Práctica 3
Aprendizaje Automático

Ignacio Aguilera Martos

20 de Mayo de 2019

Índice

1. Problema optdigits: clasificación

1.1. Problema a resolver

El problema que debemos resolver consta de un conjunto de datos llamado Optical Recognition of Handwritten digits. Contiene en total 5620 instancias con 64 variables más su correspondiente clase. Las clases son 10 (del 0 al 9) indicando el número con el que se identifica la instancia.

Si leemos la descripción del conjunto de datos vemos que todos los datos son numéricos y que no tenemos ningún valor perdido en ninguna instancia. Esto será útil de cara a realizar preprocesamiento de los datos.

Además se provee al final del fichero de descripción del conjunto de datos cómo acierta un modelo K-NN utilizando k desde 1 a 11 donde se puede observar que el porcentaje de acierto es de más del 97 %. Esta información es muy útil, pues si pensamos en cómo funciona el algoritmo K-NN podemos deducir sin pintar ni representar información del conjunto de datos que los mismos están aglomerados de forma clara en clusters. Esto será también relevante a la hora de probar ciertos algoritmos como perceptrón, pues podemos saber más o menos la estructura del conjunto de datos e intuir que va a funcionar correctamente si la separación de los clusters entre sí es suficiente.

Además el número de instancias totales de cada clase está más o menos balanceado, es decir tenemos más o menos el mismo número de instancias de cada uno de los dígitos y por tanto no tenemos ninguno descompensado con respecto al resto.

Por tanto tras este primer análisis del conjunto de datos el problema que tenemos que resolver es, dado este conjunto de datos, ser capaces de proveer un modelo que ajuste lo mejor posible la clasificación de las instancias y obtenga el mejor score posible en el conjunto de test.

1.2. Preprocesado de los datos

En primer lugar debemos recordar del apartado anterior que el conjunto de datos no tiene ningún valor perdido por lo que no corresponde hacer ningún tipo de preprocesamiento dirigidoa solventar este problema.

En segundo lugar disponemos de un conjunto con 64 atributos por lo que en un principio cabría descartar cualquier preprocesamiento que añada nuevas variables al conjunto de datos tales como expansiones polinómicas de ordenes superiores. Estas técnicas pretenden añadir más información al conjunto de datos pero no tenemos signos que nos indiquen que esto sería necesario por lo que en un principio no conviene emplear la técnica.

Lo que si he decido aplicar es tanto una normalización como un escalado o estandarización de los datos. En el caso de la normalización la operación es sencilla, es hacer que todos los vectores tengan norma 1 y en mi caso yo he escogido la norma con la que aplicar la operación la L2 o norma euclídea.

El segundo tipo de preprocesado es una estandarización de los datos a media cero y escalados mediante la varianza. Esto es realizar una transformación del tipo $z = \frac{x-\bar{x}}{\sigma}$ donde $\bar{x}$ es la media del conjunto, x es una instancia del mismo y σ su varianza. De esta forma al restar a todo el conjunto el valor de la media lo convertimos en un conjunto de media cero y además lo

escalamos según la varianza.

Los preprocesados que he explicado los he aplicado de tres formas, primero sólo una normalización, sólo una estandarización y una combinación de normalización y estandarización. De esta forma podremos comprobar qué resultados obtenemos con estas transformaciones previas.

A parte de este preprocesado he aplicado algoritmos de reducción de dimensionalidad para intentar reducir la complejidad en cuanto al número de variables de las instancias del conjunto de datos. Los algoritmos que he empleado son PCA, Incremental PCA, Kernel PCA y Factor Analysis.

PCA es un algoritmo que pretende descomponer el conjunto de variables en componentes ortogonales que expliquen la máxima cantidad posible de la varianza del conjunto. Esto en términos matemáticos nos da un sistema de generadores ortogonal de dimensionalidad el número de componentes que hemos elegido y que, por el orden en el que se toman hacen que expliquen una mayor cantidad de varianza. Si lo pensamos un momento podemos obtener un 100 % de explicación de la varianza si tomamos como número de componentes el mismo número de atributos. Es por esto que el criterio que he tomado es, primero hacer PCA sobre el conjunto de datos tomando el número de componentes igual al número de atributos y después obtener la varianza que explica cada componente. Utilizando este dato podemos ver (calculando la varianza acumulada) con qué número de componentes obtenemos un 95 % de explicación de la varianza. Este ha sido el criterio que he empleado para encontrar el número de componentes a tomar, no sólo en este algoritmo si no en todos los demás.

De igual forma Incremental PCA no es más que un análisis PCA realizado usando minibatches para que se consuma menos memoria y combinándolos después de forma adecuada.

Kernel PCA utiliza una técnica de proyección sobre espacios de menor dimensionalidad utilizando núcleos o kernels. Este comportamiento es muy parecido a los conceptos de filtros en imágenes, por ejemplo los filtros gaussianos o bluring que no es más que realizar una transformación punto a punto utilizando la información local, esto es de los puntos que se encuentran en el entorno. De esta forma se pueden conseguir proyecciones que continúen manteniendo de alguna forma la información y estructura del conjunto de datos y así reducir la dimensionalidad.

Por último Factor Analysis utiliza una descomposición similar a PCA pero posee ventajas como la facilidad de explicación de los datos a través de los factores obtenidos. Esto se puede resumir en que PCA intenta dividir el conjunto de datos en subvariables que no tienen por qué ser siquiera interpretables en el concepto general del problema (no tener interpretabilidad en el mundo real) mientras que FA intenta corregir esto otorgándole algo más de sentido. Esto viene de la capacidad de Factor Analysis de poder explicar la varianza en cualquier dirección del sistema ortogonal que calcula.

1.3. Selección de clases de funciones

En este caso se nos pide por parte del enunciado del problema que empleemos modelos lineales, esto es modelos que intenten ajustar una función que sea una transformación lineal de los atributos. Por tanto la clase de funciones viene prefijada y por tanto $\mathcal{H}$ viene dada por las fuciones de la forma $h(x) = w_0 + w_1x_1 + ... + w_dx_d \in \mathcal{H}$.

1.4. Conjuntos de training, test y validación

Para comprobar la efectividad de los modelos he utilizado una partición del conjunto original en test y train. En este caso he utilizado un 80 % del conjunto para entrenamiento y un 20 % para test. Esta partición la he tomado mediante la funcion train_test_split de sklearn estratificando los datos, esto es, manteniendo el porcentaje de clases en los conjuntos de test y train equilibrados con respecto al conjunto original.

1.5. Regularización, necesidad e implementación

Tras haber probado el funcionamiento de los modelos con conjuntos de train y test he comprobado que el ajuste del modelo es muy bueno tanto con el propio conjunto de train como con el conjunto de test, esto es que el overfitting del propio modelo no se produce pues aunque el ajuste es muy bueno en la propia muestra de entrenamiento esto no hace que el comportamiento con los datos de test (esto es como estimación del acierto fuera de la muestra) sea también muy satisfactorio.

Por tanto al no existir un sobreajuste que nos deje en una posición desventajosa no he visto necesario aplicar regularización.

1.6. Modelos usados y parámetros empleados

Lo primero que he realizado es un estudio de los posibles algoritmos a probar para determinar el comportamiento de los mismos y poder escoger a posteriori un algoritmo o unos algoritmos que merezca la pena pararse a configurar y probar.

La batería inicial de algoritmos a probar es:

- Mínimos cuadrados
- Ridge
- Lasso
- SGD
- Logistic Regression
- Passive-Aggresive
- Perceptron

Entre estos algoritmos tenemos tanto algoritmos de clasificación como algoritmos de regresión. La intención de probar algoritmos de regresión en este tipo de problemas es que un algoritmo de regresión podría aprender la función que asigna clases a cada instancia por lo que en un principio son válidos para aplicar aunque veremos que funcionan de forma muy deficiente.

Veamos para empezar los resultados con estos algoritmos sin ningún tipo de reducción de dimensionalidad ni preprocesamiento:

- Mínimos cuadrados: 0.5124410858401369
- Ridge: 0.5126173886330556
- Lasso: 0.5283626936997821
- SGD: 0.9317170818505338
- Logistic Regression: 0.953959074733096
- Passive-Aggresive: 0.9377224199288257
- Perceptron: 0.8876779359430605

Cabe decir que el algoritmo Mínimos cuadrados, Ridge y Lasso son algoritmos de regresión por lo que su unidad de medida es la métrica R^2 que mide la bondad del ajuste de forma que cuanto más cercano sea el valor a -1 o +1 mejor será dicho ajuste y cuanto más cerca de 0 peor. De esta forma podemos ver que el ajuste provisto es muy pobre y por tanto es normal descartar estos algoritmos frente al resto pues hemos conseguido unos resultados mucho mejores con el resto de algoritmos.

Cabe decir que el espacio parece bastante separable pues el algoritmo perceptrón ha sido capaz de separar y clasificar de forma satisfactoria el 88.77 % de los datos del conjunto de test con lo que, entre la información del propio conjunto de datos y la información que nos da el resultado del algoritmo perceptrón podemos imaginar que el conjunto de datos está conformado por clusters que además tienen una separación notable entre ellos.

Por tanto vamos a explicar el funcionamiento de los distintos modelos y posteriormente acabaremos eligiendo un solo algoritmo de los de clasificación que han funcionado bien.

- SGD: Gradiente Descendente Estocástico es el mismo algoritmo que hemos aprendido y discutido en clase con la única diferencia de que se tiene un comportamiento definido y compatible con escenarios de clasificación multietiqueta. Este hecho se logra gracias a que se toma el algoritmo definido para el caso binario y se enfrentan 2 a 2 todas las clases entre sí de forma que se puede dibujar un escenario de varias w que dividen el espacio para cada pareja de clases. Así se pueden tomar las regiones que dividen el espacio según cada una de las clases. Recordemos que nuestro escenario es de clusters muy agrupados entre sí. Este hecho es muy significativo pues hace que el algoritmo SGD sea muy ventajoso al tener regiones muy claras y definidas en el espacio y por tanto obteniendo un resultado muy bueno con este algoritmo.
- Logistic Regression: el algoritmo de regresión logística para clasificación sigue el mismo esquema que el visto en clase. Se basa en seleccionar probabilidades mediante una función sigmoidal de forma que nos de la probabilidad de que una instancia pertenezca o no a una clase. La única diferencia nuevamente con respecto a lo que conocemos de teoría es el escenario de varias clases. En este caso el modelo o comportamiento definido para obtener un resultado es enfrentar una clase contra el resto. De esta forma lo que obtenemos es la probabilidad de que un elemento pertenezca a una clase o al resto de las clases y así obtendremos tantas funciones sigmoidales como número de clases tengamos y con ello podremos clasificar el total de las clases para cada instancia. Por defecto la implementación de este algoritmo incluye regularización por lo que nos dará la ventaja de que no se producirá sobreajuste que nos penalice en el conjunto de test o al menos lo intentaremos evitar.

- Passive-Aggressive: este algoritmo es en realidad una familia de algoritmos que no tienen parámetros y que incorporan por defecto regularización para no cometer overfitting. Estos algoritmos tienen como origen el algoritmo Perceptrón partiendo de la base de un espacio separable con ciertas mejoras.

- Perceptrón: de igual forma que con SGD o Logistic Regression este algoritmo ya nos es conocido de clase e incluso de haberlo implementado en prácticas anteriores para el caso de clasificación binaria. La única diferencia que tenemos ahora es que el escenario es de clasificación con varias etiquetas por lo que necesitamos estudiar cómo funciona para este escenario concreto. El modelo es muy parecido a la estrategia empleada con SGD. Lo que intentamos hacer es lanzar algoritmos Perceptrón de clasificación binaria que nos den una separación lineal lo mejor posible entre cada una de las clases y el resto. De forma que con varias funciones lineales podamos separar el espacio. Cuando queramos predecir una instancia tendremos gracias a todas las funciones lineales definidas una serie de regiones que se corresponden con cada una de las clases.

1.7. Selección y ajuste del modelo final

Vamos a estudiar los resultados obtenidos para todos los tipos de preprocesamiento y reducción de dimensionalidad de los algoritmos que hemos seleccionado:

Algoritmo	Sin reducción de dimensionalidad			
	Sin	Estandarización	Normalización	N+E
SGD	0.9506	0.9470	0.9381	0.9435
Logistic Regression	0.9548	0.9601	0.9439	0.9541
Passive-Aggressive	0.9443	0.9375	0.9430	0.9432
Perceptrón	0.9463	0.9292	0.9368	0.9243

Algoritmo	Reducción: PCA 28 variables			
	Sin	Estandarización	Normalización	N+E
SGD	0.9346	0.9408	0.9439	0.9250
Logistic Regression	0.9419	0.9475	0.9410	0.9479
Passive-Aggressive	0.9074	0.9328	0.9392	0.9359
Perceptrón	0.8133	0.9214	0.9263	0.9190

Algoritmo	Reducción: PCA Incremental 28 variables			
	Sin	Estandarización	Normalización	N+E
SGD	0.9452	0.9361	0.9470	0.9361
Logistic Regression	0.9450	0.9492	0.9403	0.9537
Passive-Aggressive	0.9252	0.9279	0.9441	0.9370
Perceptrón	0.8451	0.9268	0.9417	0.9219

Algoritmo	Reducción: Kernel PCA 28 variables			
	Sin	Estandarización	Normalización	N+E
SGD	0.9459	0.9354	0.9381	0.9277
Logistic Regression	0.9455	0.9512	0.9401	0.9461
Passive-Aggressive	0.9217	0.9323	0.9357	0.9266
Perceptrón	0.8867	0.9221	0.8943	0.9121

Algoritmo	Reducción: Factor Analysis 47 variables			
	Sin	Estandarización	Normalización	N+E
SGD	0.9326	0.9314	0.9381	0.9223
Logistic Regression	0.9510	0.9501	0.9430	0.9475
Passive-Aggressive	0.9399	0.9328	0.9386	0.9359
Perceptrón	0.9301	0.9232	0.9223	0.9194

Como podemos observar en la tabla para cada algoritmo los mejores resultados que hemos obtenido son:

Algoritmo	Mejor combinación		
	Reducción	Preprocesamiento	Score
SGD	Sin	Sin	0.9506
Logistic Regression	Sin	Estandarización	0.9601
Passive-Aggressive	Sin	Sin	0.9443
Perceptrón	Sin	Sin	0.9463

Como podemos observar el ganador de forma ligera con respecto al resto es el algoritmo Logistic Regression y además resulta especialmente curioso el hecho de que como mejor funcionan los algoritmos es sin preprocesamiento. Vamos a intentar observar el conjunto de datos proyectado sobre dos dimensiones para razonar el por qué de estos resultados.

En primer lugar cabe decir que la ténica de reducción de dimensionalidad que estoy aplicando es TSNE que es una técnica ampliamente utilizada en reducción de dimensionalidad orientada a visualización de datos. En este caso he reducido a dos dimensiones y como resultado he obtenido la siguiente imagen:

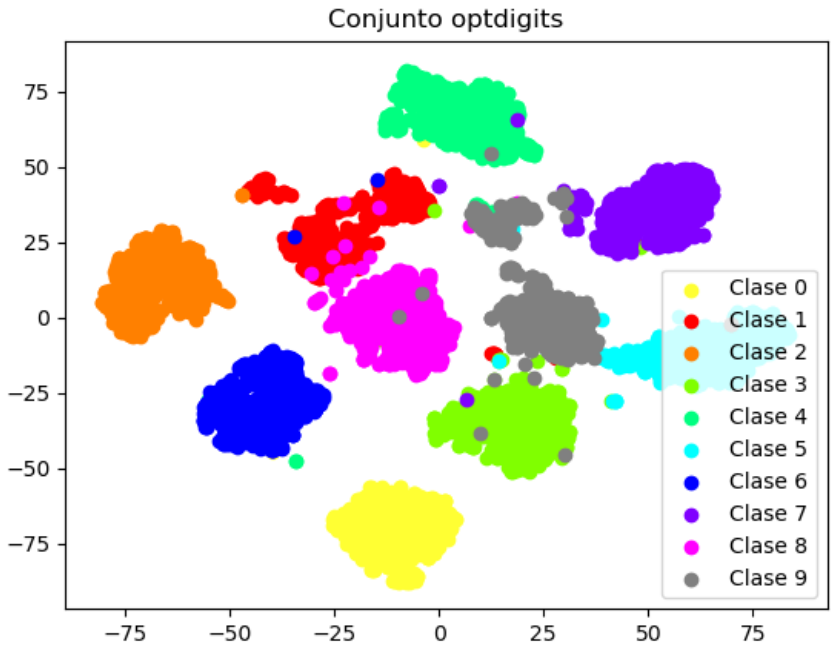

Figura 1: Datos dibujados como proyección 2D

Como podemos observar en el intento de dibujo de los datos éstos están claramente bien separados en clusters y el espacio queda separable de forma muy satisfactoria. Es por esto que el preprocesamiento no se hace necesario en este caso pues los datos de por sí ya son suficientemente buenos como para la aplicación de todas las técnicas que conocemos y hemos estudiado.

Por tanto el modelo final que he decido ajustar es Regresión Logística y por tanto este es en el que vamos a ajustar los parámetros.

Yo he decidido tomar los siguientes:

- max_iter: Número máximo de iteraciones tomado 10.000
- tol: tolerancia, tomado 10^-5
- C: parámetro de regularización, tomado 1.0
- random_state: semilla de generación aleatoria, tomada 123456789
- solver: algoritmo usado para la minimización interna del error. En este caso lbfgs.
- multi_class: tipo de esquema para clasificación con varias clases, tomado multinomial.
- verbose: información por pantalla del algoritmo.

Como resultado final he conseguido obtener 0.9521 de score lo cual es un poco menor que en el caso anterior (probablemente por cuestiones de generación aleatoria).

1.8. Idoneidad de la métrica usada en el ajuste

La métrica usada para la evaluación de los resultados es tomar el conjunto de train y entrenar el modelo seleccionado y realizar una predicción de los datos de test para posterior comparación.

Por tanto la métrica de evaluación es comparar las etiquetas reales con las predichas por el modelo y obtener el tanto por 1 de acierto de la predicción sobre los datos reales.

Esta métrica no es una métrica del todo idónea pues no se tienen en cuenta los fallos de los falsos positivos contra los falsos negativos.

Para entender esta métrica debemos primero definir los conceptos de precisión y exhaustividad.

$$precision = \frac{tp}{tp + fp}$$

$$exhaustividad = \frac{tp}{tp + fn}$$

Donde tp es los verdaderos positivos, fp falsos positivos y fn falsos negativos.

El score F1 es la media armónica de la precisión y la exhaustividad, esto es:

$$F_1 = 2 \cdot \frac{precision \cdot exhaustividad}{precision + exhaustividad}$$

Esto se realiza para cada una de las clases, por lo que podemos obtener el score separado o hacer la media. Veamos los resultados:

Algoritmo	Medidas	
	Acierto	Media F1
Logistic Regression	0.9521	0.9583

Algoritmo	Medidas									
	F1 C0	F1 C1	F1 C2	F1 C3	F1 C4	F1 C5	F1 C6	F1 C7	F1 C8	F1 C9
LR	0.9909	0.9343	0.9673	0.9516	0.9510	0.9638	0.9810	0.9800	0.9311	0.9583

Como podemos observar aquí tenemos el score desgranado por clases. Podemos deducir que hemos acertado más en la clase 0 y en la clase 7, por lo que suponemos que la complejidad de detectar estos dígitos es menor para el clasificador. Los dígitos más complejos y por tanto en los que más fallamos son el 1 y el 8, por lo que suponemos que son los que más complejidad presentan. El score F1 medio ha sido de 0.9583 por lo que el score que nos provee sklearn y el propio F1 nos arrojan una información similar.

1.9. Estimación de E_{out}

La estimación del error fuera de la muestra la podemos obtener a partir del acierto que hemos obtenido en el apartado anterior. En el caso de regresión logística que ha resultado ser el mejor

de nuestros modelos hemos obtenido un tanto por uno de acierto de 0.9521 en el conjunto de test. En nuestro caso el conjunto de train es el tomado como la muestra y por tanto para poder calcular el error fuera de la muestra nos basaremos en el conjunto de test. Si el acierto fuera de la muestra es de 0.9521 entonces el error fuera de la muestra será $1 - 0{,}9521 = 0{,}0479$.

1.10. Justificación del modelo y calidad del mismo

Por tanto, por los resultados obtenidos considero que el modelo ofrece una buena calidad pues obtenemos un error fuera de la muestra pequeño, o lo que es lo mismo, un acierto fuera de la muestra alto.

La calidad del modelo, viene del hecho de que, como ya hemos razonado anteriormente, el conjunto de datos era fácilmente separable por lo que casi todos los algoritmos que hemos estudiado en clase han dado unos resultados parejos siendo quizás el más descolgado el algoritmo Perceptrón.