

Trabajo 3: Programación

Patricia Córdoba Hidalgo

Índice

| | |
|------------------------------|---|
| 1. Problema de clasificación | 1 |
| 2. Problema de regresión | 4 |

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 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).

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 proporciona 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 Logística* 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. Probandos 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 $[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1]$.

El error que minimizamos usando el algoritmo de *Regresión Logística* 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.

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 ninguna 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 Logística* 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 que 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 | 0.41895422496760854 | 0.1850746588024572 |
| Error conjunto de validación | 0.8872315082160108 | 0.5702993911481589 |
| Error del conjunto test | 0.6811653754239967 | 0.4339443597913593 |
| 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 tamaño 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, disminuyendo 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 definitivo, 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.

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 predecir el número de crímenes violentos en una comunidad de USA por cada 100000 habitantes a partir de datos socio-económicos de la comunidad. Nuestro espacio de características, χ , está formado por datos socio-económicos de la comunidad. El conjunto de etiquetas, Y , es el número de crímenes violentos en esa comunidad de USA por cada 100000 habitantes. La función desconocida f es aquella que a cada comunidad (elemento de χ) le asigna el número de crímenes violentos correspondiente (su etiqueta correspondiente).

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.

TO KNOW PANDAS: - LEER FICHERO - DROP - QUITAR VALORES NULOS - QUITAR NO PREDICTIVE (LAS 5 PRMERAS CARACTERISTICAS)