Practica 3

Aprendizaje Automático

José Manuel Pérez Lendínez

Contents

1 Digits Data Set

1.1 Comprender el problema a resolver.

Los datos almacenados en el dataset pertenecen a un conjunto de dígitos manuscritos. Para el reconocimiento de los dígitos se utiliza una matriz de 32x32 inicialmente. Esta matriz se dividirá en matrices mas pequeñas de 4x4 sin solaparse. A cada una de estas matrices le daremos un valor numérico en el rango [0,16].

Por tanto finalmente obtendremos por cada dígito manuscrito una matriz 8x8 que contendrán un valor numérico entre [0,16]. Esto es un total de 64 enteros por cada dígito analizado. La división de la matriz inicial en matrices mas pequeñas nos asegura que el tamaño del problema se mucho menor al reducir el numero de valores por cada muestra y que al tener en cuenta matrices de 4x4 para dar un valor evitamos muchas invarianzas a pequeñas distorsiones que se podrían dar si usáramos directamente la matriz de 32x32.

Las clases para los dígitos vienen dada por los números naturales de un único dígito ([0-9]), teniendo un total de 10 posibles clases.

La base de datos seleccionada ya nos da la partición de train y test con el siguiente numero de instancias y participantes.

Tipo	Nº Participantes	Nº instancias
Train	30	3823
Text	13	1797

La proporción de cada clase es muy parecida tanto en el train como en el text con una diferencia como máximo en el conjunto de entrenamiento de 13 instancias entre la clase que mas tiene y la que menos. En el caso del test la diferencia máxima es de 9 instancias.

Clase	Nº instancias train	Nº instancias test
0	376	178
1	389	182
2	380	177
3	389	183
4	387	181
5	376	182
6	377	181
7	387	179
8	380	174
9	382	180

1.2 Preprocesado de datos

En este caso el he realizado una normalización muy sencilla, al saber que los datos están en el rango [0sklearn.preprocessing.PolynomialFeatures
-16], solo tenemos que dividir los datos entre 16.

También se ha realizado una eliminación de variables que no aportan al problema. Estas variables son aquellas en cuya columna los datos tengan una varianza menor a 0 en este caso(todas las variables que solo tienen un único valor en todas las muestras.). Para esto he utilizado la función de VarianceThreshold de sklearn.feature_selection. Esto nos ahorra 2 variables que tienen una varianza de 0. Estas partes de la matriz nunca han sido utilizadas para ningún numero.

Para esto utilizo la siguiente función:

```
def eliminarDatosVarianza(train_x,train_y,limite):
        row_train = np.size(train_X,0)

        datos = np.concatenate((train_X, test_X), axis=0)

        selector = VarianceThreshold(limite)
        datos_procesados = selector.fit_transform(datos)

        train = datos_procesados[:row_train, :]
        test = datos_procesados[row_train:, :]
        return train,test
```

A continuación vamos a añadir nueva información que nos ayude a clasificar, para esto uso la librería sklearn.preprocessing.PolynomialFeatures con el parámetro de grado 2. El código de la función seria el siguiente.

```
def anadirInformacionPolinomial(train_X, test_X, grado = 2):
        poly = PolynomialFeatures(grado)
        train_X = poly.fit_transform(train_X)
        test_X = poly.fit_transform(test_X)

        return train_X, test_X
```

Esto nos añadirá nueva información de la siguiente manera. Si partimos de una muestra con [a,b] nos daría $[1, a, b, a^2, ab, b^2]$. En este caso he optado por elegir 2 porque con 4 la memoria se desborda al añadir demasiada información y con 3 no me mejoro el problema.

1.3 Selección de clase de funciones a usar.

El problema al que nos enfrentamos es un problema de clasificación. Vamos a usar una función lineal para solucionar este problema. Al ser clasificación y tener mas de dos etiquetas este problema no puede ser solucionado con una función lineal unicamente. Para esto utilizaremos la función lineal enfrentando una clase a todas las demás(one vs rest).

Vamos a poner un pequeño ejemplo con tres clases para ver como funcionaria. Como se ve en la siguiente imagen es imposible dividir las clases con una única linea.

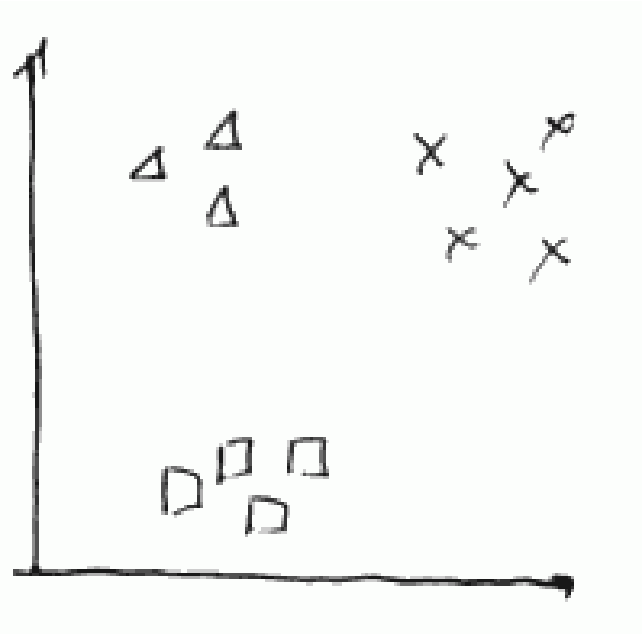
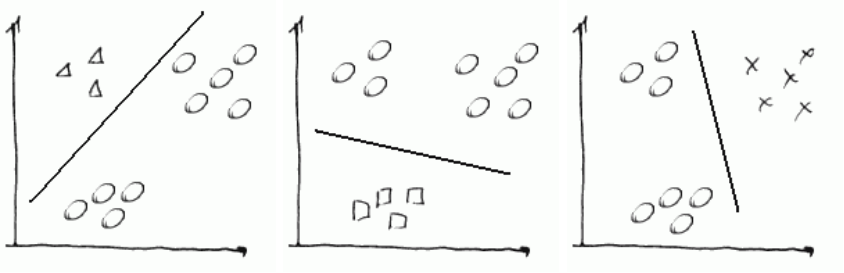


Figure 1: Conjunto de clases

En las siguientes imágenes se ve como enfrentamos una clase a las otras dos de forma que si se puede conseguir una separación mediante una función lineal.



(a) Triangulos (b) Cuadrados (c) Cruces

Figure 2: División mediante one vs rest

Por tanto nos quedaremos con las funciones lineales para este problema.

1.4 Definición de training, validación y test.

En este caso los datos vienen divididos ya en train y test. En el train se tienen 40 personas y el test 13. Si somos capaces de conseguir un buen E_{out}, estaremos consiguiendo que sea capaz de reconocer los números sin tener en cuenta la

forma de escribir de cada participante. Esto me parece la mejor opción y no he realizado ninguna modificación en este apartado.

1.5 Necesidad de regularización

Como se explico en teoría la regularización nunca viene mal para eliminar un poco de sobreajuste de nuestro modelo. En este caso he utilizado la regularización l1 (usa el valor absoluto) puesto después de varias pruebas me ha dado mejores resultados que l2(usa pesos al cuadrado). En este caso la diferencia es muy pequeña llegando a alrededor de un 1%. Pero al ser también mas eficiente y rápida la regularización l1 he terminado optando por esta.

1.6 Definición del modelos a usar y sus parámetros

El modelo a usar sera regresión logística multiclase , utilizando one vs rest(ovr). Los parámetros que tendremos que tener en cuenta sera el parámetro que nos indica el tipo de regularización (l1 o l2) y el parámetro c que indica como de fuerte sera la regularización.