Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Aprendizaje automático

Practica 5: Clasificación Bayesiana

Autora:
Nerea Gallego Sánchez (801950)

7 de diciembre de 2022

Índice

1. Introducción y objetivos

Esta práctica consiste en aplicar clasificación Bayesiana regularizada en un problema real de clasificación multi-clase: el reconocimiento de dígitos manuscritos.
Se utiliza como datos de entrada una versión reducida del conjunto de datos MNIST.
Cada muestra es una imagen de 20x20 píxeles. Como atributos para la clasificación se utilizan directamente los niveles de intensidad de los 400 píxeles.

Se van a utilizar técnicas de predicción con modelos Gaussianos regularizados. Para ello, se van a utilizar tanto la clasificación con Bayes ingenuo, como la clasificación con las covarianzas completas.
Por último se calcula la matriz de confusión y tanto el valor de precisión como el de recall para cada una de las clases.

2. Estudio previo

```
function modelo = entrenarGaussianas(Xtr, ytr, nc, NaiveBayes, lauda)
    for i = 1:nc
        modelo(i).N = sum(ytr == i)
        Xp = Xtr(ytr==i)
        modelo(i).mu = mean(Xp, 1)
        % resta = Xp - modelo(i).mu
        modelo(i).Sigma = cov(Xp, Xp')
        modelo(i).Sigma = modelo(i).Sigma + lauda*eye(size(Xp,2))
        if NaiveBayes == 1
            modelo(i).Sigma = diag(modelo(i).Sigma)
        end
    end
end

function yhat = clasificacionBayesiana(modelo, X)
    y = []
    for i=1:nc
        y = [y gausslog(modelo(i).mu, modelo(i).Sigma, X)]
    end
    [t, yhat] = max(y, [], 2)
end
```

Figura 1: Estudio previo a la práctica.

3. Entrenamiento y clasificación con modelos Gaussianos regularizados

En este apartado se pide programar las funciones entrenarGaussianas para que aprenda el modelo Gaussiano de cada clase, tanto el el caso general como en el caso de Bayes ingenuo. Además, se pide programar la función clasificacionBayesiana para que haga la clasificación de muestras utilizando los modelos Gaussianos entrenados.

Para crear la función entrenarGaussianas se ha utilizado un código muy similar al realizado en el apartado previo. Se ha creado la estructura de datos, tal y como se indica en los ficheros que nos proporcionan los profesores de la asignatura. Este struct contiene 3 campos. En el campo N se almacenan cuántas muestras hay de esa clase, en el campo mu, se almacena la media de esa clase, y por último, en el campo $Sigma$, se almacena la covarianza de la misma.
A continuación, para cada clase, se calcula la cantidad de muestras que hay, su media y su varianza, y se almacena en la componente correspondiente del vector de structs creado anteriormente.
La función contiene un campo llamado $NaiveBayes$ que indica si se quieren calcular los pesos del modelo utilizando Bayes Ingenuo o no. Para ello, se ha establecido que para realizar clasificación con Bayes Ingenuo esa variable tendrá valor 1. En ese caso, la matriz de covarianza se calculara como la matriz diagonal de las covarianzas.
La función *entrenarGaussianas* es la siguiente:

```
% Dados los atributos de entrada, la salida esperada, la cantidad de clases
% que se quieren predecir, si se quiere usar Bayes ingenuo o no y el
% parametro de regularizacion que se quiere usar.
% Devuelve el modelo para realizar el entrenamiento.
function modelo = entrenarGaussianas( Xtr, ytr, nc, NaiveBayes, landa )
% Entrena una Gaussiana para cada clase y devuelve:
% modelo{i}.N     : Numero de muestras de la clase i
% modelo{i}.mu    : Media de la clase i
% modelo{i}.Sigma : Covarianza de la clase i
% Si NaiveBayes = 1, las matrices de Covarianza seran diagonales
% Se regularizaran las covarianzas mediante: Sigma = Sigma + landa*eye(D)
modelo = struct('N',{},'mu',{},'Sigma',{});
    for i = 1:nc
        % Cantidad de muestras que hay de esa clase
        modelo(i).N = sum(ytr == i);
        % Guarda las muestras de la clase i en Xp
        Xp = Xtr(ytr == i,:);
        % Media para la clase i
        modelo(i).mu = mean(Xp);
        % Matriz de covarianza
        modelo(i).Sigma = cov(Xp);
        modelo(i).Sigma = modelo(i).Sigma + landa * eye(size(Xp,2));
        % Si se quiere usar Bayes ingenuo se queda con la matriz diagonal de
        % covarianzas.
        if NaiveBayes == 1
            modelo(i).Sigma = diag(diag(modelo(i).Sigma));
        end
    end
end
```

Para la clasificación Bayesiana, se ha construido la función clasificacionBayesiana tal y como se indica en el enunciado. Para ello, se ha calculado para cada clase, su predicción. Esto se ha realizado con ayuda de la función *gaussLog* proporcionada por los profesores. Dicha función devuelve en un vector columna la verosimilitud que se calcula dados los datos de entrada, la media para esa clase y la matriz de covarianza. Sin embargo, se quiere obtener la probabilidad a posteriori ($p(y_i|x) = \frac{p(x|y_i)*p(y_i)}{p(x)}$). Para obtener dicha probabilidad se ha multiplicado la verosimilitud, por la probabilidad a priori ($p(y_j) = \frac{N_j}{N}$) de manera que se obtiene en cada columna de y la probabilidad a posteriori para una clase determinada. Se ha decidido realizar el cálculo de esta manera porque el denominador para cada clase es el mismo. Por último, se quiere devolver cual es la clase que se obtiene de la clasificación. Para ello, se devuelve la clase que ha obtenido una mayor probabilidad.

La función $clasificacionBayesiana$ es la siguiente:

```
% Dado el modelo y los datos de entrada, devuelve la clase predicha para
```

```
% cada muestra.
function yhat = clasificacionBayesiana(modelo, X)
% Con los modelos entrenados, predice la clase para cada muestra X
    y = [];
    for i=1:10
        % Calcula la verosimilitud de las muestras para una clase.
        y = [y gaussLog(modelo(i).mu, modelo(i).Sigma,X)];
        % Probabilidad a posteriori
        y(:,i) = y(:,i).*(modelo(i).N/size(X,1));
    end
    % Clase predicha
    [t,yhat] = max(y,[],2);
end
```

4. Bayes ingenuo

En este apartado se pide, basándome en el código de la práctica anterior programar el entrenamiento y clasificación multiclase, usando Bayes ingenuo. Para ello, es necesario separar un 20 % de los datos para validación y encontrar el mejor parámetro de regularización.

Para empezar, se han cargado los datos de entrenamiento y test en las variables X, X_{test}, y, e y_{test}. Esto se realiza mediante la instrucción:

```
load('MNISTdata2.mat');
```

Proporcionada por los profesores de la asignatura en la práctica anterior.

A continuación, es necesario separar el 20 % de los datos para validación. Para ello, se ha utilizado la función separar construida en la práctica anterior.

```
% Dado el conjunto de datos completo, el porcentaje de datos que se quieren
% usar como datos de entrenamiento, las columnas de los atributos y las
% columnas de las salidas esperadas, devuelve el conjunto de datos de
% entrenamiento con sus salidas esperadas y el conjunto de datos de
% validaci n con sus salidas esperadas.
function [Xtr, Ytr, Xtst, Ytst] = separar(data, porc, col_x, col_y)
    % Realiza una permutacion de los datos para que se escojan de manera
    % aleatoria.
    randOrd = randperm(size(data,1));
    perm = data( randOrd,:);
    % Establece el limite entre los datos de entrenamiento y los de
    % validacion.
    lim = int32(size(data,1)*porc);
    Xtr = perm((1:lim),col_x);
    Ytr = perm(1:lim,col_y);
    Xtst = perm(lim+1:end,col_x);
    Ytst = perm(lim+1:end, col_y);
end
```

Esta función, dado el conjunto de datos inicial, el porcentaje de datos que se quiere usar para entrenamiento, las columnas en las que se encuentran los atributos de entrada y la columna en la que se encuentra la salida, devuelve un conjunto de datos para realizar el entrenamiento, correspondiendo este al porcentaje indicado por parámetro con respecto del total y el conjunto de datos que se utilizan para validación. Ambos conjuntos de datos se devuelven con la salida esperada.

Se ha creado un vector para los posibles valores de regularización. Para ello, se ha utilizado la función *logspace*, al igual que en las prácticas anteriores. Se ha decidido utilizar 30 valores entre 10^{-6} y 10.

Para evaluar cuál de los modelos es mejor, se quiere obtener el error que produce un modelo concreto. Se ha creado la función errorRegularizaciónMulticlase de manera que dado en conjunto de datos, la salida esperada, y el modelo, devuelve la tasa de error que produce.

```
% Dada la entrada, la salida esperada y el modelo, devuelve la
% media de casos erroneos que se obtienen.
function err = errorRegularizacionMulticlase(X,y,t)
    % Calcula la prediccion obtenida con la matriz de pesos
    ypred = clasificacionBayesiana(t,X);
    % Se suma la cantidad de filas que son distintas de las que hay en el
```

```
    % vector de salidas esperadas, y se calcula la media de error.
    err = sum(y ~= ypred) / size(y,1);
end
```

Para calcular el error, hace uso de la función construida en el apartado anterior clasificacionBayesiana.

Para calcular el mejor parámetro de regularización, se ha utilizado un bucle, que para cada posible valor de regularización, construye el modelo utilizando Bayes Ingenuo y calcula tanto el error que produce con los datos de entrenamiento con los que se ha construido el modelo, como el error que produce con los datos de validación, que no han sido usados para construir el modelo.
En este caso, igual que en la práctica anterior se ha utilizado un único pliegue, haciendo uso de un 80 % de los datos de entrenamiento para calcular el modelo y el 20 % restante para realizar la validación del mismo.
Se guardan en dos vectores, los errores obtenidos para poder mostrarlos en una gráfica. Por último se comprueba si el error con los datos de validación para el modelo actual, es el menor obtenido hasta el momento, quedándose con el modelo que produce menor error de validación, ya que este será el mejor modelo.

```
bestModel = 0;
bestErrV = inf;
errT = [];
errV = [];
naiveBayes = 1;
for model = 1:size(lambda,1)
    err_T = 0;
    err_V = 0;
    % se calcula la matriz de pesos
    th = entrenarGaussianas(Xtr, Ytr, 10, naiveBayes, lambda(model));
    % se calculan los valores de error
    err_T = errorRegularizacionMulticlase(Xtr,Ytr,th);
    err_V = errorRegularizacionMulticlase(Xv,Yv,th);
    % se guardan los valores de error obtenidos para posteriormente
    % imprimir una gr fica
    errT = [errT err_T];
    errV = [errV err_V];
    % si el modelo es mejor que el anterior, se guarda
    if err_V < bestErrV
        bestModel = model;
        bestErrV = err_V;
    end
end
```

De esta manera, se obtiene en bestModel, el índice del mejor parámetro de regularización de entre los posibles valores que se encuentran en el vector de lambda. También se obtienen en los vectores $errT$ y $errV$ los errores de entrenamiento y validación respectivamente que se obtienen con cada modelo utilizado.

El mejor parámetro de regularización obtenido es $\lambda = 7{,}278954e-03$.
Como se puede observar en la figura 2 el parámetro de regularización con el que se obtiene menor error para los datos de validación es el indicado anteriormente. En la gráfica también se muestra como, utilizar un parámetro de regularización más pequeño produce sobreajuste, ya que el error obtenido con los datos de entrenamiento es muy pequeño, pero el error obtenido con los datos de validación es bastante mayor. Esto se debe a que la penalización que realiza a modelos complejos es pequeña. Sin embargo, si se aumenta mucho el valor del parámetro λ se obtiene subajuste, ya que penaliza los modelos complejos y lo simplifica a uno más sencillo. Los errores que se obtienen tanto con los datos de entrenamiento como de validación se equiparan pero son muy grandes.

Por último, se pide re-entrenar con todos los datos de entrenamiento para el mejor valor de lambda, y utilizar los datos de test para calcular la matriz de confusión y los valores de precisión y recall para cada dígito.

Para calcular la matriz de confusión, se ha utilizado la función matrizConfusion creada en la práctica anterior.

```
% Dada la salida predicha, la salida esperada y la cantidad de clases,
% devuelve la matriz de confusion de manera que contiene en las filas la
% clase real y en las columnas, la clase predicha.
```

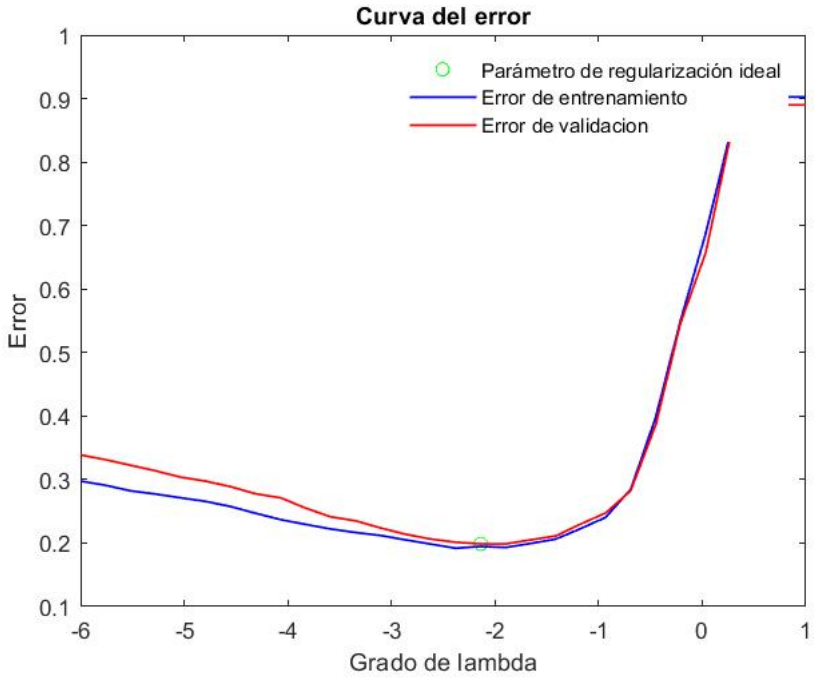

Figura 2: Curva de evolución del error en función del parámetro de regularización lambda.

```
function matrizConf = matrizConfusion(ypred,ytest,clases)
    matrizConf = zeros(clases,clases);
    for i = 1: clases
        for j = 1:clases
            matrizConf(i,j) = sum(ytest==i & ypred == j);
        end
    end
end
```

Dada la salida predicha, la salida real y la cantidad de clases, calcula la matriz de confusión. Contiene en las filas la clase real y en las columnas la clase predicha.

Para calcular la matriz de confusión, ha sido necesario re-entrenar con todos los datos de entrenamiento. Para ello, primero se ha añadido una columna de unos, como se realiza para cada modelo, a los datos de entrenamiento y a los datos de test. A continuación, se ha creado el modelo de con los datos de entrenamiento. Para ello se ha utilizado la función entrenarGaussianas pasándole esta vez, el mejor parámetro de λ el que se ha obtenido como mejor anteriormente ($\lambda = 7{,}278954e-03$).
Se ha calculado, a continuación, las salidas predichas con los datos de test. Para ello, se ha utilizado la función clasificacionBayesiana pasándole el modelo que se acaba de construir y los datos de test como entrada. Finalmente, se llama a la función matrizConfusion y se obtienen los siguientes resultados que se encuentran en la tabla 1.

Como se puede observar en la matriz de confusión el número más problemático es el 5, seguido del 3 y el 4.
De la misma manera, los que más se confunden con otros son el 9 y el 1.

También se pide calcular la precisión y el recall para cada una de las clases.
Se define como precisión a la fracción de positivos que hay realmente de todos los que fueron clasificados como positivos.
Se calcula de la siguiente manera: $Precisión = \frac{TP}{(TP+FP)}$.
Calculando TP y FP con respecto a la clase que se observa en ese instante.
Se define como recall a la cantidad de positivos que se clasificaron como tal, del total de positivos que había.
Se calcula de la siguiente manera: $Recall = \frac{TP}{(TP+FN)}$.

		Clase predicha									
		1	2	3	4	5	6	7	8	9	0
Clase real	1	98	1	1	0	0	0	0	0	0	0
	2	5	76	0	0	0	9	0	8	0	2
	3	5	6	71	1	2	2	1	7	5	0
	4	1	4	0	71	1	3	0	3	17	0
	5	3	1	11	4	58	6	1	7	5	4
	6	2	1	0	0	0	95	0	2	0	0
	7	4	0	0	4	0	0	83	1	8	0
	8	7	2	3	3	3	1	0	74	7	0
	9	2	0	1	4	0	0	2	2	89	0
	0	0	0	0	0	3	4	0	4	0	89
	TOTAL	127	91	87	87	67	120	87	108	131	95

Cuadro 1: Matriz de confusión.

	Precision	Recall
1	0.77178	0.98
2	0.83526	0.76
3	0.8161	0.71
4	0.8161	0.71
5	0.8657	0.58
6	0.7917	0.95
7	0.9540	0.83
8	0.6852	0.74
9	0.6794	0.89
0	0.9368	0.89

Cuadro 2: Tabla de precisión/recall.

Para obtener tanto la precisión como el recall, se han calculado las medidas TP (true positive), FP (false positive), TN (true negative) y FN (false negative) para cada una de las clases. Por último, se ha calculado las medidas de precisión y recall en dos vectores.

```
TP = [];
TN = [];
FP = [];
FN = [];
for i = 1:10
    TP = [TP sum(ytest == i & ypred == i)];
    TN = [TN sum(ytest ~= i & ypred ~= i)];
    FP = [FP sum(ypred == i & ytest ~= i)];
    FN = [FN sum(ypred ~= i & ytest == i)];
end
Precision = TP ./ (TP + FP)
Recall = TP ./ (TP + FN)
```

En la tabla precision/recall se muestran los resultados obtenidos tanto de precisión como de recall para cada una de las clases.

Como se puede observar, las clases con menor precisión son el 9, seguido del 8 y el 1. Esto es debido a que en numerosas ocasiones se predicen que las entradas son de esta clase, cuando en realidad no lo son. De la misma manera, las clases con peor recall son el 5, 3 y 4. Como ya hemos podido observar anteriormente en la matriz de confusión, son las clases que peor predicción tienen y por consecuente, los número más problemáticos.

Por último, se pide visualizar las confusiones con la función $verConfusiones$.

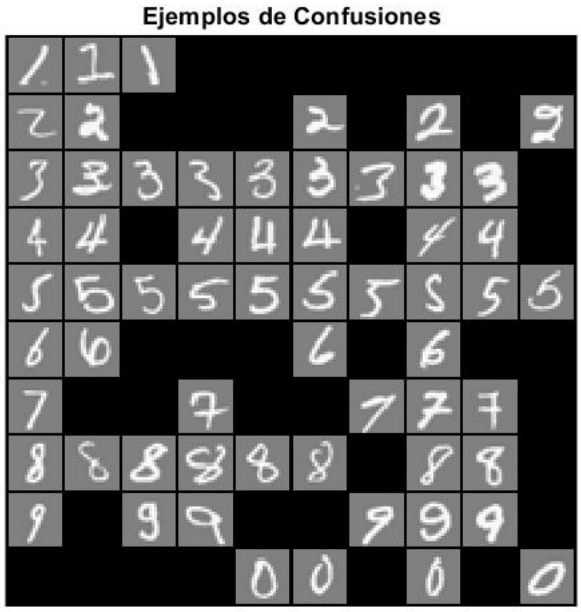

Figura 3: Ejemplos de confusión con los dígitos para la clasificación con Bayes ingenuo.

Como se puede observar en la imagen 3 el modelo confunde algunos dígitos con otros debido a su mala grafía. Se muestran más ejemplos de aquellos números que tienen una peor predicción como pueden ser el 5 o el 3.

5. Covarianzas completas

En este caso, se pide repetir los pasos realizados en el apartado anterior, pero esta vez con covarianzas completas. Es decir, sin usar Bayes ingenuo.

Primero, es necesario buscar el mejor parámetro de regularización. Para ello, se ha utilizado la función 4 pero esta vez, con el parámetro $naiveBayes = 0$.

El mejor parámetro de regularización obtenido es $\lambda = 3{,}856620e-02$.
Como se puede observar en la figura 4 el parámetro con el que se obtiene menor error para los datos de validación es el indicado anteriormente. En la gráfica se muestra como, utilizar un parámetro de regularización más pequeño, igual que ocurría con el modelo de Bayes ingenuo, produce sobreajuste, ya que el error obtenido con los datos de entrenamiento es muy pequeño, pero el error obtenido con los datos de validación es bastante mayor. De la misma manera, si se aumenta mucho el valor del parámetro λ se obtiene subajuste. Los errores que se obtienen tanto con los datos de entrenamiento como de validación se equiparan, pero son muy grandes.

A continuación, se pide re-entrenar con todos los datos de entrenamiento para el mejor valor de lambda, y utilizar los datos de test para calcular la matriz de confusión y los valores de precisión y recall para cada dígito.

Se ha creado un nuevo modelo con los datos de entrenamiento al completo. Para ello, se ha utilizado la función entrenarGaussinas pasándole el mejor parámetro de λ obtenido anteriormente ($\lambda = 3{,}856620e-02$).
Se ha calculado a continuación las salidas predichas con los datos de test. Para ello, se ha utilizado la función clasificacionBayesiana pasándole el modelo que se acaba de construir y los datos de test como entrada. Finalmente, se invoca a la función matrizConfusion pasándole como parámetros las salidas predichas, las salidas reales y el número de clases.

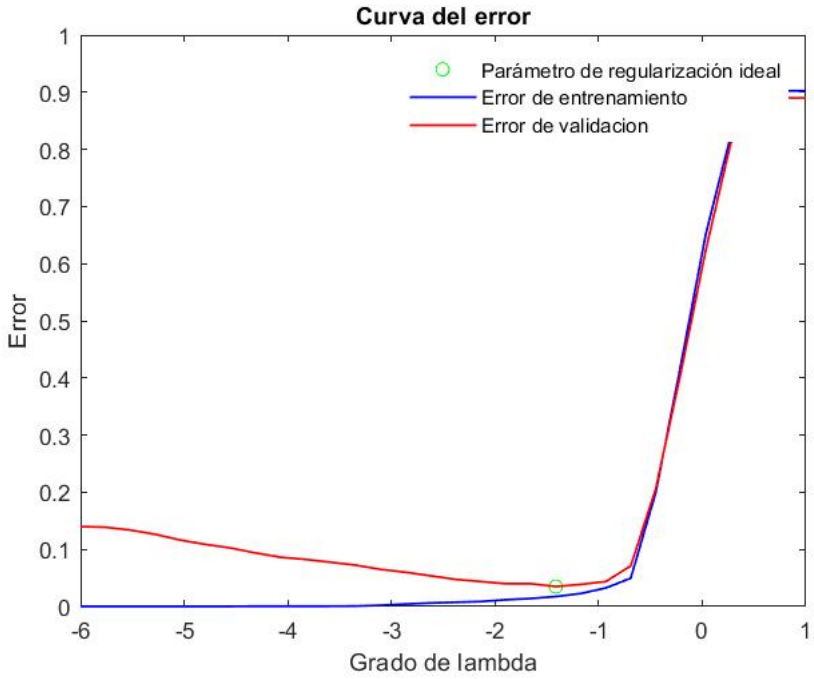

Figura 4: Curva de evolución del error en función del parámetro de regularización lambda.

Se obtienen los siguientes resultados:

		Clase predicha									
		1	2	3	4	5	6	7	8	9	0
Clase real	1	99	1	0	0	0	0	0	0	0	0
	2	1	96	1	0	0	0	1	1	0	0
	3	1	2	94	0	1	0	0	1	1	0
	4	1	1	0	97	0	0	0	0	0	1
	5	0	0	1	0	94	1	0	4	0	0
	6	0	0	0	0	0	99	0	0	0	1
	7	2	0	0	0	0	0	96	0	2	0
	8	2	2	3	0	1	0	0	91	1	0
	9	1	0	2	0	0	0	1	2	94	0
	0	0	0	0	0	0	0	0	0	0	100
	TOTAL	107	102	101	97	96	100	98	99	98	102

Cuadro 3: Matriz de confusión con Bayes completo.

Como se puede observar en la matriz de confusion el número más problemático es el 8, seguido del 3, 5 y 9. De la misma manera, lo números que más se confunden con otros son el 1, el 2 y el 0.

También se pide calcular la precisión y el recall para cada una de las clases. Se ha utilizado el código escrito en la sección anterior para calcular estas medidas.

En la tabla precision/recall se muestran los resultados obtenidos tanto de precisión como de recall para cada una de las clases.

Como se puede observar, las clases con menor precisión son el 8, seguido del 1 y el 3. Esto es debido a que en numerosas ocasiones se predicen que las entradas son de esta clase, cuando en realidad no lo son. De la misma manera, las clases con peor recall son el 8, 3, 5 y 9. Como ya hemos podido observar

	Precision	Recall
1	0.9252	0.99
2	0.94126	0.96
3	0.9307	0.94
4	1	0.97
5	0.9792	0.94
6	0.99	0.99
7	0.9796	0.96
8	0.9192	0.91
9	0.9592	0.94
0	0.9804	1

Cuadro 4: Tabla de precisión/recall con Bayes completo.

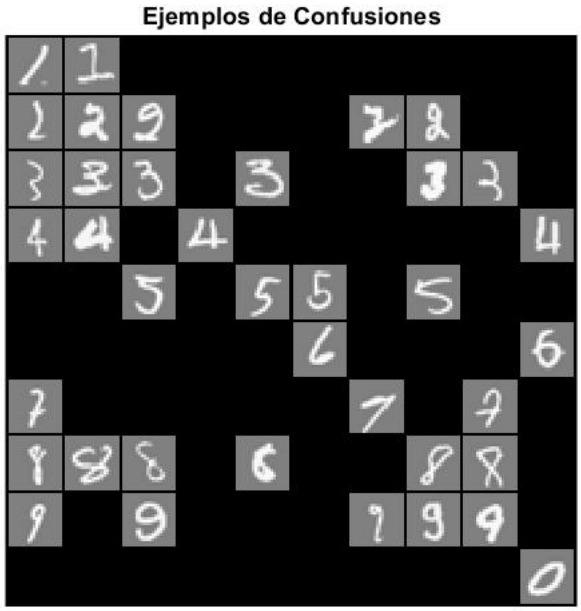

Figura 5: Ejemplos de confusión con los dígitos para la clasificación con Bayes completo.

anteriormente en la matriz de confusión, son las clases que peor predicción tienen y por consecuente, los número más problemáticos.

Por último, se pide visualizar las confusiones con la función $verConfusiones$.

Como se puede ver en la imagen 5 el modelo confunde algunos dígitos con otros debido a su mala grafía. Se muestran más ejemplos de aquellos números que tienen una peor predicción como pueden ser el 8 o el 3.

6. Comparación entre Bayes ingenuo y Bayes completo

Se pide compara cuál de los dos modelos funciona mejor para este conjunto de datos en concreto.

Si se comparan las matrices de confusión: ubicadas en la tabla 1 y en la tabla 3. Se puede observar como el modelo de Bayes completo produce muchas menos confusiones que el modelo de Bayes ingenuo. Bayes completo produce un mayor número de aciertos para todas las cifras. A pesar de que el modelo es mejor, la cifra 1, en ambos casos se confunde con el resto en numerosos casos.

Si se comparan las cifras de precisión y recall: ubicadas en la tabla 2 y en la tabla 4. Se puede observar como todas las cifras obtenidas en la segunda tabla son notablemente mejores que las cifras obtenidas en la primera.

Estos resultados se deben a que el modelo de Bayes ingenuo supone la independencia de los atributos. Es decir, supone que no existe correlación entre la intensidad que tiene un píxel y la intensidad que tienen los píxeles adyacentes. Sin embargo, si que la hay, ya que al ser un dibujo, los píxeles cercanos a uno dado, tendrán una intensidad similar al primero. Por consecuente, el modelo de Bayes completo realiza una predicción mucho mejor, ya que no supone esta independencia entre atributos.

7. Comparación entre regresión logística y Bayes completo.

Los resultados obtenidos con regresión logística fueron:

		Clase predicha									
		1	2	3	4	5	6	7	8	9	0
Clase real	1	97	1	0	0	0	0	0	2	0	0
	2	3	83	3	2	1	2	2	3	0	1
	3	1	5	85	0	5	1	1	2	0	0
	4	1	3	0	88	0	2	0	2	4	0
	5	1	0	6	2	83	2	0	6	0	0
	6	0	0	0	0	0	99	0	0	1	0
	7	2	3	0	2	0	0	91	0	2	0
	8	4	1	3	2	3	2	0	82	3	0
	9	1	0	3	3	1	0	4	1	86	1
	0	0	0	0	1	2	3	0	0	0	94
	TOTAL	110	96	100	100	95	111	98	98	96	96

Cuadro 5: Matriz de confusión con regresión logística.

	Precision	Recall
1	0.8818	0.97
2	0.8646	0.83
3	0.85	0.85
4	0.88	0.88
5	0.8737	0.83
6	0.8919	0.99
7	0.9286	0.91
8	0.8367	0.82
9	0.8958	0.86
0	0.9792	0.94

Cuadro 6: Tabla de precisión/recall con regresión logística.

Si se comparan las matrices de confusión ubicadas en la tabla 3 y en la tabla 5, se puede observar como el modelo de Bayes completo produce menos confusiones que el modelo de regresión logística. Sin embargo, si lo comparamos con el modelo de Bayes ingenuo ubicado en la tabla 1, el modelo de regresión logística produce mejores resultados ya que no produce tantas confusiones.

Para comparar precisión y recall, se ha creado la tabla 7 donde se compara la precisión media y el recall medio por cada uno de los modelos. Como se puede observar, el modelo de regresión logística,

produce mejores resultados que el modelo de Bayes ingenuo, pero peores resultados que el modelo de Bayes completo.
Por lo tanto, el mejor modelo calculado hasta el momento para este conjunto de datos es Bayes completo obteniendo una precisión de media de 96,05 % y un recall medio de 96 %.

	Precision media	Recall medio
Regresión logística	0.88823	0.888
Bayes Ingenuo	0.8152	0.804
Bayes completo	0.9605	0.96

Cuadro 7: Tabla de precisión/recall medio.