv_dataset, hash_max_size, kfolds=10, verbose=False):
    """
    This function is used to perform the k-fold cross validation
  of the model. By default it performs 10-fols cross validation
    :param csv_dataset: csv dataset file path
    :param hash_max_size: maximum hash size
    :param kfolds: optional parameter indicating the number
  of folds to use
    :param verbose: boolean indicating where to show on
  screen output
    :return: a tuple containing accuracy of the model, averages
  times for training and test operations and average number of
  bits used in hash
    """
```

```
# init all local variables
i = 0
data, labels = load_data(csv_dataset)

# Compute k-cross fold validation to test results
# by default kfolds = 10
kf = KFold(data.shape[0],
           n_folds=kfolds,
           shuffle=False,
           random_state=None)

accuracies,
bits,
hbits,
train_times,
test_times = np.zeros(kfolds),\
             np.zeros(kfolds),\
             np.zeros(kfolds),\
             np.zeros(kfolds),\
             np.zeros(kfolds)

# file to print the results
head, tail = os.path.split(csv_dataset)

output = tail.upper() + '\n\n'
output += '\tfilename : ' + csv_dataset + '\n'
output += '\tsamples : ' + str(data.shape[0]) + '\n'
output += '\tattributes : ' + str(data.shape[1]) + '\n'
output += '\thash-max-size : ' + str(hash_max_size) + '\n'

cols = ['K-Fold-' + str(j) for j in range(0, kfolds)]
cols.insert(0, ' ')

for train_indexes, test_indexes in kf:
    # train model and measure running time
    t1 = datetime.now()
    features,
cbps,
nvects,
ori_cbps = fit(data[train_indexes],
         labels[train_indexes],
         hash_max_size)

    t2 = datetime.now()

    # test model and measure running time
    prediction = predict(features,
           cbps,
           nvects,
           labels[train_indexes],
           data[test_indexes])
    t3 = datetime.now()

    bits[i] = ori_cbps.shape[0]
    train_times[i] = np.true_divide((t2 - t1).microseconds, 1000)
    test_times[i] = np.true_divide((t3 - t2).microseconds, 1000)
    hbits[i] = cbps.shape[0]
```

```
        accuracies[i] = compute_accuracy(prediction,
                        labels[test_indexes])
        i += 1

    accuracy = np.true_divide(accuracies.sum(), kfolds)
    traintime_avg = np.true_divide(train_times.sum(), kfolds)
    testtime_avg = np.true_divide(test_times.sum(), kfolds)
    hbits_avg = np.true_divide(hbits.sum(), kfolds)
    bits_avg = np.true_divide(bits.sum(), kfolds)

    str_accuracies = [str(accuracies[i]) for i in range(0, kfolds)]
    str_accuracies.insert(0, 'Accuracy')

    str_cbps = [str(bits[i]) for i in range(0, kfolds)]
    str_cbps.insert(0, '# CBPs')

    str_hcbps = [str(hbits[i]) for i in range(0, kfolds)]
    str_hcbps.insert(0, 'Hashed # CBPs')

    str_train_times = [str(train_times[i]) for i in range(0, kfolds)]
    str_train_times.insert(0, 'Train time')

    str_test_times = [str(test_times[i]) for i in range(0, kfolds)]
    str_test_times.insert(0, 'Test time')

    output += '\taccuracy : ' + str(accuracy) + '\n\n'
    output += tabulate(np.array([str_accuracies,
                str_cbps,
                str_hcbps,
                str_train_times,
                str_test_times]),
                      cols,
                      tablefmt='grid') + '\n\n'

    # file to print the results
    fname = 'output.txt'
    with open(fname, "a") as f:
        f.write(output)
        f.close()

    if verbose:
        print output
    else:
        print '+'

    return accuracy, traintime_avg, testtime_avg, bits_avg, hbits_avg
```

Codi font de l'algorisme d'obtenció dels CBP

iii Apèndix III: Tractament dels conjunts de dades de proves

Breast Cancer Wisconsin (Original) Data Set (bcw)

Obtingut de https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29. Aquest conjunt de dades s'ha estandaritzat, i s'han eliminat les dades incompletes.

Liver Disorders Data Set (bld)

Obtingut de https://archive.ics.uci.edu/ml/datasets/Liver+Disorders. Aquest conjunt de dades s'ha estandaritzat, i s'han eliminat les dades incompletes.

Ionosphere Data Set (ion)

Obtingut de https://archive.ics.uci.edu/ml/datasets/Ionosphere. Aquest conjunt de dades s'ha estandaritzat, i s'han eliminat les dades incompletes.

Breast Cancer Wisconsin (Diagnostic) Data Set (bre)

Obtingut de https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29. Aquest conjunt de dades s'ha estandaritzat, i s'han eliminat les dades incompletes.

Connectionist Bench (Sonar, Mines vs. Rocks) Data Set (snr)

Obtingut de https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar%2C+Mines+vs.+Rocks%29. Aquest conjunt de dades s'ha estandaritzat, i s'han eliminat les dades incompletes.

Congressional Voting Records Data Set (vot)

Obtingut de https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records. Les dades d'aquest conjunt de dades són totes binàries, i, per tant, s'han deixat tal i com s'han obtingut.

Credit Approval Data Set (cre)

Obtingut de https://archive.ics.uci.edu/ml/datasets/Credit+Approval. Aquest conjunt de dades s'ha estandaritzat, i s'han eliminat les dades incompletes.

iv Apèndix IV: Resultats obtinguts per a cada joc de proves

Per a cada joc de proves mostraré una taula on hi figuren les diferents precisions obtingudes per a cada valor de hash de l'algorisme. A cada taula hi figuren les següents columnes.

1. La primera columna ens indica el nombre de bits solicitats a la funció de hash.
2. La segona columna ens mostra la precisió obtinguda per a aquest nombre de bits.
3. La tercera columna ens mostra la desviació estàndar del model.
4. La quarta columna ens mostra el temps dedicat a l'entrenament del model, en mil·lisegons. 5. La cinquena columna ens indica el temps que s'ha emprat en classificar les dades de test. 6. La sisena columna ens mostra en nombre inicial de CBPs calculats amb l'algorisme OGE. 7. La setena columna ens indica el nombre de bits que s'han usat, és a dir, el hashing real. Com que l'algorisme prioritza la precisió per sobre del nombre de bits de hashing sol·licitats, si assolim un grau major de classificació amb menys bits, ens retornarà menys bits. El motiu pel qual aquesta columna indica un nombre decimal de bits és per que és el resultat de la mitja de les K iteracions realitzades en el K-Fold.

	bcw.csv accuracy	bcw.csv std_deviation	bcw.csv train ms	bcw.csv test ms	bcw.csv cbps	bcw.csv hash cbps
4 bits	0.966411	0.0331601	30.4017	0.016	590	2.7
8 bits	0.966411	0.0331601	31.7545	0.016	590	2.7
12 bits	0.966411	0.0331601	28.0707	0.014	590	2.7
16 bits	0.966411	0.0331601	25.9061	0.011	590	2.7
20 bits	0.966411	0.0331601	25.8606	0.013	590	2.7
24 bits	0.966411	0.0331601	25.6984	0.013	590	2.7
28 bits	0.966411	0.0331601	25.9345	0.015	590	2.7
32 bits	0.966411	0.0331601	25.9297	0.012	590	2.7

Figura 17: Resultats del joc de proves bcw

	bld.csv accuracy	bld.csv std_deviation	bld.csv train ms	bld.csv test ms	bld.csv cbps	bld.csv hash cbps
4 bits	0.491092	0.134663	24.5296	0.01	1176.8	3.7
8 bits	0.49916	0.111829	1172.8	0.007	1176.8	5.9
12 bits	0.475546	0.107795	49.6318	0.009	1176.8	6.4
16 bits	0.475546	0.107795	51.0545	0.012	1176.8	6.4
20 bits	0.475546	0.107795	51.4516	0.008	1176.8	6.4
24 bits	0.475546	0.107795	51.2974	0.012	1176.8	6.4
28 bits	0.475546	0.107795	51.132	0.011	1176.8	6.4
32 bits	0.475546	0.107795	51.3377	0.011	1176.8	6.4

Figura 18: Resultats del joc de proves bld

	ion.csv accuracy	ion.csv std_deviation	ion.csv train ms	ion.csv test ms	ion.csv cbps	ion.csv hash cbps
4 bits	0.866032	0.0848849	15.9785	0.0073	836.1	4
8 bits	0.885952	0.0828931	29.1585	0.0078	836.1	6.6
12 bits	0.885952	0.0828931	30.58	0.0075	836.1	6.8
16 bits	0.885952	0.0828931	30.9811	0.0086	836.1	6.8
20 bits	0.885952	0.0828931	30.9	0.0079	836.1	6.8
24 bits	0.885952	0.0828931	30.6346	0.0079	836.1	6.8
28 bits	0.885952	0.0828931	30.9278	0.0079	836.1	6.8
32 bits	0.885952	0.0828931	30.5848	0.0076	836.1	6.8

Figura 19: Resultats del joc de proves ion

	bre.csv accuracy	bre.csv std_deviation	bre.csv train ms	bre.csv test ms	bre.csv cbps	bre.csv hash cbps
4 bits	0.961278	0.0292839	143.162	0.0248162	1680.8	3.5
8 bits	0.959524	0.0295381	206.993	0.0259086	1680.8	4.4
12 bits	0.959524	0.0295381	197.816	0.0251492	1680.8	4.4
16 bits	0.959524	0.0295381	192.925	0.0246898	1680.8	4.4
20 bits	0.959524	0.0295381	194.153	0.0253868	1680.8	4.4
24 bits	0.959524	0.0295381	193.862	0.0252914	1680.8	4.4
28 bits	0.959524	0.0295381	193.8	0.0247105	1680.8	4.4
32 bits	0.959524	0.0295381	192.226	0.0248405	1680.8	4.4

Figura 20: Resultats del joc de proves bre

	snr.csv accuracy	snr.csv std_deviation	snr.csv train ms	snr.csv test ms	snr.csv cbps	snr.csv hash cbps
4 bits	0.555238	0.290126	14.659	0.005	1968	3.6
8 bits	0.568095	0.248168	25.707	0.004	1968	5.9
12 bits	0.539524	0.247967	31.6965	0.004	1968	6.8
16 bits	0.539524	0.247967	32.3531	0.002	1968	6.8
20 bits	0.539524	0.247967	32.3872	0.003	1968	6.8
24 bits	0.539524	0.247967	32.4443	0.005	1968	6.8
28 bits	0.539524	0.247967	32.3483	0.006	1968	6.8
32 bits	0.539524	0.247967	32.2347	0.007	1968	6.8

Figura 21: Resultats del joc de proves snr

	vot.csv accuracy	vot.csv std_deviation	vot.csv train ms	vot.csv test ms	vot.csv cbps	vot.csv hash cbps
4 bits	0.866032	0.0418572	15.9785	0.0073	836.1	4
8 bits	0.942336	0.0418572	29.1585	0.0078	836.1	6.6
12 bits	0.940063	0.0417495	30.58	0.0075	836.1	6.8
16 bits	0.942336	0.0418572	30.9811	0.0086	836.1	6.8
20 bits	0.937790	0.0389496	30.9	0.0079	836.1	6.8
24 bits	0.937790	0.0389496	30.6346	0.0079	836.1	6.8
28 bits	0.942336	0.0418572	30.9278	0.0079	836.1	6.8
32 bits	0.942336	0.0418572	30.5848	0.0076	836.1	6.8

Figura 22: Resultats del joc de proves vot

	cre.csv accuracy	cre.csv std_deviation	cre.csv train ms	cre.csv test ms	cre.csv cbps	cre.csv hash cbps
4 bits	0.832909	0.175594	187.272	0.017	4760.9	3.1
8 bits	0.829784	0.182736	262.331	0.014	4760.9	3.8
12 bits	0.829784	0.182736	283.194	0.013	4760.9	3.8
16 bits	0.829784	0.182736	306.551	0.015	4760.9	3.8
20 bits	0.829784	0.182736	261.997	0.016	4760.9	3.8
24 bits	0.829784	0.182736	261.751	0.014	4760.9	3.8
28 bits	0.829784	0.182736	262.939	0.015	4760.9	3.8
32 bits	0.829784	0.182736	263.19	0.011	4760.9	3.8

Figura 23: Resultats del joc de proves cre

Bibliografia

[1] O. Pujol i D. Masip "Geometry-Based Ensembles: Toward a Structural Characterization of the Classification Boundary". *IEEE Transactions on pattern analysis and machine intelligence.* vol.31.6(2009): 1140-1146

[2] V. Torra "Mòdul 1. Aprenentatge". *Apunts de l'assignatura d'Intel·ligència Artificial II.* P07/11007/02546(2007)

[3] K.R. Gabriel i R.R Sokal "A New Statistical Approach to Geographic Variation Analysis". *Systematic zoology.* vol.18(1969):259-270

[4] P.H.A. Sneath i R.R. Sokal "Numerical Taxonomy". San Francisco, CA, EUA: W.H. Freeman and Company, pg. 157.