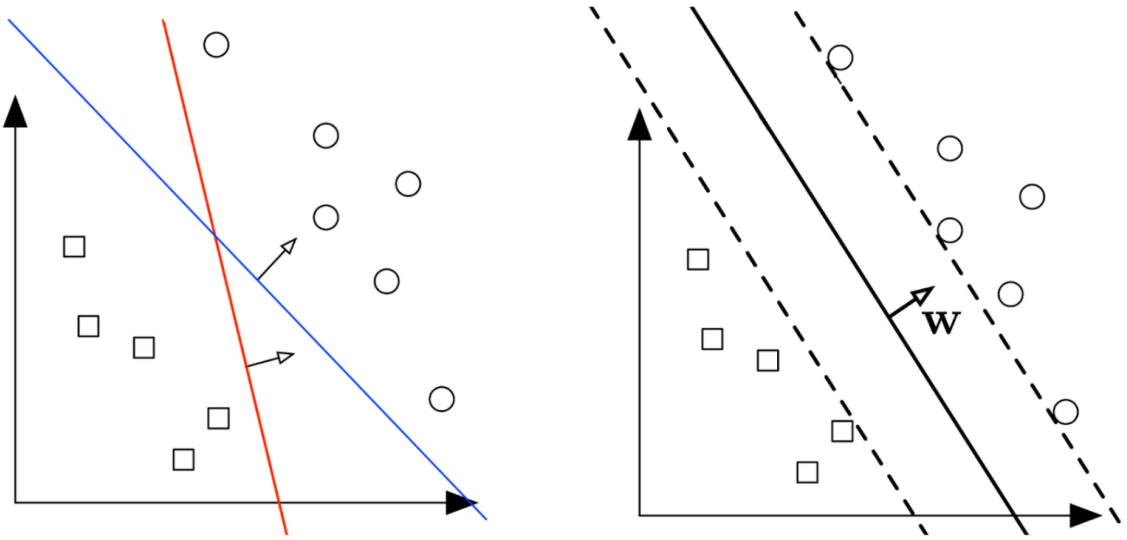
Support vector classifier in AMPL

Optimització Matemàtica: Lab assignment 2



Gabriel Fortuny Carretero i Pau Mateo Bernadó

Optimització Matemàtica

UPC - Grau en Ciència i Enginyeria de Dades

Curs 2023-2024 Q2

Taula de continguts

1. Introducció

En aquest segon entregable del curs de l'assignatura d'Optimització Matemàtica hem imlementat una *Support Vector Machine* en el llenguatge matemàtic AMPL, en les seves dues formulacions, la primal i la dual.

Posteriorment hem avaluat la nostra implementació amb diferents datasets: els generats per l'aplicació donada a l'enunciat del projecte, un dataset de dades reals (Wisconsin Breast Cancer Dataset) i alguns datasets generats amb funcions de la llibreria de python sklearn.datasets: *make_classification* i *make_swiss_roll*. En totes aquestes execucions hem volgut analitzar-ne el rendiment, guardant els valors de temps, precisió (*accuracy*) i el nombre d'iteracions. També hem mirat que els dos hiperplans generats (el del problema primal i dual) coincideixin. En addició, hem implementat una altra versió del problema dual del SVM amb un kernel Gaussià, per després aplicar-lo a datasets linealment no separables.

Així per tant, tenim els tres codis d'AMPL en format *.mod*, dos documents *.run* on hi ha les comandes per executar els programes, definir les variables, calcular l'hiperplà (ω i γ) i també calcular les prediccions i la precisió de cada execució. Tots aquests, juntament amb els datasets en format *.dat*, es troben a la carpeta zip adjunta a l'entrega.

Breu teoria de SVM

Les *Support Vector Machines* són un tipus de models discriminatius usats en problemes de classificació supervisada. Es basen en calcular un hiperplà per separar les dues classes del problema. L'espai d'entrada és un conjunt de m punts $x_i \in R^n$ i les seves etiquetes $y_i \in \{0, 1\}$. L'hiperplà el denotem per $w^T x + \gamma = 0$. Donat un conjunt d'entrenament, es vol trobar l'hiperplà que maximitza la separació d'aquest en les classes. Es pot demostrar que aquesta separació és exactament $2 \cdot ||w||^{-2}$. Per altra banda, per poder considerar punts mal classificats, considerem les variables $s_i \geqslant 0$. Amb aquesta formulació, el problema esdevé un problema d'optimització, on es vol minimitzar la següent funció per tal de maximitzar la separació i minimitzar, de forma ponderada amb ν, els errors.

$$\begin{aligned} \min_{w,\gamma,s \in \mathbb{R}^{N+1+m}} \quad & \frac{1}{2}||w||_2 + \nu \sum_{i=1}^{m} s_i \\ \text{s.t} \quad & y_i(w^\intercal \phi(x_i) + \gamma) \geq 1 \\ & s_i \geq 0 \end{aligned}$$

Aquesta és la formulació primal del problema d'optimització. El paràmetre ν pondera la importància que se li atribueix als punts mal classificats. Agafar un valor baix de ν implica donar més importància a trobar un hiperplà correcte i que maximitzi la separació d'aquest amb els grups, i un valor alt de ν vol dir donar més importància a agafar un hiperplà que classifiqui correctament els punts d'entrenament. Intuïtivament, el valor òptim de ν en quant a la precisió del pla (per exemple, l'*accuracy* sobre un conjunt de punts diferent al d'entrenament) és un valor ni molt gran ni molt petit.

És molt útil tractar les SVM amb la formulació dual d'aquest problema, ja que així ens permet aplicar diversos kernels que a la formulació primal no es pot usar, com el kernel Gaussià. El que fa aquest kernel és que envia l'espai d'entrada a un espai de dimensió "infinita", on les dades sí que són linealment separables per un hiperplà.

La formulació dual del problema és la següent.

$$\begin{aligned} \max_{\lambda} \quad & \lambda^\intercal e - \frac{1}{2}\lambda^\intercal Y A A^\intercal Y \lambda \\ \text{s.t.} \quad & \lambda^\intercal Y e = 0 \\ & 0 \leq \lambda \leq \nu \end{aligned}$$

Usant el kernel Gaussià, es substitueix $AA^T = K = [K(x_i, x_j)]_{i,j=1..m}$, on $K(x,y) = e^{-||x-y||^2/2\sigma^2}$ és el kernel Gaussià comentat. Aquest kernel no es pot usar amb la formulació primal perquè no es pot obtenir una funció $\phi(x)$ tal que $K(x,y) = \phi(x)^T\phi(y)$.

2. Modelatge

A continuació mostres les implementacions dels problemes primal, dual sense kerne i dual amb kernel.

Implementació problema primal

```
param n;
param m;
param nu;
param A_train{1..m,1..n};
param y_train{1..m};
param A_test{1..m,1..n};
param y_test{1..m};

var w {1..n};
var gamma;
var s {1..m};

minimize primal:
     (1/2) * sum{p in {1..n}}(w[p]^2) + nu*sum{b in {1..m}}(s[b]);

subject to c1 {p in {1..m}}:
     -y_train[p]*(sum{b in {1..n}}(A_train[p,b]*w[b]) + gamma) -s[p] + 1 <= 0;

subject to c2 {p in {1..m}}:
     -s[p] <= 0;
```

Implementació problema dual

```
param n;
param m;
param nu;
param y_train {1..m};
param A_train {1..m,1..n};
param y_test {1..m};
param A_test {1..m,1..n};

var la {1..m} >=0, <= nu;

maximize dual:
      sum{i in {1..m}}la[i]
      -(1/2)*sum{i in {1..m}, j in {1..m}}la[i]*y_train[i]*la[j]*
      y_train[j]*(sum{k in {1..n}}A_train[i,k]*A_train[j,k]);

subject to c1:
      sum{i in {1..m}}(la[i]*y_train[i]) = 0;
```

Implementació problema dual amb el kernel Gaussià

```
param n;
param m;
param nu;
param sigma;
param m_test;
param y_train {1..m};
param A_train {1..m,1..n};
param y_test {1..m};
param A_test {1..m,1..n};

var la {1..m} >=0, <= nu;

maximize dual:
      sum{i in {1..m}}la[i]
      -(1/2)*sum{i in {1..m}, j in {1..m}}la[i]*y_train[i]*la[j]*y_train[j]
      *(exp(-sum{k in {1..n}}((A_train[i,k]-A_train[j,k])^2) / (2*sigma^2)));

subject to c1:
      sum{i in {1..m}}(la[i]*y_train[i]) = 0;
```

3. Avaluació

3.1 Dataset sintètic linealment separable

En primer lloc, hem avaluat els codis dels problemes primal i dual del Support Vector Machine amb el solver cplex per a diferents combinacions de tamany del dataset i valor de ν.

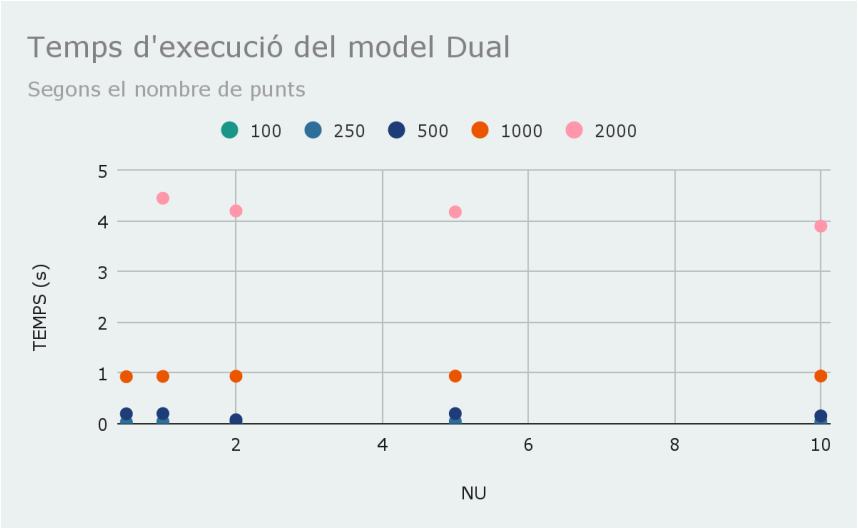
nombre de punts	100, 250, 500, 1000, 2000, 5000
ν	0.5, 1, 2, 5, 10

Cal destacar que hem calculat l'hiperplà de separació del SVM en totes les avaluacions. D'aquesta manera, hem pogut confirmar que en tots els casos el resultat del problema dual, aquest hiperplà calculat a partir de les lambdes que el resultat dona, coincideix amb el resultat òptim donat pel model primal. Per exemple, quan avaluem els dos problemes amb el conjunt de dades de 100 punts, tots els valors de ω^* i el de γ coincideixen. Aquesta sortida del programa svm.run on podem confirmar això es troba a l'annex; es pot veure que els valors per aquestes variables són iguals.

Per a aquest mateix cas, els valors pel temps d'execució, el nombre d'iteracions i la precisió (*accuracy*) per diferents valors de ν han resultat ser els següents:

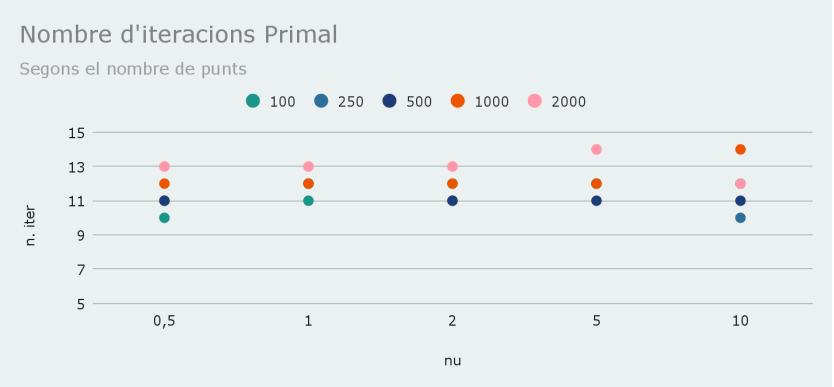
ν	**temps Primal**	**temps Dual**	**valor de la funció objectiu**	***accuracy***	**iteracions Primal**	**iteracions Dual**
0,5	0	0	28	0,86	10	10
1	0	0	49,26	0,88	11	13
2	0	0	87,88	0,88	12	13
5	0	0	195,86	0,88	12	12
10	0	0	370	0,88	12	13

El primer en què ens hem fixat és que l'ampl soluciona molt ràpidament el problema d'optimització. L'eina timing del solver cplex d'ampl ens permet saber el temps que tarda el problema en llegir l'input, resoldre el problema i en treure l'output. En el cas del dataset de 100 punts, el temps retornat és de zero segons, independentment del valor de ν emprat. Per això, ens ha interessat veure com creix el temps segons el nombre de punts. Hem graficat els valors del temps pel cas del model Dual, doncs el Primal es queda amb valors molt pròxims a zero en tots els casos, el qual fa que no es pugui observar tanta diferència.

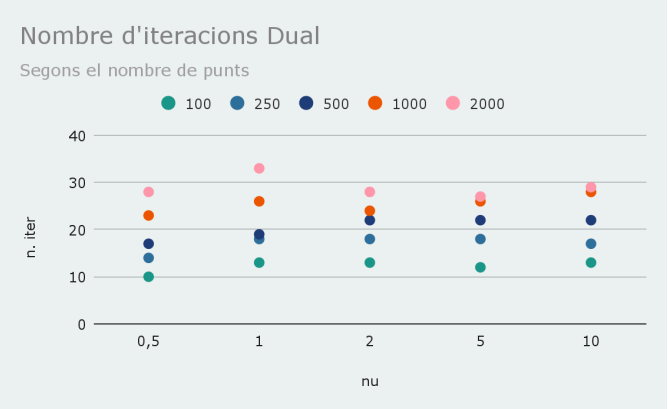


Com s'observa al gràfic, sembla ser que el valor de la ν no afecta el temps d'execució, doncs els punts es mantenen iguals al llarg de l'eix X. En contraposició, el temps d'execució sí que sembla dependre de la mida del fitxer d'entrada. Com més gran és aquest, més gran és la matriu que ha de calcular el problema dual, així que més segons triga. Així i tot, considerem que són temps molt ràpids, sent el major d'ells 4 segons i mig per resoldre un problema amb 2000 punts.

En la mateixa línia, hem volgut confirmar que el nombre d'iteracions també depèn de la mida del fitxer d'entrada. Per això hem graficat els valors del nombre d'iteracions. En el fitxer excel adjunt es poden trobar totes aquestes taules i gràfics que per no saturar no hem adjuntat en l'informe.

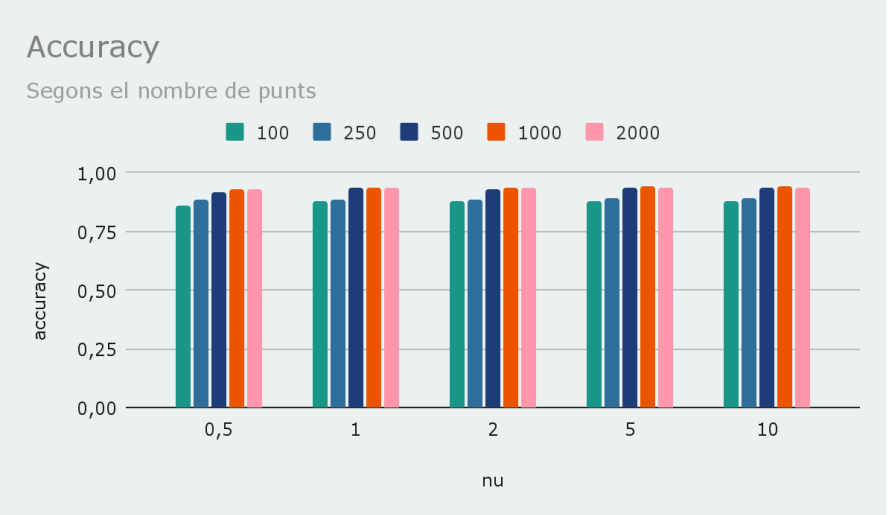


Així com ho havíem vist amb el temps d'execució, el nombre d'iteracions tampoc sembla tenir una dependència de la ν. La mida del fitxer, és a dir el nombre de punts, afecta més aquest valor. Els inputs majors fan que el programa iteri més vegades. Així i tot, tots els valors són semblants, hi ha poca variància. En el cas del model dual potser s'observa més un patró pel nombre d'iteracions en funció de la ν emprada.



Com podem observar, el nombre d'iteracions pel problema dual és major, en general, que pel primal. Sobretot en casos de gran nombre de punts. També existeix la relació creixent entre nombre de punts i d'iteracions. A més, observem el patró esmentat segons la ν, el qual sembla indicar que un valor igual a un és el que fa tenir un nombre d'iteracions major.

Finalment hem volgut estudiar la precisió del problema dual d'aquest apartat. Per això hem emprat els valors que el codi predeix d'un conjunt test independent del conjunt train i veure quants errors comet. Així, hem agafat els valors de precisió pels diferents valors de ν i de nombre de punts, obtenint els següents resultats.



Com podem observar, no sembla haver-hi una gran relació entre el valor de ν i la precisió, així com la diferència segons el nombre de punts del problema no sembla significativa. És per això que concloem que el programa executa igualment bé tota mida de fitxer. El valor de ν que ha obtingut major precisió és quan hem posat ν igual a 10. tot i que amb molt poca diferència. En relació al nombre de punts, el fitxer de 1000 punts és el que presenta millor precisió.

També hem fet una avaluació amb 5000 punts d'entrenament, però no l'hem inclòs a l'anàlisi perquè el problema dual només l'hem pogut executar per al cas de ν=1, ja que per als altres casos el solver aturava l'execució degut a massa memòria consumida. En aquest cas, el problema primal trigava de nou pràcticament zero segons, i el dual trigava 39 segons només en el temps d'*input*, esmentat anteriorment. Això és degut a que la formulació dual del problema ha de calcular la matriu de Gram AA^T, que per a molt punts d'entrada esdevé una matriu molt gran.

3.2 Dataset real: Breast Cancer Wisconsin

A continuació analitzarem el rendiment del classificador SVM amb un dataset real. Hem escollit el dataset de Breast Cancer Wisconsin, que és un dataset molt conegut en machine learning. Va ser introduït per W. Street, W. Wolberg i O. Mangasarian el 1993 amb l'article *Nuclear feature extraction for breast tumor diagnosis*, i pretén identificar cèl·lules de tumors com a benignes o malignes en funció de 30 característiques computades a partir d'imatges de *fine needle aspiration (FNA)*.

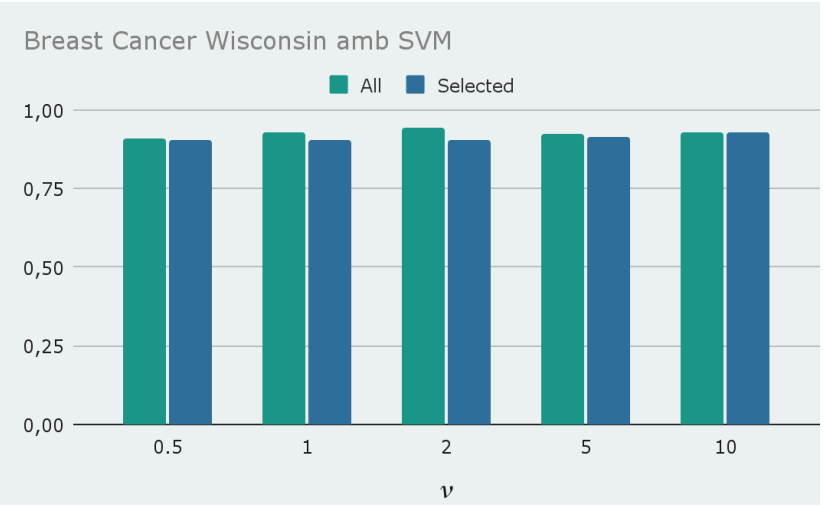
Per avaluar el classificador hem separat el dataset en un conjunt d'entrenament i un conjunt de test (75% train, 25% test), i hem entrenat el model amb el conjunt d'entrenament amb el solver cplex i després calculat les prediccions de la SVM sobre el conjunt de test, i hem calculat l'*accuracy* obtinguda per cada cas. Hem obtingut els següents resultats.

ν	temps Primal	temps Dual	*accuracy*	iter. primal	iter. dual
0.5	0	0,15	0.91	26	24
1	0	0,15	0,93	26	23
2	0,01	0,1	0,944	23	22
5	0,04	0,11	0,923	24	22
10	0,03	0,15	0,93	24	21

Podem veure que les precicions obtingudes són bastant altes, prenent el valor més alt amb ν = 2. En general, el problema primal fa una mica més d'iteracions que el dual, però segueix sent bastant més ràpid.

Hem provat també el mateix dataset però amb menys variables per veure si afectava als resultats. Hem fet una selecció de les 11 variables més significatives usant *Recursive Feature Elimination with Cross Validation* (RFECV) amb Random Forest de la llibreria *sklearn* de Python. Al següent gràfic es pot veure la precisió dels dos casos (veure document adjunt pels resultats més detallats, en les taules).

Es pot veure que la precisió amb totes les variables és un pèl més alta per als tres casos intermitjos, però en general és pràcticament igual. Els temps d'execució també són pràcticament els mateixos, ja que el tamany del problema és massa petit com per notar diferència al temps d'execució entre usar 30 variables i 11.

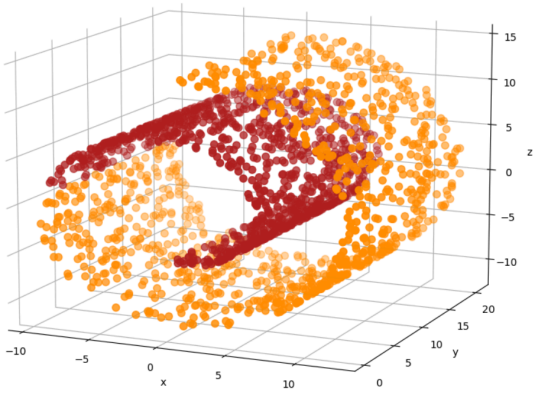


Hem provat també d'aplicar el SVM en un dataset sintètic linealment separable, creat a partir de la funció

make_classification() de la llibreria sklearn.datasets de Python. Els resultats es poden trobar a les taules del document adjunt. El més important a destacar és que hem confirmat els resultats obtinguts amb el primer tipus de dataset. El temps i el nombre d'iteracions depenen del nombre de punts del fitxer d'entrada, però no així l'*accuracy*, que en aquest cas ha estat més irregular que abans, amb resultats diferents per diferents datasets.

3.3 Dataset linealment no separable

En darrer lloc, hem avaluat el classificador amb el kernel gaussià implementat amb un dataset linealment no separable amb el solver cplex. Hem generat un conjunt de punts sintètics de tipus Swiss Roll, amb la comanda make_swiss_roll() de la llibreria skelarn.datasets de Python. Es tracta d'un conjunt de punts tridimensionals que s'assemblen a la forma d'un braç de gitano. La següent figura mostra les dades que hem generat, amb els dues classes dels punts de diferents colors.



Ja es pot veure que sobre aquest conjunt de punts una SVM com la que hem implementat no resultarà útil, però si s'usa un Kernel, com per exemple el Kernel Gaussià, el problema canvia totalment i la SVM pot obtenir molt bons resultats.

Per aquest apartat també hem avaluat la nostra implementació per a diferents combinacions de valors de ν i σ:

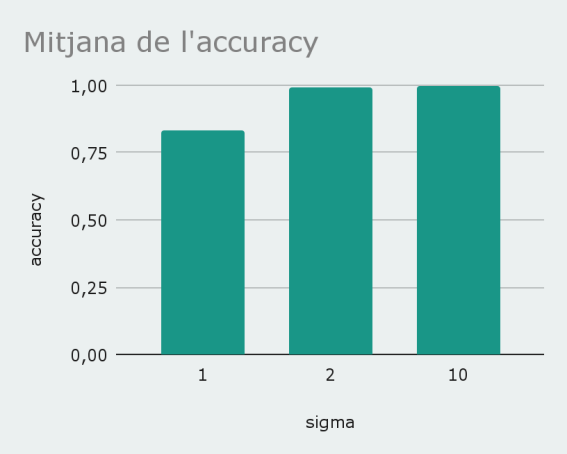
nombre de punts	100, 400, 1200, 2000
valors de ν	2, 50
valors de σ	0.5, 1, 10

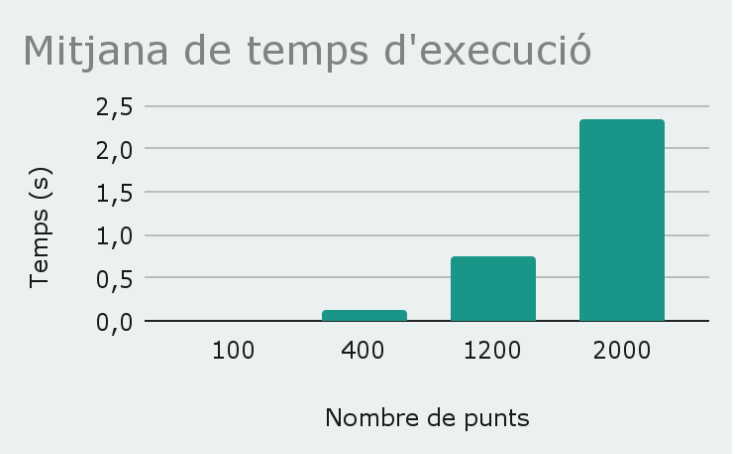
Per exemple, aquests són els resultats obtinguts per al cas de 400 punts amb ν = 10.

σ	**temps total (s)**	**nombre d'iteracions**	***accuracy***
1	0,09375	11	0,9975
2	0,15625	12	1
10	0,15625	16	1

El primer que ens ha sobtat ha estat l'alta precisió d'aquest programa. En molts casos hem obtingut una precisió del 100% i en la resta dels casos precisions superiors a 0.90. Això ens

indica que la implementació d'un kernel en el Support Vector Machine implica una millora en la separació de les dues classes. Hem dut a terme diferents combinacions dels valors de σ i ν per veure si afecten en el temps, nombre d'iteracions o precisió. Com ha passat anteriorment, no semblen tenir gran pes sobre els paràmetres guardats.



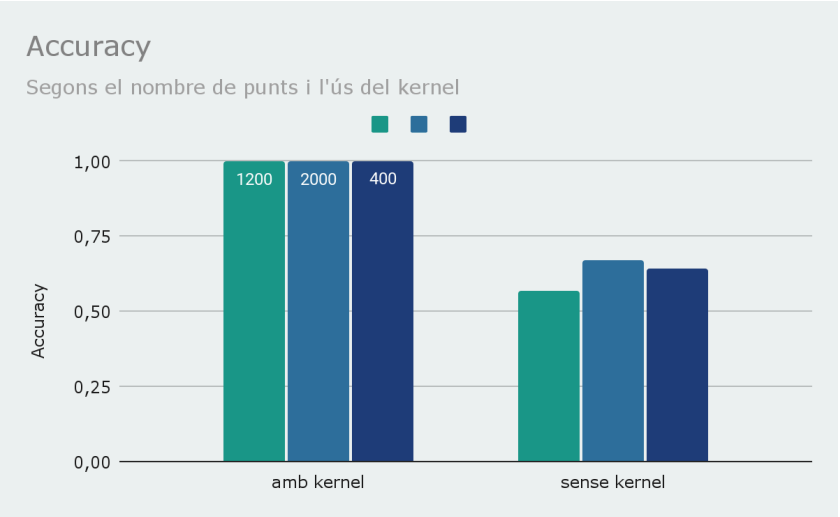


Observant les dades que hem agafat de diferents execucions, les quals estan guardades al fitxer adjunt, hem confirmat que la ν no té efecte en el temps d'execució ni en el valor de la precisió. No és així per la σ, la qual hem observat que afecta la precisió. Com es pot observar al gràfic de l'esquerra, en mitjana a mesura que el valor de σ creix la precisió s'ha incrementat. Així i tot, degut a que els valors de precisió eren molt grans, sovint iguals a un, tampoc es tracta d'un increment massa significatiu. En addició, així com passava pel primer dataset, el temps d'execució i el nombre de punts són directament proporcionals. El programa tarda zero segons en executar el problema sobre un conjunt de 100 punts però passa dels dos segons quan el fitxer en té 2000. El mateix succeeix amb el nombre d'iteracions.

Finalment, hem volgut comparar el rendiment del model dual amb kernel gaussià amb l'implementat a la primera part, sense el kernel. Per això hem executat ambdós programes amb els mateixos paràmetres per comparar-los, obtenint aquests valors.

nombre de punts	**ús del kernel**	**temps total (s)**	**nombre d'iteracions**	***accuracy***
1200	amb kernel	0,703125	12	0,99875
	sense kernel	0,5	16	0,57
2000	amb kernel	3,3125	26	0,998182
	sense kernel	1,015625	16	0,67
400	amb kernel	0,140625	13	1
	sense kernel	0,109375	14	0,64

La major diferència entre emprar el kernel i no emprar-lo, com esperàvem, es troba en la precisió. En el gràfic es pot observar millor.



Com es veia a la taula, els valors de la precisió es veuen molt millorats si es fa ús d'un kernel gaussià quan el dataset és linealment no separable. Pel que fa a les altres mètriques, el temps d'execució i el nombre d'iteracions, la diferència no és tan clara ni important. Tot i així, en general per datasets majors el model dual amb kernel incrementa més en nombre d'iteracions que el model dual sense kernel. Això també es tradueix en el temps d'execució, que va molt lligat al nombre d'iteracions.

4. Conclusions

Hem vist que les dues formulacions obtenen, per a tots els casos, el mateix hiperplà de separació. Per a datasets amb pocs punts d'entrenament (menys de 500 punts) tant el primal com el dual triguen molt poc, però a partir de 500 punts el dual comença a trigar molt, sobretot al calcular la matriu de productes creuats.

Per al dataset generat amb el codi que se'ns ha proporcionat hem obtingut bastants bons resultats en quant a precisió. Per aquest dataset el problema dual tendeix a fer més iteracions que el primal.

Per al dataset de Wisconsin hem obtingut també molt bona *accuracy* considerant que és un dataset de dades no sintètiques i no hem usat cap kernel. Amb aquest dataset les iteracions han estat al revés: el primal feia lleugerament més iteracions que el dual. Hem vist també que obviant pràcticament dos terços de les dimensions del problema (variables) obtenim resultats molt similars, però no millors.

Per últim, a les dades no linealment separables hem vist que el kernel combinat amb la SVM fa una eina de classificació molt potent. La diferència entre usar i no usar el kernel és molt gran. Sense usar el kernel amb prou feines es poden obtenir millors resultats que un classificador aleatori, i amb el kernel hem obtingut per a gairebé tots els casos presicions de més del 99%. Tanmateix, usant el kernel el tamany del conjunt d'entrenament afecta més al temps d'execució.

5. Annex

Sortida del programa svm.run pel cas del fitxer 100_svm.dat, un fitxer amb 100 punts de train i 100 de test, de 4 coordenades. Podem observar com l'hiperplà òptim (en negreta) trobat pels dos models, primal i dual, coincideix.

```
> ampl: include svm.run
PRIMAL_SVM:
CPLEX 22.1.1.0: timing=1

Times (seconds):
Input =  0
Solve =  0
Output = 0
CPLEX 22.1.1.0: optimal solution; objective 87.88214009
12 separable QP barrier iterations
No basis.

w [*] :=
1  2.55198
2  2.15898
3  2.19971
4  3.37397
;

gamma = -5.03941
```

```
s [*] :=
  1 4.66238e-11     26 0.684777       51 4.70372e-11     76 0.295913
  2 4.7248e-11      27 4.94129e-11    52 0.153992        77 4.88508e-11
  3 4.61304e-11     28 4.76875e-11    53 0.414024        78 0.590236
  4 4.7026e-11      29 4.77575e-11    54 0.887638        79 1.50101
  5 4.71519e-11     30 2.0924e-10     55 4.47901e-11     80 4.72602e-11
  6 0.125874        31 9.20342e-11    56 9.66993e-11     81 4.70233e-11
  7 0.654626        32 0.428751       57 4.69839e-11     82 0.403622
  8 1.81544         33 4.74371e-11    58 4.69159e-11     83 4.82414e-11
  9 2.13986         34 0.239719       59 4.73402e-11     84 4.28917e-10
 10 4.61111e-11     35 4.59664e-11    60 0.863206        85 4.75366e-11
 11 4.78614e-11     36 4.89182e-11    61 0.290356        86 4.70685e-11
 12 4.68868e-11     37 0.23356        62 0.866618        87 0.464852
 13 0.0235          38 4.64047e-11    63 1.38258         88 4.61655e-11
 14 5.1693e-11      39 2.85141        64 0.901043        89 4.6608e-11
 15 0.0920602       40 1.89406        65 1.22369         90 0.793006
 16 0.4051          41 4.66834e-11    66 4.58139e-11     91 0.197278
 17 4.64188e-11     42 0.245624       67 4.77323e-11     92 1.2839
 18 4.83355e-11     43 4.63133e-11    68 4.80349e-11     93 0.0278238
 19 1.21724         44 7.0684e-11     69 4.58358e-11     94 1.46781
 20 0.474614        45 0.0650529      70 1.47339         95 0.427816
 21 4.72436e-11     46 4.73697e-11    71 4.7042e-11      96 0.16195
 22 4.72416e-11     47 0.00101333     72 1.00973         97 4.65699e-11
 23 0.203358        48 4.61943e-11    73 3.98094         98 4.79107e-11
 24 4.73984e-11     49 0.72544        74 4.85268e-11     99 0.215605
 25 0.308971        50 0.051144       75 4.73097e-11    100 0.932808
;

DUAL_SVM:
CPLEX 22.1.1.0: timing=1

Times (seconds):
Input =  0
Solve =  0.015625
Output = 0
CPLEX 22.1.1.0: optimal solution; objective 87.88214005
13 QP barrier iterations
No basis.
```

```
la [*] :=
  1 1.9856e-10     26 2             51 3.51834e-10    76 2
  2 1.55648e-10    27 9.63582e-11   52 2              77 1.97837e-10
  3 2.20055e-10    28 8.23395e-10   53 2              78 2
  4 2.43181e-10    29 2.00889e-09   54 2              79 2
  5 3.16718e-10    30 1.35864       55 2.91542e-10    80 1.54405e-10
  6 2              31 0.813377      56 1.03669        81 2.34552e-10
  7 2              32 2             57 2.75078e-10    82 2
  8 2              33 6.08642e-10   58 2.24207e-09    83 2.32376e-10
  9 2              34 2             59 7.32487e-10    84 1.75342
 10 4.84578e-10    35 4.40959e-10   60 2              85 7.60416e-10
 11 2.00209e-09    36 1.31691e-10   61 2              86 8.80589e-10
 12 4.84771e-10    37 2             62 2              87 2
 13 2              38 1.47868e-09   63 2              88 4.16901e-10
 14 2.99946e-09    39 2             64 2              89 3.57248e-10
 15 2              40 2             65 2              90 2
 16 2              41 4.64553e-10   66 3.09285e-10    91 2
 17 1.67938e-08    42 2             67 2.06923e-10    92 2
 18 2.23999e-10    43 2.92551e-10   68 1.48792e-10    93 2
 19 2              44 0.618095      69 2.88333e-10    94 2
 20 2              45 2             70 2              95 2
 21 2.59983e-10    46 1.97657e-09   71 2.3307e-10     96 2
 22 8.85885e-10    47 2             72 2              97 1.97691e-09
 23 2              48 3.92229e-10   73 2              98 2.34292e-10
 24 6.6325e-10     49 2             74 5.02156e-10    99 2
 25 2              50 2             75 3.42207e-10   100 2
;

w [*] :=
1  2.55198
2  2.15898
3  2.19971
4  3.37397
;

gamma = -5.03941
```

```
prediction [*] :=
  1  1    13  1    25  1    37  1    49 -1    61  1    73  1    85 -1    97 -1
  2  1    14  1    26 -1    38 -1    50  1    62  1    74 -1    86  1    98  1
  3  1    15  1    27 -1    39  1    51 -1    63 -1    75  1    87 -1    99  1
  4  1    16  1    28  1    40  1    52 -1    64 -1    76 -1    88  1   100  1
  5  1    17 -1    29 -1    41  1    53  1    65  1    77  1    89 -1
  6  1    18  1    30 -1    42 -1    54  1    66 -1    78 -1    90 -1
  7 -1    19  1    31 -1    43  1    55 -1    67  1    79 -1    91 -1
  8 -1    20 -1    32  1    44 -1    56  1    68 -1    80  1    92  1
  9 -1    21  1    33 -1    45  1    57  1    69 -1    81 -1    93  1
 10  1    22  1    34  1    46  1    58 -1    70 -1    82 -1    94 -1
 11 -1    23  1    35 -1    47 -1    59 -1    71 -1    83  1    95 -1
 12 -1    24 -1    36  1    48  1    60 -1    72  1    84 -1    96  1
;

errors = 12

accuracy = 0.88
```

La resta de dades explicades i agafades es troben al fitxer excel adjunt.