Universidad de Córdoba
Escuela Politécnica Superior de Córdoba

Ingeniería Informática
Especialidad: Computación
Cuarto curso. Primer cuatrimestre

Introducción a los modelos computacionales.

Práctica 4: Maquinas de vectores soporte.

José Manuel Cuevas Muñoz
31013019A
i62cumuj@uco.es

Curso académico 2019-2020
Córdoba, 27 de noviembre de 2019

Índice

Índice de figuras

Índice de tablas

1. Trabajando con el primer dataset y un kernel lineal

Lo primero que se hace a la hora de ejecutar el programa es importar las librerías necesarias, en este caso numpy para las operaciones matriciales, pyplot para los gráficos, sklearn para tener acceso al SVM y pandas para poder leer CSV. Tras esto, carga el fichero en CSV, separa los datos en entradas y salidas y crea un objeto de la clase SVC con el que se entrenará.

```
svm_model = svm.SVC(kernel='linear',C=200)
```

Como podemos ver tanto aquí como en la figura 1, el kernel utilizado es lineal ya que intenta separar las clases con un hiperplano separador lineal; y el valor de penalización C es igual a 200. El resto del código sirve para crear y mostrar una gráfica como la que se ve en la figura 1, donde se pueden ver tanto los vectores soporte como el hiperplano separador.

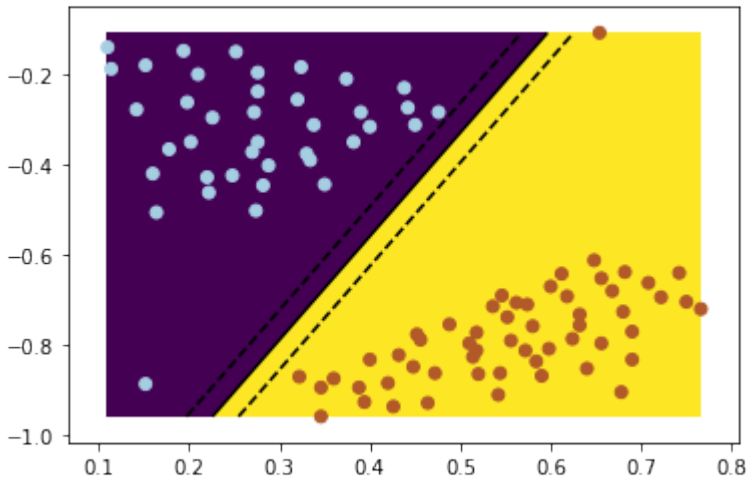

Figura 1: SVM con kernel lineal y base de datos dataset1

En caso de que en la gráfica quitáramos la parte relativa a mostrar el hiperplano separado, quedarían solo la nube de puntos que se muestra en la figura 2. En esta se puede como instintivamente, la manera más simple de separar estas dos clases sería con una recta.

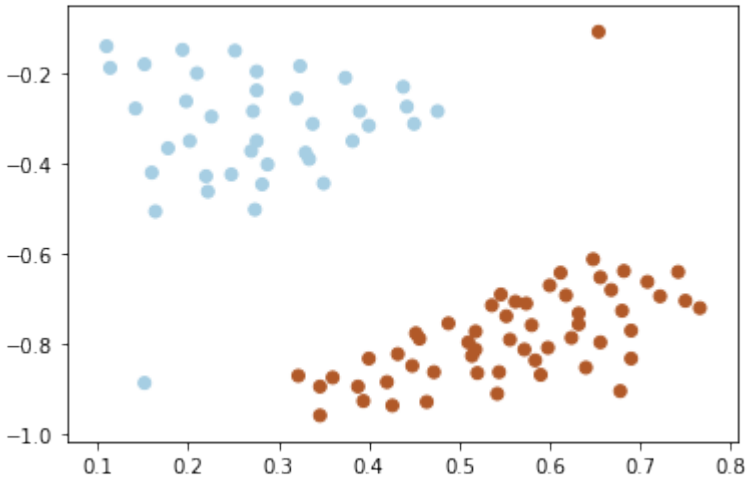

Figura 2: Nube de puntos relativa al dataset1

Por último, debemos hablar del parametro C, el cual es un parámetro de penalización, osea, cuanto se penalizarán los errores cometidos por un clasificador. Vamos a ir modificando este parámetro y viendo como cambia el resultado.

C	CCR
10e-2	83.1578947 %
10e-1	97.894736 %
10e0	97.894736 %
10e1	100 %
10e2	100 %
10e3	100 %
10e4	100 %

Tabla 1: CCR de la base de datos dataset1 modificando C

Aquí se puede ver como a partir de C=10 (Cuando se empiezan a penalizar más los errores) el CCR es del 100 %. En la figura 3 parece que en principio, a partir de 10e2 no varía. Sin embargo, con valores a partir de 10e2 el margen es el mínimo.

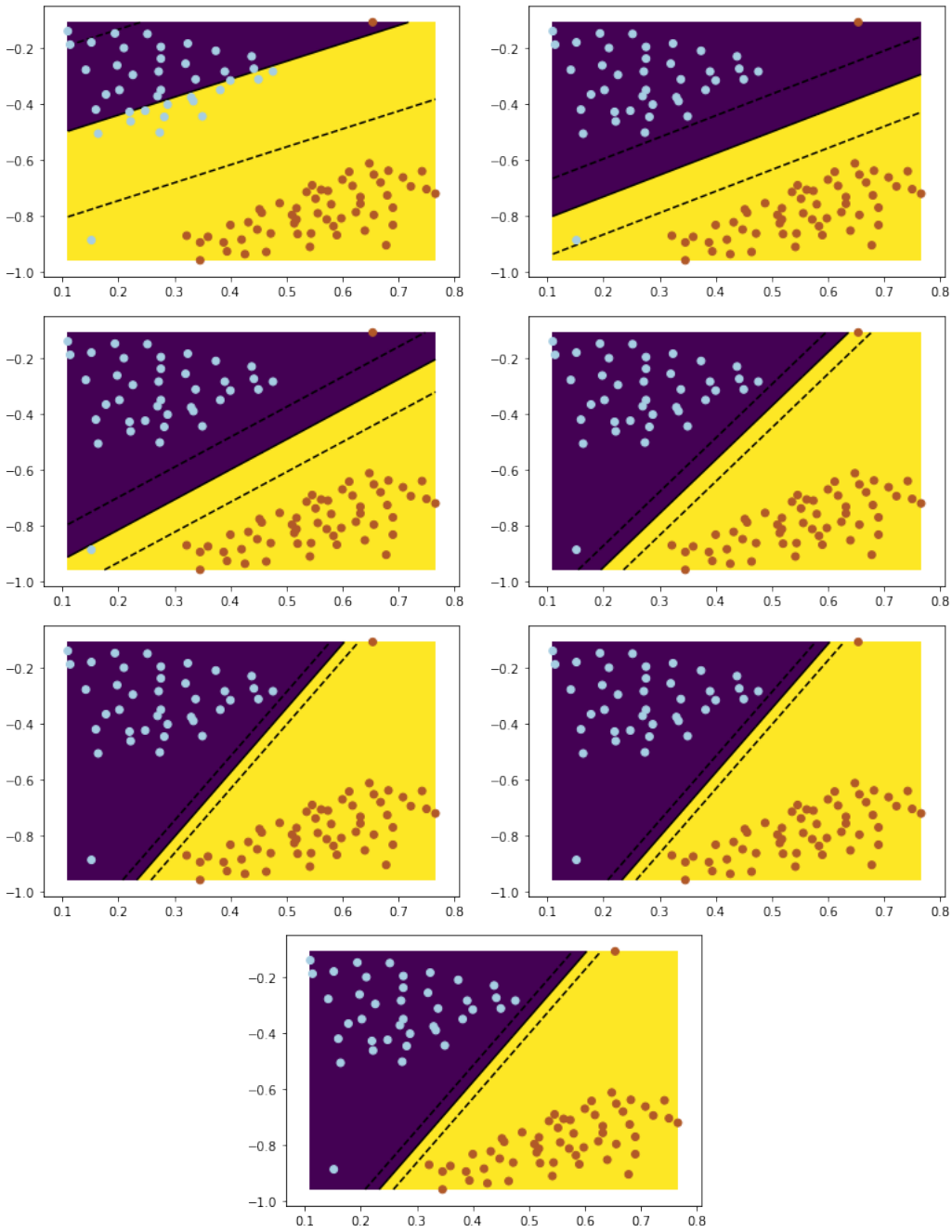

Figura 3: Resultado del dataset1 con mientras aumenta C

2. Trabajando con el segundo dataset y el kernel 'rbf'

¿Que pasa si probamos los datos del ejercicio anterior? Pues, dá igual como se modifique el C dentro de los datos del anterior ejercicio, el resultado siempre será el que se muestra en la figura 4, con un CCR del 66.20046 % ya que al no encontrar una mejor solución lineal (la base de datos no es linealmente separable) incluye todos los patrones en la clase mayoritaria. La única C con la que me ha salido algo distinto ha sido C=10e10, donde me sale lo que se muestra en la figura 5.

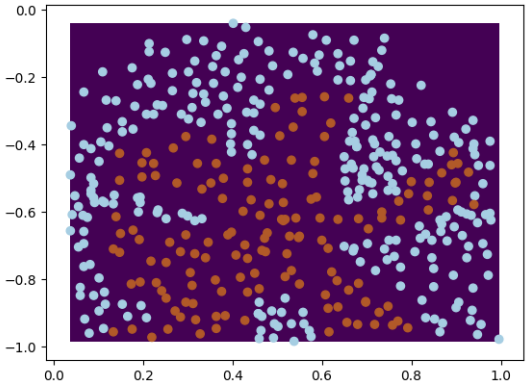

Figura 4: Intento de clasificación lineal con dataset2

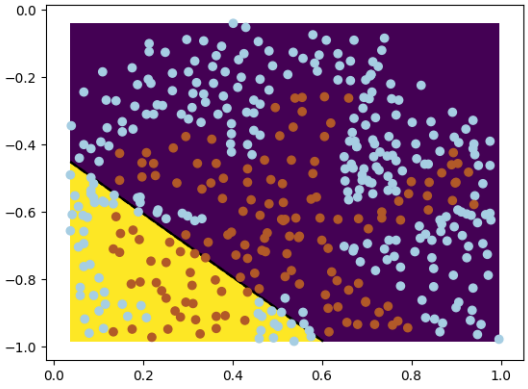

Figura 5: Intento de clasificación lineal con dataset2 y C muy alta

Por eso, necesitamos otro tipo de kernel. En este caso vamos a utilizar el kernel 'rbf'. Además del C, el kernel RBF necesita un parámetro γ. Mientras más alto sea el parámetro, más se ajustará este a la base de datos. Por eso debemos buscar el valor idóneo de gamma entre los posibles.

```
svm_model = svm.SVC(kernel='rbf',C=10e3, gamma=5)
```

El resultado de esta configuración se muestra en la figura 6 mientras que en la figura 7 se muestra el sobreentrenamiento e infraentrenamiento en la misma base de datos.

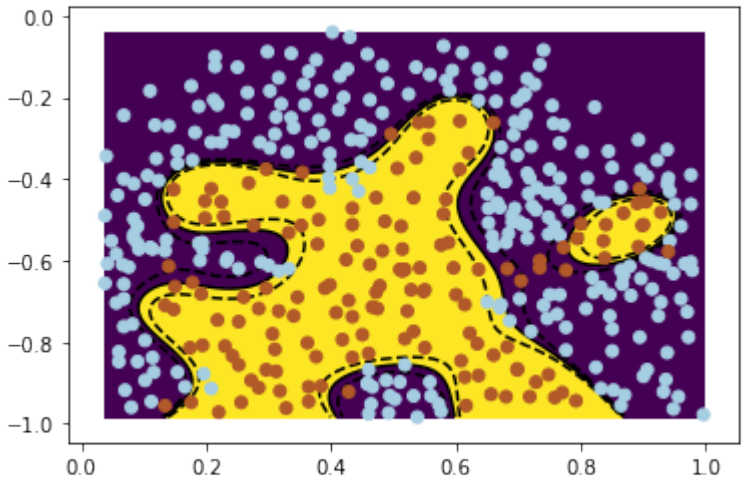

Figura 6: SVM con kernel RBF y dataset2

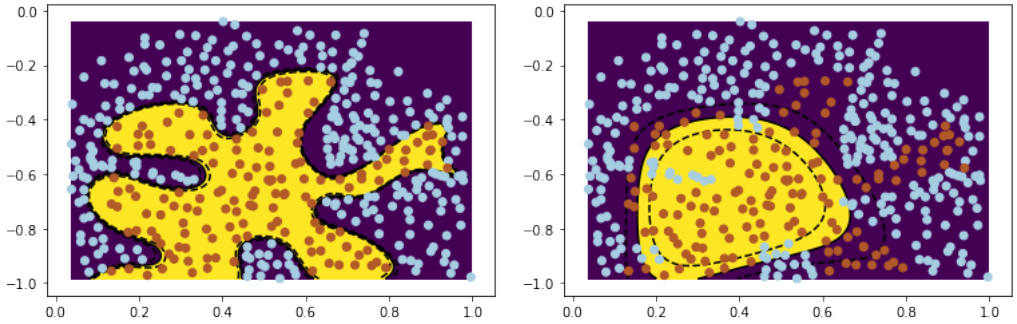

Figura 7: Sobreentrenamiente e infraentrenamiento en SVM con kernel RBF

3. Trabajando con el tercer dataset

Lo primero que nos podemos preguntar del tercer dataset es ¿Es linealmente separable? Como se puede comprobar en la figura 8, no es linealmente separable, además de que algunos de los puntos salen de la distribución de densidad de las clases (los puntos que están dentro de las otras clases) por lo que se puede pensar en un principio que serían outliers.

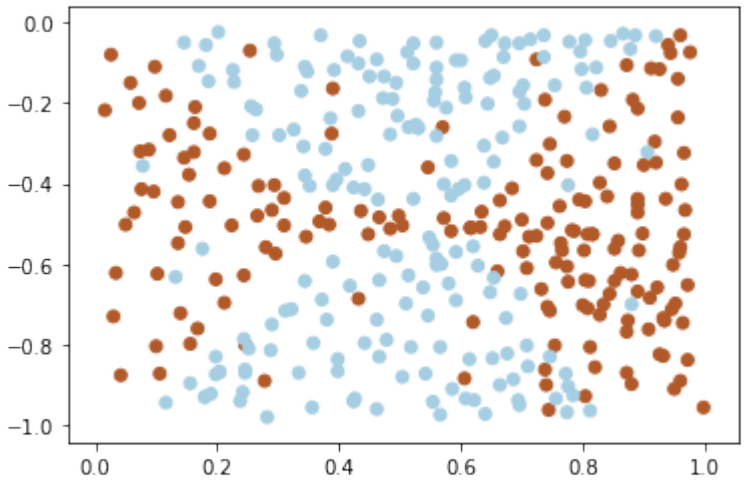

Figura 8: SVM con kernel RBF y dataset2

Entonces tendremos que volver a probar los valores de c y de gamma. Para ello probaremos con las combinaciones de c=$\{0.02,0.2,2,200\}$ y γ=$\{0.02,0.2,2,200\}$. El CCR obtenido se muestra en la tabla 2, mientras que la distribucion obtenida se muestra en la figura 9. Si solo mirasemos la tabla nos quedaríamos con C=200 y γ=200 por ser la que más ccr de entrenamiento tiene pero si miramos las figuras, la que mejor capta la arquitectura de las clases sin dejarse llevar por outlayers es c=2 y γ=200, por lo que me quedaría con esta arquitectura.

C	gamma	CCR
0.02	0.02	51.35135
	0.2	51.35135
	2	51.35135
	200	51.35135
0.2	0.02	51.35135
	0.2	53.24324
	2	77.56756
	200	91.89189

C	gamma	CCR
2	0.02	55.40540
	0.2	71.89189
	2	87.02702
	200	93.51351
200	0.02	71.08108
	0.2	87.56756
	2	89.18918
	200	99.72972

Tabla 2: CCR del dataset 3 variando C y gamma

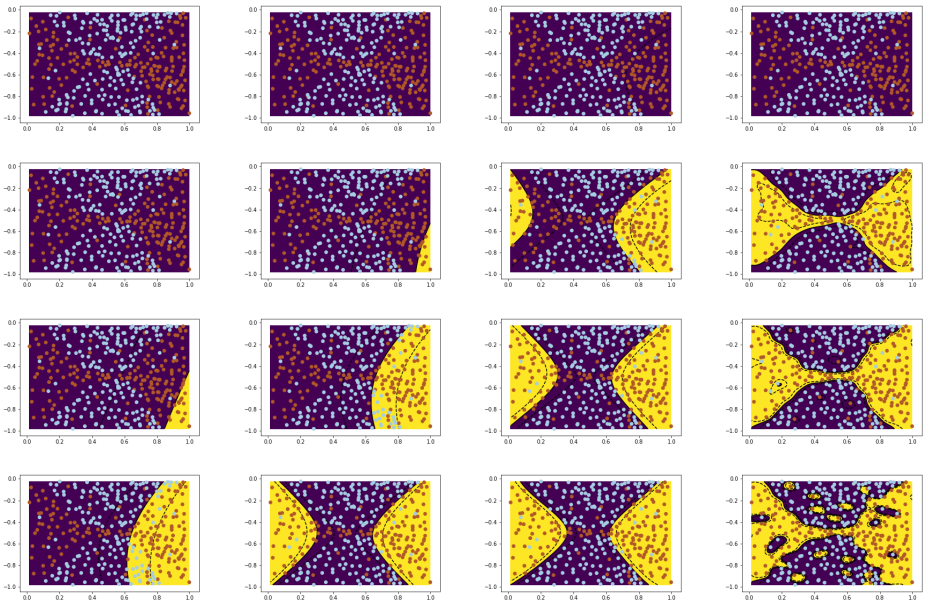

Figura 9: Salidas de la base de datos 3 variando c(filas) y gamma(columnas)

Una de las cosas que podemos probar es a normalizar los datos de entrenamiento y ver el que nos sale. Para ello vamos a utilizar C=2 y γ=200 y la función StandardScaler de la librería sklearn y a dividir la base de datos en un conjunto de entrenamiento y otro de test con train_test_split.

```
scaler = preprocessing.StandardScaler()
X = scaler.fit_transform(X,y)

#Partimos el conjunto en train y test
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y,
    stratify=y,test_size=0.25,random_state=1)
```

Tras normalizarla y poner como parámetros los mejores vistos anteriormente me da un CCR de 69.892473 % usando la semilla 21 para partir train y test y 75.2688172 % utilizando la semilla 1. Podemos comprobar que la semilla varía el resultado mínimamente y que el resultado es bastante peor que si no normalizáramos.

Uno de los problemas es el encontrar los parámetros perfectos. Para ello podemos utilizar cross-validation. Sklearn nos ofrece esta oportunidad gracias a la función gridSearch. En nuestro caso vamos a utilizar parámetros que se encuentran dentro del espacio logarítmico indicado en el código y vamos a probar con 5 particiones, 9 valores para γ y 11 para C.

```
Cs = np.logspace(-5, 15, num=11, base=2)
Gs = np.logspace(-15, 8, num=9, base=2)
optimo = GridSearchCV(estimator=svm_model, param_grid=
    dict(C=Cs,gamma=Gs),n_jobs=-1,cv=5)
optimo.fit(X_train,y_train)
print(optimo.score(X_test,y_test))
```

El valor óptimo obtenido ha sido de 84.9462365 %. Este método nos beneficia dado a que los parámetros C y γ dependerán siempre de la base de datos y variará dependiendo del conjunto de entrenamiento utilizado, por lo que queremos encontrar los mejores valores para los datos en general, no para un conjunto de test cogido en un momento dado. ¿Como funcionaría esto por dentro?

El siguiente codigo muestra mi version del GridSearchCV:

```
#Conseguimos los distintos C y gamma
Cs = np.logspace(-5, 15, num=11, base=2)
Gs = np.logspace(-15, 8, num=9, base=2)

#Hacemos k particiones, siendo k=5
sss = StratifiedShuffleSplit(n_splits=5, test_size=0.2,
    random_state=1)
"""Inicializamos el vector donde vamos a guardar el CCR
    de los distintos modelos
Las dos primeras columnas de cada fila seran el C y el
    gamma de esa fila"""
```

```
pruebas=np.zeros((len(Cs)*len(Gs),7))
i=2
for train_index, test_index in sss.split(X, y):
  #Separamos en conjunto de entrenamiento y de test
  X_train, X_test = X[train_index], X[test_index]
  y_train, y_test = y[train_index], y[test_index]
  j=0
  for c in Cs:
    for gam in Gs:
      #Para cada C y cada gamma entrenamos
      aux = svm.SVC(kernel='rbf',C=c,gamma=gam).fit(
         X_train,y_train)
      if i==2:
        #Llenamos las dos primeras columnas con c y gamma
        pruebas[j,0]=c
        pruebas[j,1]=gam
      #guardamos el CCR
      pruebas[j,i]=aux.score(X_test,y_test)
      j=j+1
  i=i+1
maximoj=0
maxmedia=np.mean(pruebas[0,2:7])
for x in range(1,(len(Cs)*len(Gs))):
  #Vamos buscando la media mayor entre los distintos
     conjuntos
  med=np.mean(pruebas[x,2:7])
  if med>maxmedia:
    maxmedia=med
    maximoj=x
#Entrenamos con el optimo
optimo=svm.SVC(kernel='rbf',C=pruebas[maximoj,0],gamma=
   pruebas[maximoj,1])

#Partimos el conjunto en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y,
   stratify=y, test_size=0.25,random_state=1)
optimo.fit(X_train,y_train)
print(optimo.score(X_test,y_test))
```

En los comentarios del codigo se explica lo que se va haciendo en cada paso. Con esta versión he conseguido un CCR del 90.322580645 %, lo cual es un incremento con respecto al grid search de sklearn.

4. Base de datos nomnist

Base de datos formada por 900 patrones de train y 300 de test. La base de datos incluye imágenes de letras de la A a la F identificada por sus valores en escala de grises.

Como hemos visto, sacar los mejores valores de C y γ no es un problema trivial, por ello volveremos a utilizar gridSearch con los mismos valores que utilicé en el dataset3. Tras realizar el entrenamiento, la mejor salida que nos da es de 88.666667 % de CCR.

Hasta ahora hemos trabajado siempre con valores de k de 5. Sin embargo, computacionalmente hablando, esto requiere un tiempo muy alto, ya que para cada uno de las 5 bases de datos, tiene que entrenar 11 valores de C por 9 valores de γ, lo cual da un total de 495 entrenamientos. Una forma de reducir el tiempo total de realizar el gridSearch es modificar el número de particiones (K) que se realizarán. Por eso probaremos con tres valores de K y veremos tanto su CCR como el tiempo que tarda en entrenar.

K	CCR	Tiempo
3	90.33333 %	139.9218838 segundos
5	88.66666 %	286.6194005 segundos
10	92 %	655.0491595 segundos

Tabla 3: Tiempos y resultados de la base de datos nomnist con cross-validation variando la K

Como se muestra en la tabla 4, a mayor K, el tiempo aumenta de forma considerable. La mejor solución que hemos conseguido ha sido con K=10, sin embargo con K=3 el resultado ha sido parecido y el tiempo computacional menos de un cuarto que con K=10. Si comparamos con otras practicas, en la RBF consiguió un ccr de 91.27 % en test y lo ha realizó en 14.288677 segundos y en la red neuronal de la practica 2 con un CCR de test del 84 % y una duración de 213.199 segundos.

5. Base de datos spam

Uno de los campos donde se utiliza el aprendizaje automático con un mayor éxito es la detección automática de spam en servidores de correo. Para ello, vamos a preprocesar la base de datos con los siguientes valores:

- Se escribirán todas las palabras en letras minúsculas.
- Utilizar texto plano, eliminando HTML.
- Todas las URL serán cambiadas por httpaddr.
- Todas los correos serán cambiadas por emailaddr.
- Todos los números serán cambiados por number.
- Utilizar las raíces de las palabras en vez de la palabra entera.
- Quitar todo lo que no sean palabras.

El resultado será que cada atributo representa una palabra y en cada instancia, aparece 1 o 0 dependiendo si la palabra aparece o no aparece.

Lo primero que haremos será entrenar esta base de datos con un kernel lineal y variando la C, para comprobar cual de los valores es el mejor. El resultado se muestra en la tabla 5,donde podemos comprobar que el mejor C es 10e-2 (el que menos penalización) y que, dado los buenos resultados, podemos concluir que es linealmente separable.

C	CCR
10e-2	98.9 %
10e-1	97.8 %
10e0	97.8 %
10e2	97.5 %

Tabla 4: Resultado de entrenar spam con distinto C.

Mientras que en la figura 10 se muestra la matriz de confusión en la base de datos spam con un kernel lineal y una C de 10e-2. Se puede ver como la mayoría de errores son de spam falsamente clasificados como no spam, aunque sea la clase minoritaria.

$$\begin{pmatrix} 684 & 8 \\ 3 & 305 \end{pmatrix}$$

Figura 10: Matriz de confusión en la base de datos spam

Estos son algunas de las palabras que más se repiten el los correos mal clasificados como spam. Como podemos ver, algunas de estas palabras están

mal escritas adrede, palabras como invest o expens que son muy comunes en los correos de spam; además de que en todos aparece una dirección de correo. Si nos fijamos en la cantidad de palabras, a excepción del patrón 9, todos los patrones tienen muchas palabras, como suelen acostumbrar los correos de spam.

Apariciones	Palabra
2	ani
2	ar
2	awai
2	befor
2	caus
2	com
2	current
2	direct
2	due
3	emailaddr
2	expens
2	govern
2	invest
2	know
2	list
2	over
2	pleas
2	respect
2	self
2	through
2	www

Tabla 5: Palabras que más aparecen en correos que mal clasificados como spam

Mientras que estas son algunas de las palabras mal clasificadas de mensajes que eran spam. La mayoría de esos correos eran bastante cortos a diferencia que la media de los correos de spam o son largos pero con palabras mucho más técnicas y con mejor ortografía y por eso, en parte, se saltan los filtros antispam.

Apariciones	Palabra
6	advertis
5	dai
6	emailaddr
5	ha
5	httpaddr
7	mail
8	number

Tabla 6: Palabras que más aparecen en correos que mal clasificados como no spam

Ahora, podemos probar a comprobar el resultado que nos da el entrenar una red RBF y compararlo con el resultado que nos ha dado anteriormente. El resultado de entrenar una red RBF para el problema es de 99.44 % +- 0.12 % de CCR de entrenamiento y 98.16 % +- 0.19 % de CCR de test, mientras que el resultado de entrenar un modelo SVM es del 99.825 % de CCR de entrenamiento y el 98.9 % de test. Como podemos apreciar, el resultado es ciertamente parecido, dándonos una salida ligeramente mejor en el SVM. En cuanto a tiempo de computo, también es similar en ambas ejecuciones, siendo el SVM ligeramente más rápido. Por último, si tuviera que escoger uno de los dos modelos, escogería el SVM por ser este un modelo lineal y más sencillo.

Por último, podemos comprobar el resultado que nos daría al entrenar SVM con kernel RBF en la base de datos spam. Para ello utilizaremos como parámetro C el mejor en las anteriores ejecuciones y como parámetro gamma utilizaremos el parámetro heurístico 'scale', que se representa como $1/(num_variables * varianza)$. El resultado es de un 94 % de CCR en test, menor al del SVM con kernel lineal. En la figura 11, se muestra la matriz de confusión. Como podemos comprobar, el error de tipo 2, osea, ser marcado como spam cuando no lo es, es mayor que con el entrenamiento lineal. Se puede deber a que en este caso, la C necesite ser más grande o si no, se irán más patrones a la clase mayoritaria, dado a que si no, se centrará mucho en los patrones de la clase mayoritaria. De hecho, con C=1, el CCR será del 98.9 %.

$$\begin{pmatrix} 690 & 2 \\ 58 & 250 \end{pmatrix}$$

Figura 11: Matriz de confusión en la base de datos spam con RBF