Trabajo 3: Programación

Patricia Córdoba Hidalgo

Índice

1. Problema de clasificación

Se implementó en el archivo `clasificacion.py`.

Este problema consiste en clasificar imágenes de dígitos escritos a mano, asignándole a cada una el dígito que representan. Nuestro espacio de características, χ, está formado por datos de 64 características, cada una de ellas representando la intensidad de trazado (un número del 0 al 16) de una de las casillas de una matriz 8×8, donde se ha trazado el dígito. El conjunto de etiquetas, Y, son los dígitos del 0 al 9. La función desconocida f es aquella que a cada imagen (elemento de χ) le asigna el dígito que representa (su etiqueta correspondiente). Este es un problema de aprendizaje supervisado, concretamente de clasificación.

Primero leemos los ficheros de datos con la función `readData`, que separa los datos de sus etiquetas. Los datos del fichero test serán los datos que usaremos para calcular el E_{out} y los del fichero training los dividimos en datos de entrenamiento y datos de validación. Un 25 % de los datos del fichero de datos training serán usados para validar. En un problema de aprendizaje corriente lo ideal sería mezclar los datos de ambos ficheros, ya que normalmente nos dan una base de datos y no tenemos un conjunto test, y dividir de ahí los conjuntos de entrenamiento, validación y test. En este caso, esto no es plausible, ya que los datos son números dibujados por personas y si mezclamos los ficheros existe la posibilidad de que haya dígitos trazados por la misma persona tanto en el conjunto training como en el test. Si esto ocurriese, como el algoritmo se entrenó para adaptarse a la caligrafía de ese individuo, la estimación del error fuera de la muestra que nos porporciona el conjunto test no sería realista. Por esto, en este caso, no mezclamos los ficheros, dejamos que el fichero test se use como conjunto test.

Las clases de funciones que usaré para resolver el problema son los polinomios de grado uno y dos. Usaré estas clases de funciones porque son relativamente sencillas y no tienen tanta variabilidad como otras clases de funciones, así no habrá tanta tendencia al sobreajuste y el error E_{out} no será tan alto.

Antes de empezar a trabajar con los datos, se preprocesan. Primero se usa PCA, para deshechar aquellas características que no aporten mucha información al modelo, y nos quedamos con aquellas capaces de explicar el 99 % de la distribución. Tras esto, escalamos los datos, ya que, si una característica tiene una varianza varios órdenes mayor que otra, puede tener repercusiones en el cálculo de la función objetivo y puede hacer que no se aprenda correctamente del resto de características.
Estas transformaciones se ajustan a los datos de entrenamiento (con la función `fit`) y luego se usan las mismas sobre el resto de datos (tanto al conjunto de validación como al de test).

Para clasificar los datos, usaremos regresión logística multietiqueta. Por el teorema de "No-Free-Lunch" todos los algoritmos son iguales en media, no hay un algoritmo mejor que otro, hay problemas donde unos tienen mejor desempeño que otros y elegí usar este algoritmo inicialmente porque ya lo hemos trabajado en otras prácticas, es sencillo y hemos visto como usarlo en caso de clasificación multietiqueta. Los resultados obtenidos fueron buenos, por lo que concluí que era un algoritmo con buen desempeño en éste problema.

La implementación del algoritmo de *Regresión Logistica* está en la función `rl_sgd`. La función `rl_sgd` devuelve una lista de 10 vectores, cada uno de ellos calculado en una de las iteraciones del bucle principal (usando gradiente descendente estocástico y el error visto en teoría), que corresponde a los pesos de la función que separa esa clase del resto. Los hiperparámetros usados son: $\eta = 0{,}005$, se hacen 100 iteraciones y el tamaño de minibatch usado es 32. Probando con diferentes valores para η, éste fue el que mejor resultado produjo en conjunto en todos los modelos usados, así que es el que se dejó al final. El número de iteraciones y el tamaño de minibatch se ajustaron según un criterio de calidad-tiempo, ya que con muchas más iteraciones, el tiempo que tarda el programa en ejecutarse es demasiado, y con esta proporción se obtenían resultados aceptables.
Antes de ejecutar esta función, es necesario ajustar el formato de las etiquetas, pasando de tener dígitos a tener vectores de 10 coordenadas. A la etiqueta que representa el dígito i se le asocia el vector e_i, aquel que tiene ceros en todas sus componentes menos en la posición i, que tiene un 1. La etiqueta 0 pasa a ser [1 0 0 0 0 0 0 0 0 0], la etiqueta 1 pasa a ser [0 1 0 0 0 0 0 0 0 0], y así sucesivamente, hasta la etiqueta 9 que pasa a ser [0 0 0 0 0 0 0 0 0 1].

El error que minimizamos usando el algoritmo de *Regresión Logistica* es la pérdida logarítmica, que nos permite penalizar mucho cuando la estimación es errónea y que el error sea muy cercano a cero cuando la estimación es buena. Sin embargo, la métrica que usaremos para elegir el mejor modelo será la precisión (accuracy), que es el porcentaje de etiquetas que acierta nuestro modelo. Usaré esta métrica para ver realmente cuál es el desempeño del modelo y cómo de bien clasifica las etiquetas. Podemos usar esta métrica en nuestro caso porque los datos que pertenecen a cada clase están casi balanceados (en el fichero `optdigits.names` la cantidad de dígitos de cada clase).

Los resultados obtenidos son:

	CARACTERÍSTICAS LINEALES	**CARACTERÍSTICAS CUADRÁTICAS**
Error del conjunto training	0.23515871536552188	0.04682725465198909
Error conjunto de validación	0.3144886201093401	0.1374816964128216
Error del conjunto test	0.32641955137068873	0.17682268491873723
Precisión del conjunto training	0.9518828451882845	0.9923291492329149
Precisión conjunto de validación	0.9455497382198953	0.9675392670157068

El modelo con características cuadráticas consigue una mayor precisión que el modelo con características lineales, es decir, consigue asignar la etiqueta correcta al mayor número de datos. El error dentro de la muestra es considerablemente menor que con características lineales. Esto se debe a que la clase de funciones es más rica y puede ajustarse mejor a los datos, aunque en el proceso puede producirse sobreajuste. Es por esto que la regularización es importante, para evitar que el modelo se ajuste demasiado bien a los datos de entrenamiento y no tan bien a los datos de la distribución. Nosotros, que poseemos un conjunto de test, podemos ver que realmente el modelo con características cuadráticas se ajusta mejor a los datos de fuera de la muestra, pero esta información no debería impulsarnos a tomar nunguna decisión ya que, en teoría, esta información no la poseemos.

Para comprobar que no hemos sobreajustado el modelo a los datos y para limitar la varianza de la clase, que influye en el error E_{out}, es muy importante hacer siempre regularización. En mi caso, para regularizar usé la función `rl_reg` que usa el algoritmo de *Regresión Logistica* minimizando ahora el error $E_{aug} = E_{in} + \lambda||w||^2$, que es equivalente a imponer la restricción de que $||w||^2$ no supere cierta constante, y así acotamos la variabilidad de la clase. En el algoritmo, como calculamos cada uno de los 10 vectores de peso por separado, w sí es un vector, pero en el cálculo del error de regularización, con `Err_reg`, w es una matriz (contiene a los 10 vectores), luego lo que usamos es la norma de Frobenius, que es equivalente a la norma 2 por filas. Se tomó $\lambda = 0{,}0001$, ya que fue el valor de lambda que mejores resultados obtuvo entre los probados (mayor precisión). Este valor de λ es el que se usará siempre qe se haga regularización (para comparar modelos, con el modelo final y para la validación cruzada).

Los resultados obtenidos con la regularización son:

	CARACTERÍSTICAS LINEALES	CARACTERÍSTICAS CUADRÁTICAS
Error del conjunto training	0.2144840793108922	0.04325257298200755
Error conjunto de validación	0.2731787566521759	0.1443876192810912
Error del conjunto test	0.3548323883658652	0.20759725788092545
Precisión del conjunto training	0.9616457461645747	0.9905857740585774
Precisión conjunto de validación	0.9539267015706806	0.9581151832460733

Como podemos observar, el error tanto dentro como fuera de la muestra aumenta, ya que ahora el error que usamos penaliza también el tamaños del w resultante. Por esto nos basamos en la precisión para ver la bondad del modelo, ya que el porcentaje de datos que clasifica correctamente no depende del tamaño de los pesos elegidos, solo depende del buen desempeño del modelo. En este caso, el modelo con características lineales mejora con los datos de validación. Hemos conseguido que, dismunuyendo la varianza, se obtenga un modelo que se ajuste mejor a los datos de validación. En el modelo con características cuadráticas, la precisión empeora en el conjunto de validación, ya que al añadir las restricciones, las funciones que ahora consideramos tienen mayor sesgo y no compensa con el descenso de varianza producido.

Como el modelo con mayor precisión en el conjunto de validación es aquel con características cuadráticas, ésta es la clase que usaremos, nuestro "mejor modelo". Ahora, calculamos el w que se ajuste a todos los datos del fichero training. Este será el modelo definitvo, aquel que propondríamos como solución del problema de aprendizaje. Una vez calculado, calculamos la precisión del modelo con el conjunto test.

En la función de validación cruzada se devuelve la precisión media del modelo. En teoría sabemos que, en media, el error de validación cruzada es una cota pesimista del error E_{out}, luego podemos usar ese error para acotarlo. En nuestro caso, deberíamos obtener una cota pesimista de la mínima precisión que tendría nuestro modelo. Sin embargo, la precisión del modelo por validación cruzada no es una cota pesimista de la precisión fuera de la muestra. Esto se debe a que los datos del conjunto training están escritos por las mismas personas y los del conjunto test por otras diferentes. Por eso, el modelo se ajusta mejor a los datos de validación que a los datos de test, ya que se ha entrenado con esa caligrafía, y proporciona mejores resultados con los datos de validación que con los de test.

Si una empresa me encargara el problema, el modelo que le daría sería el de características cuadráticas, porque según la métrica de error que he utilizado, la precisión, es aquel que mejor comportamiento tiene. La cota que les daría sería que el modelo, en el peor de los casos, tiene una precisión del 93,796 %. Esto sabemos que es falso por el problema de sobreajuste provocado por la caligrafía de la misma persona en los datos de entrenamiento y validación, y sabemos que en nuestro conjunto test tenemos una precisión del 92,988 %, que es menor a la cota dada, pero como en un problema real no contamos con esta información, lo mejor que le podemos decir a nuestro jefe es eso. Aun así, el modelo se ajusta bien a los datos, ya que consigue una precisión del 92,988 % en el conjunto de test.

Al final del ejecutable, se muestra, a modo de ejemplo, el funcionamiento del modelo usando SOFTMAX. SOFTMAX crea un vector donde en cada casilla escribe la probabilidad de que el dato x pertenezca a esa clase. En nuestro caso, en la casilla i está la probabilidad de que el dato x sea el dígito i (si empezamos a contar las casillas por 0). Al dato se le asigna la etiqueta con mayor probabilidad de ocurrencia. Saqué por pantalla la etiqueta real, la etiqueta que se predice y la probabilidad de ésta. En este ejemplo, siempre acierta la etiqueta (Pero podría no acertar alguna, ya que la precisión no es del 100 %).

2. Problema de regresión

Se implementó en el archivo `regresion.py`.

Este problema consiste en predicir el número de crímenes violentos en una comunidad de USA por cada 100000 habitates a partir de datos socio-económicos de la comunidad. Nuestro espacio de características, χ, está formado por datos socio-económicos de la comunidad, los cinco primeros no predicativos (uno de ellos el nombre de la comunidad) y el resto son valores numéricos normalizados. El conjunto de etiquetas, Y, es el número de crímenes violentos en esa comunidad de USA por cada 100000 habitates. La función desconocida f es aquella que a cada comunidad (elemento de χ) le asigna el número de crímenes violentos correspondiente (su etiqueta correspondiente). Es un problema de aprendizaje supervisado, concretamente de regresión.

Primero leemos los ficheros de datos con la función `readData`. En este caso, a diferencia del problema de clasificación, los datos son del tipo `DataFrame` y no `numpy`, ya que éste tipo de datos tiene varias funciones útiles para el procesamiento de dats que `numpy` no posee. Dividimos el conjunto de datos de tal manera que el 60 % sea el conjunto training, 20 % sea validación y 20 % sea test. Esto se hace con la función `train_test_split`. Como sólo divide en dos conjuntos, primero cogemos el 20 % de los datos para test y luego dividimos el otro conjunto en 75 % para entrenamiento y 25 % para validación, ya que el 25 % del 80 % coincide con el 20 % del 100 %.

Las clases de funciones que usaré para resolver el problema son los polinomios de grado uno y dos, como en el problema anterior. Como ya justifiqué anteriormente, estas clases de funciones son relativamente sencillas y no tienen tanta variabilidad como otras clases de funciones, así no habrá tanta tendencia al sobreajuste y el error E_{out} no será tan alto.

Antes de empezar a trabajar los datos, se preprocesan. Como las primeras cinco características no son predicativas, es decir, no aportan ninguna información a nuestro problema, las eliminamos. Tras esto eliminamos también las características que tengan más de un 1 % de valores desconocidos en el conjunto de entrenamiento. Así, nos quedamos sólo con características cuyo valor es conocido en la mayoría de los datos y evitamos meter ruido a la muestra intentando aproximar un gran número de valores desconocidos. Estas mismas características las eliminamos también del conjunto test y del de validación.
Una vez que hemos eliminado estas características, si hay datos con el valor de alguna característica desconocida, lo aproximamos. Hacemos esto para no perder información de una característica si solo hay muy pocos datos con ese valor desconocido. El valor que introducimos para sustituir el desconocido es la media de los valores de esa característica en el resto de datos del conjunto correspondiente a ese dato (se hace en los tres conjuntos). Todo este preprocesamiento se hizo usando métodos de la clase `DataFrame`.

Una vez que nuestros datos no tienen nungún valor desconocido, preprocesamos los datos con el mismo procedimiento que usamos con el problema de clasificación. Primero usamos PCA para quedarnos sólo con las características que representen el 99 % de la distribución y tras esto escalamos los datos. Como justificamos en el problema anterior, esto se hace porque si una característica tiene una varianza varios órdenes mayor que otra, puede tener repercusiones en el cálculo de la función objetivo y puede hacer que no se aprenda correctamente del resto de características.
Estas transformaciones se ajustan a los datos de entrenamiento (con la función `fit`) y luego se usan las mismas sobre el resto de datos (tanto al conjunto de validación como al de test).
También ajustamos el formato de los datos con la función `formato_datos` para que la primera característica de todos ellos sea un 1. Esto lo hacemos para que el algoritmo nos devuelva los parámetros de un hiperplano afín, y no esté obligado a que sea un hiperplano vectorial.

El algoritmo que usaré para el ajuste de los datos es el algoritmo de *La matriz Pseudoinversa*. Usaré ese porque cuando la matriz no es excesivamente grande, es muy rápido y obtiene buenos resultados en los problemas de regresión.

La métrica de error usada es el error cuadrático medio, que es el error que minimiza el algoritmo de *La matriz Pseudoinversa*. Este error siempre es positivo y sólo es cero cuando el ajuste es perfecto, ya que tiene en cuenta todos los datos y el error que se produce en ellos. Así, esta métrica usa toda la información disponible e intenta ajustarse lo mejor posible al conjunto de los datos, no intenta ajustar muy bien una parte a costa de perjudicar otra. Esta métrica es la más simple y la más común en problemas de regresión.

Para calcular el error del modelo implementé dos funciones diferentes: `Err` y `Err_np`. Esto se debe a que las etiquetas están implementadas sobre un `DataFrame` en la mayor parte del código, pero en la función de `validacion_cruzada`, pasan a ser vectores de `numpy` al usar la función de `concatenate`. Como el acceso a los datos es diferente en estas estructuras, fue necesario implementar las dos funciones diferentes. Los conjuntos deentrenamiento, validación y test son vectores de `numpy` desde que se le aplica el algoritmo de `PCA`.
Los resultados obtenidos son:

	CARACTERÍSTICAS LINEALES	**CARACTERÍSTICAS CUADRÁTICAS**
Error del conjunto training	0.01706133273347002	0.003534236848716267
Error conjunto de validación	0.018866052767704204	0.03636803782422322
Error del conjunto test	0.02060827214004634	0.047212466396276784

Como podemos observar, el modelo con características lineales tiene mayor error en la muestra que el modelos con características cuadráticas. Sin embargo, este modelo tiene un mejor comportamiento con el conjunto de validación, difiriendo relativamente poco el error del conjunto training respecto al test, mientras que el modelo con características cuadráticas tiene un error en el conjunto de validación más de diez veces mayor que el error en el conjunto training. Esto se debe a que, al ser una clase más rica en funciones, podemos encontrar una que se ajuste mucho a los datos de la muestra, aunque su desempeño fuera de ésta sea peor. Se está produciendo sobreajuste con el modelo de características cuadráticas. El otro modelo, al ser más sencillo, no tiene a penas sobreajuste.

Para combatir con el sobreajuste producido, es necesario usar regularización. Usamos el algoritmo de *La matriz Pseudoinversa* regularizado, esto es, ahora w no es $(X^TX)^{-1}X^Ty$, si no que es $(X^TX+\lambda I)^{-1}X^Ty$. Los errores que mostramos son usando la métrica escogida. No mostramos el error aumentado, que es el que se minimiza con regularización, sino el error cuadrático medio, que es el elegido para comparar los modelos entre sí.
Los resultados obtenidos son:

	CARACTERÍSTICAS LINEALES	**CARACTERÍSTICAS CUADRÁTICAS**
Error del conjunto training	0.01706133273353623	0.005245244929496423
Error conjunto de validación	0.018866055076277446	0.034677097266386805
Error del conjunto test	0.020608266965906483	0.04193181913215595

Podemos ver que con el modelo con características lineales la regularización no ofrece una gran mejora, los cambios en el error son mínimos e incluso son un poco mayores que en el caso sin regularización (en el conjunto de entrenamiento y de validación). Es un modelo muy simple que no sobreajusta apenas. El modelo con características cuadráticas sí sobreajustaba y el uso de regularización sí hace que el error en la muestra aumente a cambio de que mejore el error del conjunto de validación. Sin embargo, no mejora lo suficiente para elegir este modelo, ya que el modelo lineal presenta menor error minimo cuadrático en el conjunto de validación.

Los hiperparámetros usados para la regularización son 0,001 con el modelo lineal y 172 en el modelo cuadrático, ya que son los que mejores resultados tenían por separado en cada modelo. Esto lo hice para demostrar que, ni eligiendo el mejor hiperparámetro posible con el modelo con características cuadráticas obtenemos mejores resultados en el conjunto de validación que con el modelo con características lineales. En el problema anterior intenté ser justa y aplicar a todos los modelos las mismas restricciones, aquí, como parece que el modelo con características cuadráticas sobreajusta mucho, por la gran diferencia entre su error en la muestra y en el conjunto de validación, intenté buscar el mejor parámetro para éste y aun así no conseguí disminuirlo más. Es por eso que dejé este valor intacto, para dejar claro que no tiene mejores resultados que el lineal en este caso.

El modelo que tiene menor error cuadrático medio en el conjunto de validación es el modelo con características linaeles sin regularización, por tanto este sería mi "mejor modelo".

Una vez identificaco cual es el modelo a usar, el modelo con características linaeles sin regularización, fusionamos los datos del conjunto training y el conjunto de validación para obtener el w que propondríamos como solución a nuestro problema de aprendizaje.

Para dar una cota del error fuera de la muestra, usamos la función de `validacion_cruzada`. Sabemos que el error de validación cruzada es, en media, una cota pesimista del error E_{out}. En nuestro caso obtenemos que $E_{out} = 0{,}017512795613507298 < 0{,}01904200368347531 = E_{cv}$. Por tanto, efectivamente obtenemos una cota del error fuerra de la muestra con validación cruzada.

Si fuese la encargada de este ajuste para una empresa, les propondría el modelo con características lineales sin regularizar, que consigue un error menor o igual que 0,01904200368347531. Todas estas decisiones se han tomado en base a los resultados del error del conjunto de validación, que es tan aleatorio como el conjunto de datos test. El conjunto test tiene en general mayor error que el error de validación, y es verdad que el error de este conjunto con el modelo lineal con regularización es menor que sin regularización, así que en caso de haber usado el conjunto test como conjunto de validación, nuestras conclusiones habrían sido diferentes, pero como el modelo con características lineales no tiene sobreajuste apenas por la sencillez del modelo, usar o no regularización no afecta mucho a los resultados.