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

1.7. Selección y ajuste del modelo final

1.8. Idoneidad de la métrica usada en el ajuste

1.9. Estimación de E_{out}

1.10. Justificación del modelo y calidad del mismo