Práctica 4: Máquinas de vectores soporte

Introducción a los modelos computacionales.

Grado en Ingeniería Informática, Escuela Politécnica Superior de Córdoba, Universidad de Córdoba.

4º Curso.

Realizado por: José Pérez-Parras Toledano
Dni: 31014082P
Email: i32petoj@uco.es

Índice:

Preguntas:

Representación 2D de las SVM

1. **Abre este *script* y explica su contenido. Podrás ver que se utiliza el primer *dataset* de ejemplo y se realiza la representación gráfica del SVM. Comenta que tipo de kernel se está utilizando y cuáles son los parámetros de entrenamiento.**

En este script lo que haremos será usar la librería *sklearn* y usaremos vectores soporte para la clasificación de nuestros *datasets*. Posteriormente se realizará una representación gráfica de dicha clasificación con los patrones y un vector lineal.
Abriendo este *script* y mirando las funciones a las que se llama y cómo se usan, se puede ver cómo el primer paso será cargar el dataset, que en este caso será el primero *dataset1.libsvm*. Después se entrenará el modelo SVM, que en este caso tendrá un kernel lineal, con C = 1 y gamma = 100. Estos son los parámetros de entrenamiento.

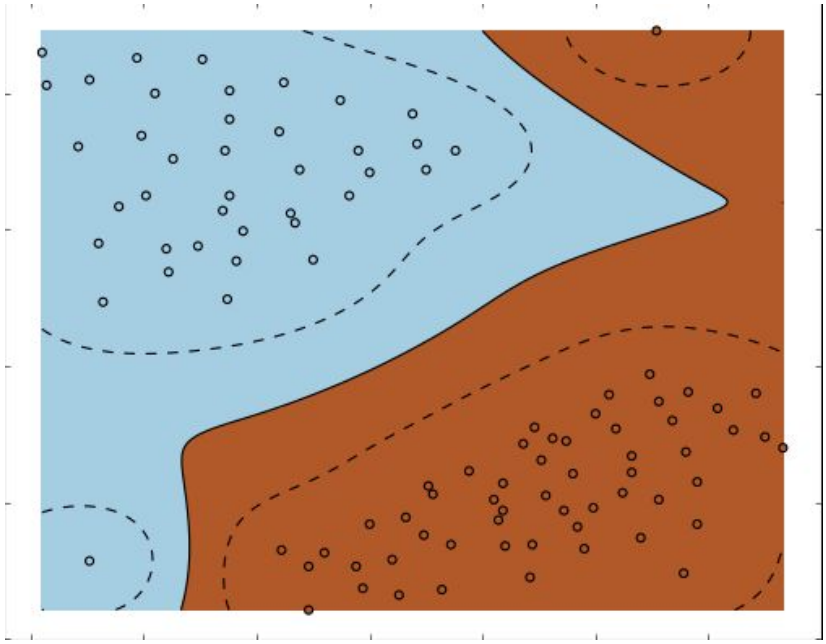
```
svm_model = svm.SVC(kernel='rbf',C=1,gamma=100)
```

Todo esto para después mostrar los patrones por pantalla junto al clasificador SVM, para que se pueda ver cómo ha clasificado los distintos patrones de las distintas clases.
Para ello usaremos un plot, en el que cada punto será un patrón en el plano y el SVM el separador.

Primer *Dataset* de ejemplo

2. **Intuitivamente, ¿qué hiperplano crees que incurrirá en un menor error de *test* en la tarea de separar las dos clases de puntos?.**

El mejor hiperplano que puede separar estos patrones será uno no lineal que que separe todos los patrones de una clase y todos los patrones de otra, utilizando el script con rbf, nos da exáctamente lo que buscamos.



3. **Debes de configurar el *script* para que se ejecute una SVM lineal (es decir, que el *kernel* sea lineal).**

Para este apartado deberemos modificar el *script*, concretamente la línea de código en la que se declara dicho SVM con sus parámetros, deberemos de cambiar el kernel. El cual mirando la documentación aportada por sklearn, puede tener los siguientes valores: 'rbf', 'linear', 'poly', 'sigmoid' o 'precomputed'. Para este caso usaremos 'linear', que será el lineal. Cabe destacar que dicho parámetro por defecto es 'rbf', además algo curioso es que todos son strings. Modificando el kernel para que sea lineal, dicha línea de código quedará de la siguiente manera:

```
svm_model = svm.SVC(kernel='linear',C=1,gamma=100)
```

4. **Modifica el *script* para que se prueben varios valores de *C*, en concreto, *C* $\{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}\}$. Observa qué sucede, explica por qué y escoge el valor más adecuado.**

Se puede observar que cuanto más aumenta el valor de C menos error se comete, puesto que al penalizar más los errores, se crea un modelo mejor para la clasificación. Para los valores de 10^{-2} hasta 1, se cometen errores es más para el valor de 10^{-2}, se clasifica todo en una misma clase. Según lo que he podido observar, el valor óptimo es 10^{3}, puesto es el modelo que realiza la mejor separación entre las dos clases y no comete error alguno, aunque para 10^{2} tampoco comete ningún error, la línea usada para separar queda muy cerca de dos patrones que están bastante alejados de la media de patrones de sus respectivas clases. Por eso he pensado que sería mejor utilizar el valor de 10^{3}, para este modelo lineal.

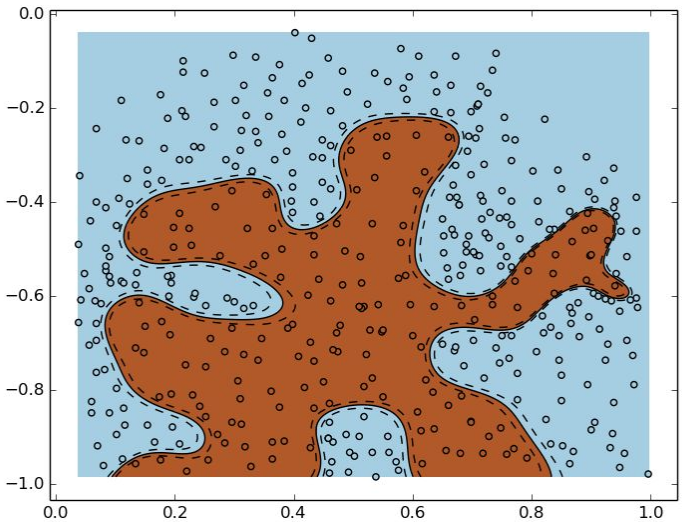
Segundo *dataset* de ejemplo

5. **Prueba a lanzar una SVM lineal con los valores para *C* que se utilizaron anteriormente. ¿Consigues algún resultado satisfactorio en el sentido de que no haya errores en el conjunto de entrenamiento?. ¿Por qué?.**

Para este dataset he probado los valores desde 10^{-2} hasta 10^{4} y en todos ocurre lo mismo, nuestro modelo no es capaz de separar los distintos patrones de las clases. Es decir, sin importar el valor de C no se consigue algún resultado satisfactorio puesto que no hay distinción de clases. Esto puede ser debido a que resulta imposible dividir dichos patrones en las distintas clases de forma lineal. Por lo que tendremos que usar modelos no lineales, como los que tienen base RBF o Gaussiana, que serán más complejos pero mucho mejores para esta tarea de dividir dichos patrones.

6. **Propón una configuración de SVM no lineal (utilizando el kernel de tipo RBF o Gaussiano) que resuelva el problema. El resultado debería ser similar al de la Figura 3. ¿Qué valores has considerado para *C* y para γ?. Además, incluye un ejemplo de una configuración de parámetros que produzca sobre-entrenamiento y otra que produzca infra-entrenamiento.**

Para este dataset como ya dijimos en la pregunta anterior, nos resultaba imposible obtener algún resultado satisfactorio de forma lineal, así que hemos optado por usar configuraciones no lineales. Dado que ya usamos las neuronas RBF en la práctica anterior me he decantado por esta configuración, dejando la γ tal cual estaba, es decir, 100 pero usando C = 10^{3}. Y este ha sido el resultado obtenido:

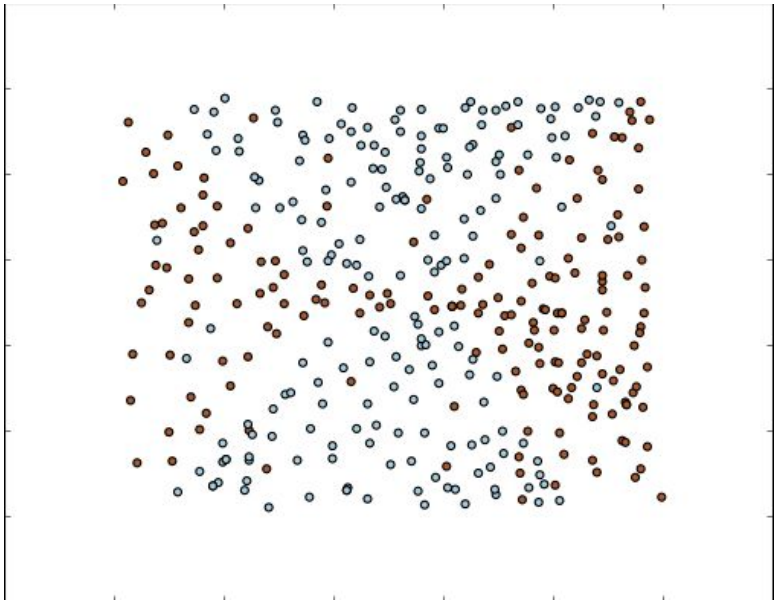


Este resultado es mejor que el resultado que se ve en el guión puesto que nuestro modelo también ha cogido un outlier de la clase y lo ha incluido en esta representación.

Tercer *dataset* de ejemplo

7. **En este caso, ¿es el *dataset* linealmente separable?. A primera vista, ¿Detectas puntos que presumiblemente sean *outliers*?, ¿por qué?.**

Esta pregunta es con el tercer *dataset* por lo que cambiaremos el código para que utilice dicho dataset, lo representaremos sin ningún tipo de SVM. Para ver si dicha base de datos es linealmente separable, el resultado representado es el siguiente:

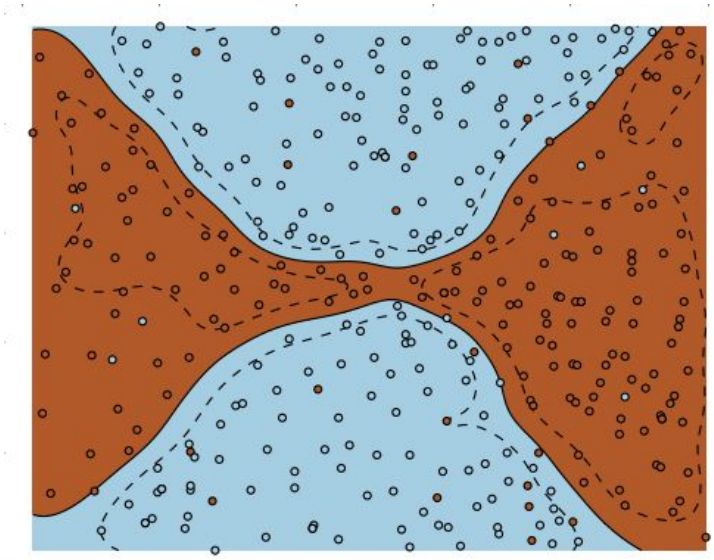


Como se puede ver los patrones de este *dataset*, no pueden ser separados linealmente puesto que están todos entremezclados unos con la otra clase, además se pueden ver distintos outliers de las clases cuando unos se sobreponen sobre otros, tanto arriba de la figura como abajo. No solo de una clase sino de ambas. Y por la forma que tienen estas clases podemos deducir que no son linealmente separables .

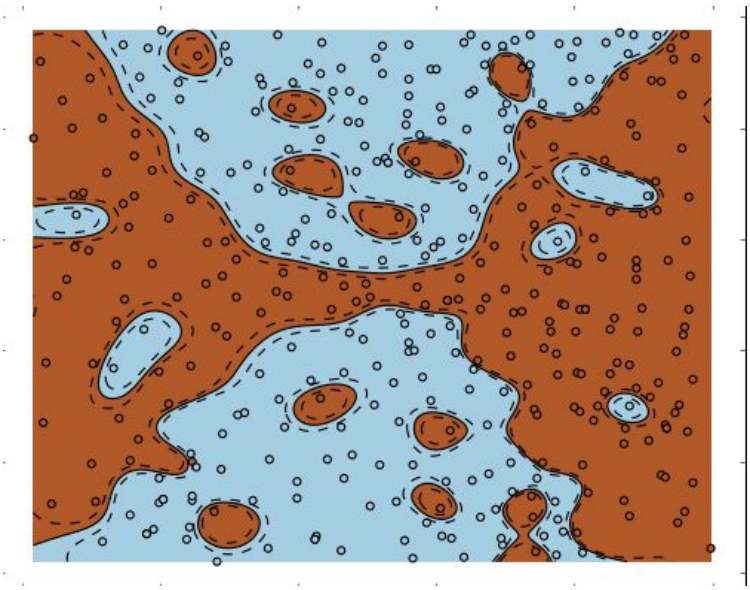
8. **Lanza una SVM para clasificar los datos, con objeto de obtener un resultado lo más parecido al de la *Figura 5*. Ajusta el ancho del *kernel* en el intervalo $\gamma \in \{0{,}02, 0{,}2, 2, 200\}$. Ajusta el parámetro de coste en el intervalo $C \in \{0{,}02, 0{,}2, 2, 200\}$. Establece el valor de los parámetros óptimos. Además, incluye un ejemplo de una configuración de parámetros que produzca sobre-entrenamiento y otra que produzca infra-entrenamiento.**

Para estas pruebas he decidido usar **rbf**, para ver que resultados nos daba y quedarnos con el mejor resultado de todas las pruebas. Por ejemplo para todas las pruebas en las que gamma era igual a 0,02, nuestro modelo está infra entrenando puesto que no era capaz de separar ambas clases en el resultado, obteniendo en la mayoría de casos resultado alguno sobre el que trabajar para la separación.

Podemos destacar que para valores de gamma y C bajos, no se obtiene resultado alguno, esto ha pasado en todas las pruebas. Para C = 200 y gamma = 0,2 ya vamos atisbando lo que va a ser nuestra solución final, pero aún queda mejorar un poco el modelo aumentando dicha gamma.
Con gamma =2, este modelo iba mejorando más notoriamente al crecer C. Aunque para esta configuración de gamma igual a 2 no hemos obtenido el mejor resultado, ahora nos falta las últimas pruebas con gamma igual a 200, en la que espero encontrar los mejores resultados, pero también un sobreentrenamiento cuando tengamos un valor de C muy alto.
Esta ha sido la mejor configuración encontrada, en la que se puede apreciar cómo no se han distinguido hasta outliers de ambas clases, para C = 0,2 y gamma = 200. Puesto que si clasificara bien los outliers, estariamos sobreentrenando el modelo.



Mientras que para C=200 y gamma = 200, se puede observar un sobre-entrenamiento demasiado claro, puesto que tiene una salida perfecta, pero únicamente sería útil para dichos patrones debido a las islas que forma alrededor de los patrones que son outliers. Esto solo viene bien para unos patrones dados como lo son estos.



Interfaz de consola

9. **Vamos a reproducir este proceso en Python. Divide el *dataset* sintético *dataset3.libsvm* en dos subconjuntos de forma estratificada, con un 80% de patrones en *train* y un 20% de patrones en *test*. Realiza el proceso de entrenamiento especificado anteriormente, utilizando los valores de *C* y γ obtenidos en el último ejercicio. Comprueba el porcentaje de buena clasificación que se obtiene para el conjunto de test.**

Para la realización de la división de ese conjunto de datos he hecho uso de la herramienta **Weka**, en la que podemos dividir un conjunto de datos que queramos indicando el % por ciento con el que nos queremos quedar. De la siguiente forma:

In the Explorer just do the following:

- training set:
  - Load the full dataset
  - select the `RemovePercentage` filter in the preprocess panel
  - set the correct percentage for the split
  - apply the filter
  - save the generated data as a new file
- test set:
  - Load the full dataset (or just use undo to revert the changes to the dataset)
  - select the `RemovePercentage` filter if not yet selected
  - set the `invertSelection` property to true
  - apply the filter
  - save the generated data as new file

Una vez realizado esto, ya tenemos nuestro conjunto de entrenamiento y test listo para ser usado. Ahora haremos un nuevo script para que podamos pasarle los patrones de nuestra base de datos de train para que entrene y nos de el CCR de la base de test. Aunque hay otra forma más sencilla de realizar esto y es usando la clase *StratifiedShuffleSplit* de la librería de **sklearn**, con dicha clase podremos indicar en cuánto porcentaje queremos dividir nuestro conjunto de datos, y además esto cuenta con la ventaja de que cada vez que ejecutemos nuestro script, se usarán distintos patrones tanto para entrenamiento como para test puesto que al ser la división aleatoria, esta cambiará en cada ejecución.

```
#Ahora estratificamos
sss = StratifiedShuffleSplit(y, test_size=0.2, train_size=0.8)
```

Esto hemos podido comprobarlo al ver el CCR obtenido en test, el cual unas veces obtenía cerca a 90% de CCR mientras que otras veces obtenía un 80% de CCR.

Cabe destacar que al calcular el CCR con los fichero que obteniamos de **Weka**, este salía muy bajo y creo que esto puede ser debido a que Weka divide el fichero cogiendo los patrones en orden y como en nuestro fichero los patrones de la clase dos están al final estos iban para el conjunto de test y claro al no haber entrenado con patrones de la clase dos pues obteníamos un CCR bastante bajo al únicamente tener que clasificar dicha clase sin apenas haberla entrenado.

10. Amplía el código anterior para realizar el entrenamiento del ejercicio anterior sin necesidad de especificar los valores de *C* y γ. Compara los valores óptimos obtenidos para ambos parámetros con los que obtuviste a mano. Extiende el rango de valores a buscar si es lo que consideras necesario.

Para esta pregunta y este script haremos uso de del objeto *GridSearchCV* de la librería **sklearn**, dicho objeto nos ayudará a realizar las pruebas para obtener los valores óptimos de C y gamma, para ello le pasaremos los valores de C y gamma que probamos en las preguntas anteriores para que realice las pruebas, retornándonos el modelo más óptimo que ha encontrado.
Una vez tengamos dicho modelo óptimo haremos uso otra vez de un 80% del conjunto de train y un 20% de test para poder obtener el CCR y obtenerlo con el de la pregunta anterior, aunque dicho CCR variará según los patrones aleatorios que se escogen al realizar dicha división.
Una vez realizado el script, he procedido a realizar distintas ejecuciones puesto que el CCR puede variar de una ejecución a otra debido a los patrones que se escogen para el conjunto de entrenamiento y test al realizar la división en dichos conjuntos.
El CCR obtenido en las pruebas está siempre bastante más cercano al 90%, y varía mucho menos que el que obteniamos con nuestros parámetros a mano, es decir, oscila menos aún teniendo dicha aleatoriedad. Además de que no existe un sobreentrenamiento al utilizar los parámetros óptimos para dichos patrones, un error en el que podríamos caer si estuviéramos cambiando los parámetros a mano.

11. ¿Qué inconvenientes observas en ajustar el valor de los parámetros "a mano", viendo el porcentaje de buena clasificación en el conjunto de test?.

El principal problema es que las pruebas "a mano" consumen mucho más tiempo que las pruebas realizadas automáticamente puesto que tienes que ir cambiando los parámetros y apuntando los resultados, por lo que nunca vas a abarcar un campo tan amplio de parámetros como el de una prueba automatizada que vaya variando los distintos parámetros para obtener el mejor modelo. Además que obteniendo en valor a mano, podemos caer en la trampa de un sobreentrenamiento, en el que por casualidad estemos obteniendo muy buenos resultados y no nos demos cuenta de dicho sobreentrenamiento, por lo que este método es mucho mejor además de probar muchas más configuraciones en un tiempo irrelevante para estos problemas con tan pocos patrones.

Base de datos *Digits*

12. **En primer lugar, necesitarás convertir ficheros tipo *arff* (formato de Weka) al formato de *libsvm*.**

Para convertir dichos ficheros a *libsvm* volveremos a hacer uso del programa **Weka**, con el que directamente abriendo nuestros ficheros en arff, nos dará la opción para guardarlos como *libsvm*, con lo que ello conlleva. Haremos esto con los ficheros y ya los tendremos guardados como libsvm y listos para ser manipulados por nuestros scripts.

13. **Una vez hayas convertido los ficheros, utiliza el *script* que desarrollaste en el ejercicio anterior para entrenar esta base de datos. Observe el valor de *CCR* obtenido para el conjunto de generalización y compáralo con el obtenido en prácticas anteriores. El proceso puede demorarse bastante. Al finalizar, toma nota de los valores óptimos obtenidos para los parámetros.**

Para esta prueba he utilizado el mismo vector de parámetros de C y gamma que para el ejercicio anterior, y aunque para esta base de datos el proceso se demora más tiempo, tampoco es tan excesivo, pero si se nota el gran número de patrones que ha de clasificar, además este tiempo crece conforme a las validaciones que ha de hacer.

Bueno comparando con el CCR de la práctica anterior, escogiendo el mejor CCR que obtuvimos probando los distintos parámetros como el número de neuronas y los valores de eta y gamma, además del tipo de regularización, fue: **94,858934%.**
Mientras que para esta ejecución en que nuestro objeto de la librería **sklearn** nos ha obtenido los parámetros óptimos, ha sido: **95.611285%.**
Por lo que ha sido pequeñamente superior, pero la gran ventaja es que no hemos tenido que ir haciendo pruebas a mano para obtenerlo, sino que lo ha calculado directamente y ha obtenido los parámetros óptimos para dicha base de datos, los cuales se pueden ver en el objeto y han sido: **C = 8 y gamma = 0.015625.**

14. **Localiza dónde se especifica el valor de *K* para la validación cruzada interna y el rango de valores que se han utilizado para los parámetros *C* y γ. ¿Cómo podrías reducir el tiempo computacional necesario para realizar el experimento?. Prueba a establecer *k* = 3, *k* = 5, *k* = 10 y compara, utilizando una tabla, los tiempos computacionales obtenidos y los resultados de *CCR* en test.**

Tanto los parámetros de C como de gamma, y la validación cruzada se los especificamos al crear dicho objeto.
Tanto el parámetro C com gamma serán un array de valores sobre los que dicho modelo ha de variar sus parámetros para obtener el mejor modelo y quedarse con los parámetros óptimos que le han ayudado a conseguirlo.

Y para el valor de **K** en la validación es lo mismo, se le ha pasar por parámetro, dicho valor corresponde al número de *folds* que se crean en la base de datos para después usarlos como *train* y *test*.

```
optimo = GridSearchCV(estimator=svm_model, param_grid=dict(C=Cs, gamma = Gs),n_jobs=-1, cv=5)
```

Ahí se ve el ejemplo de la configuración básica.
Cabe destacar que cuanto mayores sean los arrays de C y gamma, más pruebas se han de tener que realizar para la obtención del mejor modelo, así como a mayor número de *folds* se realizarán más validaciones lo que conlleva un mayor tiempo de cómputo, que para esta base de datos aún siendo una real, no llega a ser tan grande, pero aún así ya se va viendo cómo puede el tiempo computacional afectar y llegar a ser un lastre.

Para ello se ha hecho una comparación obteniendo el tiempo computacional y el CCR obtenido, variando para ello el número de *folds*, he aquí los resultados:

	k = 3	**k = 5**	**k = 10**
T. computacional (s)	64,27	119,18	95,611285
CCR	95,611285	95,611285	245,80

Como se puede observar en los datos obtenidos el tiempo computacional crece enormemente por cada *fold* añadido, mientras que el CCR apenas se ve variado, por lo que debemos de preguntarnos si merece la pena ese aumento en el tiempo, por la mínima variación de CCR y algunas veces incluso sin ninguna variación para obtener ese mejor modelo con los parámetros óptimos. Bueno y como podemos comprobar en este caso, no se ha variado de forma alguna el CCR ni los parámetros óptimos.

Base de datos *Spam*

15. Debes entrenar un modelo lineal de SVM con valores $C = 10^{-2}$, $C = 10^{-1}$, $C = 10^{0}$ y $C = 10^{1}$. Para ello utiliza un *script* similar al que usaste en la sección 2.6. Compara los resultados y establece la mejor configuración.

Para este ejercicio haremos uso de los ficheros *train_spam.libsvm* y *test_spam.libsvm*. Para ello hemos realizado distintas ejecuciones usando un gamma de 200 y las distintas Cs que ofrece esta pregunta, he aquí los resultados obtenidos, usando una configuración *'linear'* para la SVM:

C	10^{-2}	10^{-1}	10^{0}	10^{1}
T.Computacional (s)	8,90	**6,79**	5,63	5,09
CCR	98,00%	**98,90%**	97,80%	97,50%

El tiempo lo he recogido como un dato más, puesto que no va a influir en la hora de escoger los parámetros para el mejor modelo. Puesto que se puede apreciar como el tiempo computacional es muy parecido para las distintas ejecuciones.
Observando los datos obtenidos se ve como todas las ejecuciones han obtenido muy buenos resultados, aunque la mejor configuración ha sido usando **$C = 10^{-1}$,** obteniendo un tope de 98,9% de CCR, lo cual es muy alto.
Algo que cabe destacar es cómo se puede observar el principio de un sobre-entrenamiento a partir de **$C = 10^{0}$,** esto se puede ver al ver como va disminuyendo el CCR en test, mientras que si hubiéramos impreso también el CCR de entrenamiento este iría aumentando o incluso ya sería del 100% probablemente.

16. Para la mejor configuración, construye la matriz de confusión y establece cuáles son los correos en los que la SVM se equivoca. Consulta las variables de entrada para los correos que no se clasifican correctamente. Ten en cuenta que para cada patrón, cuando x_i es igual a 1 quiere decir que la palabra *i*-ésima del vocabulario aparece, al menos una vez, en el correo.

La mejor configuración obtenida en el ejercicio anterior ha sido usando **Gamma = 200 y $C = 10^{-1}$**, así que ha sido para esta entrada de parámetros para la que hemos sacado la matriz de confusión para poder observar los correos en los que nuestro modelo se ha equivocado, para posteriormente mostrar las entradas de dichos correos haciendo uso del fichero *vocab.txt*, y así podremos ver las palabras que contenían dichos correos.
La matriz de confusión es la siguiente:

684	**8**
3	**305**

Como podemos ver en la matriz de confusión, tenemos 11 correos mal clasificados, los cuales han sido los siguientes:
10, 22, 59, 74, 148, 329, 408, 527, 561, 843 y 882.
A continuación pondremos las palabras que aparecen en dichos correos, las cuales han sido las variables de entrada para nuestro modelo y en las que se ha equivocado.
Dichas palabras se pondrán una única vez por aparición y son las siguientes:

abil, about, abov, accept, access, account, action, activ, actual, ad, adam, add, address, administr, advertis, after, again, ago, aid, alb, all, allow, almost, along, also, altern, alwai, am, american, amount, an, and, ani, anoth, anti, anyon, anyth, anywai, anywher, appar, appli, applic, approach, ar, area, aren, around, as, ascii, ask, at, attornei, audio, aug, awai, awar, babi, back, bad, bank, base, basenumb, basic, bb, be, beat, beberg, becaus, becom, been, befor, begin, behavior, believ, below, best, better, between, big, bill, bit, blog, blood, bodi, boi, boot, both, boundari, box, brand, bug, bui, built, burn, busi, but, by, ca, california, call, can, card, care, carri, case, cat, catch, caus, cd, cell, cent, center, certain, chang, charg, charset, check, chief, choic, choos, chri, citi, class, clean, clear, clearli, click, close, clue, code, collect, colleg, com, come, comment, commun, comparison, compil, complet, comput, concern, confid, congress, constitut, consult, contain, content, contract, contribut, control, convers, copi, copyright, corpor, cost, could, couldn, count, countri, cours, court, creativ, credit, critic, current, custom, cv, dai, daili, dark, date, dave, de, deal, dear, decid, declin, default, defin, definit, descript, deserv, detail, detect, did, differ, director, displai, do, doe, doesn, dollar, dollarnumb, domain, don, done, doubl, doubt, dream, drive, each, earli, easi, eat, economi, ed, effect, effort, either, els, email, emailaddr, encod, end, engin, enough, entertain, entir, error, especi, establish, etc, even, ever, everi, everyon, everyth, exactli, exampl, excel, exercis, exist, exmh, experi, expir, express, extend, fair, fall, far, father, favorit, fax, feder, feel, few, figur, file, find, first, five, fix, flow, follow, for, forc, forev, format, former, fortun, found, free, freedom, from, full, further, futur, garrigu, geek, gener, get, girl, give, given, gnu, gnupg, go, gold, good, googl, got, govern, great, guarante, gui, ha, hack, had, handl, happen, hard, hate, have, haven, he, headlin, hear, heard, heart, hello, help, her, here, hi, high, him, hold, home, honor, hour, hous, how, html, httpaddr, hundr, idea, if, imagin, import, improv, in, inc, includ, independ, indic, industri, inform, instal, instead, instruct, interest, interfac, intern, internet, interview, into, investig, is, isn, it, jame, job, joe, judg, jul, just, keep, kind, knew, know, lack, land, larg, last, late, law, le, left, legal, lender, less, let, life, like, limit, line, link, linux, list, listen, littl, live, ll, lo, loan, local, lock, log, long, longer, look, lose, loss, lot, love, low, lowest, made, mai, mail, mailer, main, maintain, major, make, male, man, manag, mani, mark, market, mass, match, matter, mayb, me, mean, medic, men, messag, method, mh, microsoft, might, mike, million, mime, mind, minut, model, mon, mondai, monei, month, more, morn, mortgag, most, mostli, mother, motiv, move, ms, much, multi, multipart, must, my, myself, name, nbsp, necessari, need, net, never, new, nextpart, no, normal, not, note, noth, now, number, numberp, oblig, octob, of, offer, often, oh, old, on, onc, onli, open, oper, opportun, option, or, order, org, origin, osdn, other, otherwis, our, out, outsid, over, own, owner, packag, page, pai, paid, pain, paper, part, partner, peopl, per, perfect, perl, person, pgp, phone, piec, place, plain, platform, pleas, plu, pm, pocket, point, post, potenti, prefer, press, pretti, price, principl, problem, produc, product, profit, program, programm, progress, propos, provid, public, pudg, purpos, put, qualiti, quickli, quot, radio, rate, ratio, razor, re,

reach, read, readi, realli, reason, receiv, record, refin, relat, relationship, releas, rememb, remov, repli, report, repres, request, requir, reserv, respect, respons, rest, return, review, revok, right, risk, rock, role, run, sai, said, sale, same, school, second, see, seem, select, self, sell, send, senior, sep, septemb, sequenc, servic, setup, sever, sexual, sf, she, ship, should, show, side, sign, signatur, similar, simpli, sinc, site, situat, six, size, skill, slow, smoke, so, softwar, sold, solut, some, someth, sometim, song, soon, sort, sound, sourc, spam, special, specif, speech, speed, sponsor, spot, stai, stand, standard, start, state, step, still, stock, stop, stream, strong, stuff, stupid, subject, subscrib, subscript, such, summer, support, sure, surpris, suspect, system, take, taken, talk, tax, team, tech, tell, terrorist, text, than, thank, that, the, thei, their, them, themselv, then, there, these, thi, thing, think, those, thought, thu, thursdai, time, tire, to, todai, togeth, too, tool, top, total, touch, trade, transfer, treat, tri, troubl, true, trust, tue, tuesdai, two, type, unlimit, unseen, up, updat, us, user, ve, veri, version, via, virtual, visa, vnumber, voic, vs, wa, wai, wait, want, washington, we, websit, wed, week, weight, well, went, were, what, whatev, when, where, which, while, who, whole, why, will, win, window, wish, with, within, woman, women, won, wonder, word, work, worker, world, would, write, wrote, www, year, yesterdai, yet, york, you, your, zip.

17. Compara los resultados obtenidos con los resultados utilizando una red RBF. Para ello, haz uso del programa desarrollado en la práctica anterior. Utiliza solo una semilla.

Para este ejercicio vamos a comparar los resultados usando el script desarrollado en la práctica anterior que hacía uso de una red RBF, pero necesitaremos el número de neuronas, que para ello haremos diversas pruebas como ya haciamos en la práctica anterior, usando el número de patrones de entrenamiento y aplicando un porcentaje a estos. Pero primero deberemos de pasar los ficheros a *csv* que era lo que esta práctica era capaz de leer, para ello he usado el programa Weka, que podremos cargar los ficheros y pasarlos a lo que deseemos, una vez realizado esto seremos capaces de realizar este ejercicio.
Sabiendo que el fichero de entrenamiento cuenta con 4000 correos, con 5%, 10%, 25% y 50%. Y con una eta de 10^{-5}.
Estos han sido los resultados, teniendo en cuenta que el CCR conseguido en esta práctica fue de **98,9%**:

RBF	200	400	1000	2000
CCR test	99,3%	99,4%	99,7%	99,9%
CCR train	100%	100%	100%	100%

Además el tiempo computacional para cada prueba ha sido bastante alto, superando en exceso al tiempo computacional usado por el svm lineal. Pero los resultados obtenidos son mejores que los obtenidos en esta práctica, lo cual era difícil de superar puesto que teniamos unos resultados casi perfectos, así que debemos preguntarnos si compensa dicho

aumento en el tiempo computacional y dicho aumento de dificultad del modelo para mejorar ese CCR sobre un 1%, además teniendo cuidado con el número de neuronas, puesto que resulta bastante fácil el sobreentrenamiento en este tipo de modelos.

Cabe destacar que el resultado obtenido con 2000 neuronas ha tardado por lo menos 15 minutos en ejecutarse, pero ha obtenido un resultado casi perfecto, en mi opinión ha merecido la pena la espera para ese tiempo computacional para verlo como ejemplo, pero no se podría usar un modelo tan complejo en la práctica debido al tiempo invertido.

Bibliografía:

- Presentación de la práctica.
- Documentación de la práctica.
- Documentación python Sklearn.
- Documentación python