Breast Cancer Wisconsin - APA

Gonzalo Diez, Jaime Pascual

Enero 2016

Índice

1. Introducción

El problema que vamos a trabajar es el de predecir si existe un cáncer mamario a partir de treinta características de núcleos celulares de la masa mamaria. Estas características son extraídas automáticamente de fotografías digitalizadas de masas mamarias.

Para hacer este estudio, utilizaremos los datos que encontraremos en el siguiente enlace (http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/)

En el estudio, trataremos de buscar el mejor modelo posible para hacer la predicción correcta. Para esto, probaremos diferentes tipos de modelos, lineales y no lineales, y los compararemos para sacar el mejor de ellos. Finalmente, conseguiremos un modelo lineal con un error de train y test bastante bajo, menores al 1.6 %.

2. Procesamiento de los datos

Para poder trabajar cómodamente sobre los datos, primero deberemos tratarlos. Primero hemos mirando si había algún valor perdido y hemos visto que no lo hay. Por suerte podemos confiar en que los datos son correctos, ya que están sacados automáticamente a través de un sistema informático, o al menos, podemos confiar en que no habrá habido ningún error humano.

Para ayudar a algunos de los modelos que probaremos, normalizaremos los datos.

3. Selección del modelo

Para seleccionar el modelo probaremos diferentes técnicas de machine learning, utilizando el sistema training/test. Para el training elegimos dos tercios aleatorios de los datos totales, y luego comprobamos como de bien predice nuestro modelo comparando el tercio restante de los datos contra la predicción del modelo.

Para poder reproducir el estudio, hemos utilizado siempre la misma semilla en R.

Las técnicas que probaremos serán algunas lineales (svm lineal, svm polinómica de hasta grado 7, ...) y otras no lineales (svm RBF, MLP con una capa oculta).

3.1. Support Vector Machines

Puesto que nuestro problema es de clasificación de dos clases, hemos decidido probar con svm, ya que según hemos visto en clase, son un buen acercamiento para este tipo de problemas. Al hacer el estudio, hemos probado con diferentes tipos de kernels y diferentes C para sacar diferentes modelos de comparar después usando LOOCV. Hemos normalizado los datos para hacer más fácil el entrenamiento de los svm.

Los resultados obtenidos los podemos ver en cuadro 3.1 y más gráficamente en la figura 1 (Obviamos cuando la C es igual a 0.001 y 0.01, ya que sus valores son ampliamente mayores).

C	0.001	0.01	0.1	1	10	100	1000
Linear	6.596306	3.430079	2.902375	3.430079	3.957784	3.693931	3.693931
Poly2	36.411609	10.026385	4.749340	3.166227	3.693931	4.221636	4.221636
Poly3	31.134565	8.179420	3.957784	2.374670	2.902375	3.957784	3.957784
Poly4	26.649077	7.651715	2.902375	2.374670	4.485488	4.221636	4.221636
Poly5	23.482850	7.124011	3.430079	2.374670	4.749340	4.749340	4.749340
Poly6	21.108179	7.387863	3.166227	2.374670	4.749340	4.749340	4.749340
Poly7	19.788918	6.860158	3.693931	2.638522	5.277045	5.277045	5.277045
RBF	37.467018	37.467018	6.860158	2.902375	2.638522	3.430079	3.430079

Cuadro 1: Resultados de LOOCV con diferentes C y diferentes kernels

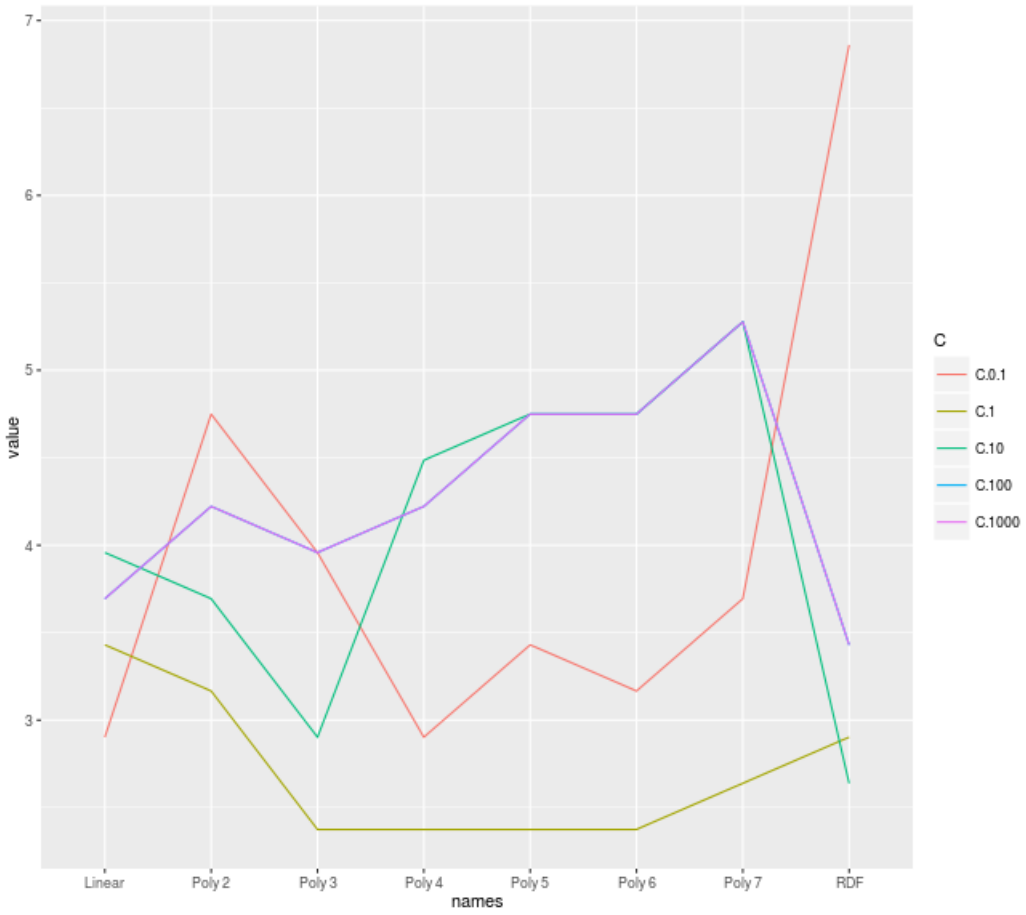

Figura 1: Comparacion de diferentes C con diferentes kernels

Como podemos observar, los mejores resultados de LOOCV es cuando la C es igual a 1.

Una vez tenemos fijada la C a 1, trataremos de buscar el mejor valor de gamma probando otra vez con diferentes kernels. Los resultados los podemos ver en el cuadro 3.1 y más gráficamente en la figura 2

Gamma	0.0625	0.125	0.25	0.5	1	2	4
Linear	3.430079	3.430079	3.430079	3.430079	3.430079	3.430079	3.430079
Poly 2	2.638522	2.902375	2.638522	6.068602	6.860158	6.332454	6.596306
Poly 3	2.638522	4.221636	5.277045	6.068602	6.068602	6.596306	6.860158
Poly 4	2.638522	5.804749	6.860158	8.707124	9.234828	12.137203	13.720317
Poly 5	3.957784	6.068602	6.860158	7.651715	8.707124	8.443272	7.915567
Poly 6	5.804749	6.596306	7.387863	10.290237	13.192612	16.886544	17.941953
Poly 7	6.860158	7.387863	8.707124	9.234828	10.290237	10.290237	10.554090
RDF	3.693931	5.277045	8.179420	22.163588	37.203166	37.467018	37.467018

Cuadro 2: Resultados de LOOCV con diferentes Gammas y diferentes kernels

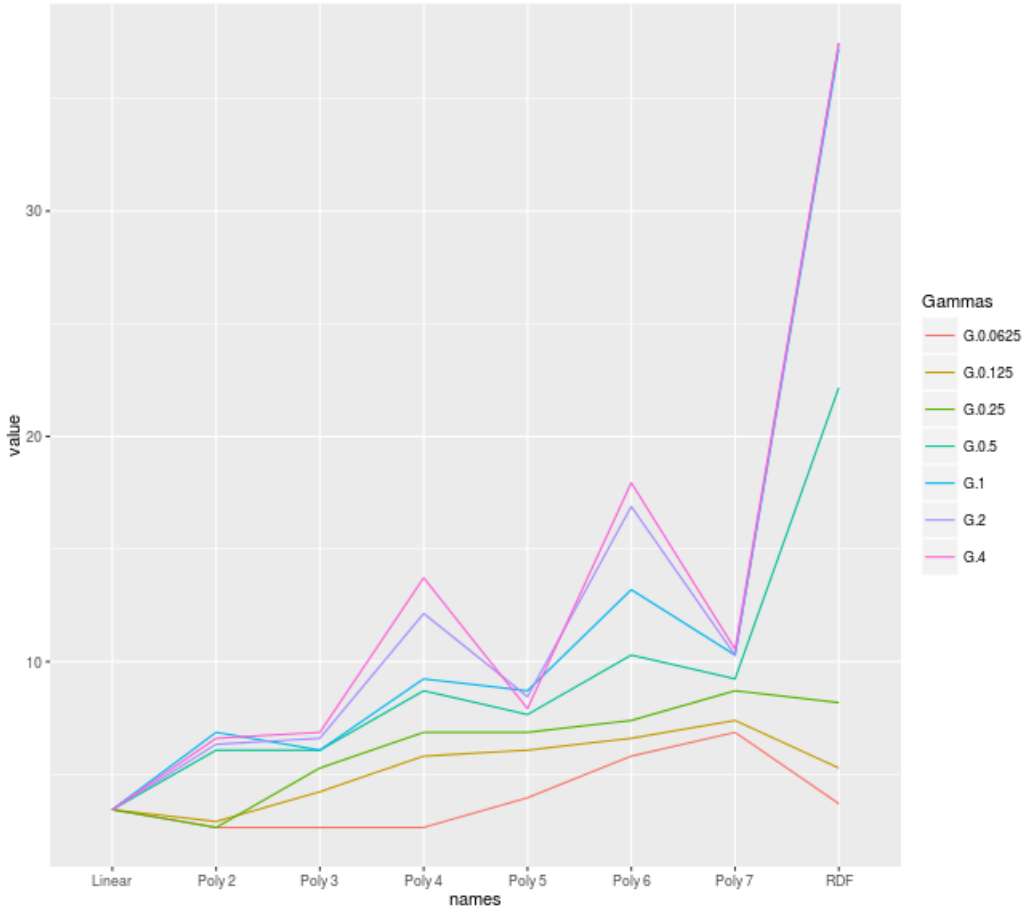

Figura 2: Comparacion de diferentes Gammas con diferentes kernels

Observamos que el mínimo LOOCV conseguido es de 2.638522. Este lo conseguimos con diferentes kernels y diferentes gammas. Para la selección del modelo, elegiremos el kernel polinómico con grado 2, ya que es mas sencillo que el 3 y el 4, que es con los que empata. Elegiremos una gamma de 0.0625.

Una vez hecho esto, probamos nuestro modelo con los datos de test que teníamos reservados para ello, dando los siguientes resultados:

		t_true	
pred		B	M
	B	119	2
	M	1	68

Con los resultados vemos que tenemos un modelo que tiene un porcentaje de acierto del 98.42105, únicamente con 54 Support Vectors (usamos únicamente el 14 % de los datos para sacar el hiperplano separador).

3.2. Linear discriminant analysis (LDA) y Quadratic classifier (QDA)

LDA y QDA son dos métodos lineales muy útiles para predecir dos o mas clases, en nuestro caso benigno o maligno. Mientras que LDA asume gausianidad entre las clases y matrices de covarianzas iguales QDA solo asume gausianidad, pero no covarianzas.

Vamos a probar los dos métodos con nuestros datos. Para cada método haremos dos ejecuciones, una con leave-one-out cross-validation (LOOCV) y otra sin.

El menor error lo obtenemos con QDA sin LOOCV el cual es estable en train y en test.

3.3. Naïve Bayes

Naive Bayes es un clasificador sencillo que asume independencia entre las variables.

No ha dado muy buen resultado. El error de train rondaba el 6 % y el de test el 5 %

3.4. k-nearest neighbors (kNN)

Este método se basa en los vecinos para predecir la clase de una entrada. Para ello hemos de estudiar cual es la mejor cantidad de vecinos (k). Sencillamente hemos probado de 0 a 20 vecinos y nos hemos quedado con la k que da el error mas bajo. Los mejores resultados se encuentran entre el k=3 y k=6.

El error medio de estos casos es 7,8 %.

3.5. Multilayer perceptron (MLP)

En este caso vamos a usar el perceptrón multicapa de una sola capa. Concretamente la rutina nnet de R.

Hemos empezado escalando los datos para evitar que la red converja demasiado rápido. Posteriormente vamos a probar la rutina ejecutándola con 2 neuronas y caída 0. Nos da un error realmente bajo, entre 0 % y 0.26 %, por lo cual creemos que hay sobre ajuste. Lo certificamos al probar con los datos de test, cuyo error es del 4 %

También vemos una gran diferencia entre el error al predecir la clase (B o M) y el error al predecir en raw (0 o 1) Vamos a ver la tabla de predicciones en este caso.

p1	B	M
0	231	0
3.45249441714434e-07	1	0
2.14419717950265e-06	1	0
2.15185572806224e-06	1	0
4.55085150221023e-06	1	0
5.31148710105192e-06	1	0
2.17240964816293e-05	1	0
0.9999662340364	0	1
0.999975258388252	0	1
0.999997997075748	0	1
0.999999170085833	0	1
1	0	138

Como se puede ver, lo que aumenta el error en modo raw son casos en los que está muy cerca de discernir (B o M) pero que no es exacto. En el modo raw seguimos viendo una diferencia del 4 % entre el error de train y el error de test.

Lo siguiente que probamos es la función train del paquete caret con el que vamos a probar distintas cantidades de la capa oculta hasta dar con la que mejor resultado nos de.

Después de varias ejecuciones vemos que la diferencia entre una cantidad de neuronas y otra es mínima, a veces 2 es suficiente y otras veces el mejor resultado es 20. Si vemos la tabla de resultados nos damos cuenta de que todas las opciones tienen una precisión de 0.95 Viendo que no tenemos suficiente diferencia como para tomar una decisión hemos optado por ejecutar el entrenamiento múltiples veces. Después de 10 ejecuciones vemos que a partir de 10 neuronas el resultado es mas elevado.

Ahora que ya hemos encontrado una cantidad de neuronas para nuestra capa oculta vamos a buscar la caída. El método train también nos permite pasarle una lista de caídas para buscar la mas óptima. En este caso nos ocurre lo mismo que en anterior, todas las opciones son prácticamente igual de válidas. Pero en este caso vemos mucho mas rápidamente que el valor 1 es el mas óptimo, el que menor error nos da.

Por último decidimos probar a llamar a nuestra rutina nnet con tamaño 10 y caída 1 y ver que errores de predicción nos devuelve.

El error de entrenamiento es 1,31 % mientras que el error de test es 2.10 % siguen estando distantes el uno del otro, hemos logrado reducir el sobre ajuste pero no eliminarlo.

4. Elección del modelo

En este estudio, hemos probado diferentes métodos de acercamiento del problema para ver cual daba mejor resultado. Finalmente nos hemos decantado por la Support Vector Machine con un kernel polinómico de grado dos, ya que ha sido la que nos ha dado un error menor en test y también porqué la diferencia entre el error de train y test era menor al 0.5 %. Al haber esta diferencia tan pequeña entre el error de train y test, estamos casi seguros de que el modelo no está entrenado haciendo overfitting.

5. Conclusiones

Estamos bastante satisfechos con el trabajo hecho. Hemos probado con unos cuantos tipos de modelos diferentes, lineales y no lineales, y esto nos a ayudado a entender mas profundamente los temas tratados en clase. También estamos contentos con el resultado obtenido, ya que un error del 1.579 % y sin overfitting, nos parece un modelo bastante bueno.

6. Posibles Extensiones y Limitaciones Actuales

Como posible extensión, en el caso de las svm's, podríamos haber tratado de optimizar el coeficiente de los kernels polinómicos, además de buscar más profundamente los valores para el parámetro regulador y la Gamma. Una posible limitación ha sido que el numero de muestras que teníamos no era demasiado extenso, y eso ha podido hacer que el MLP haya sobre ajustado.

7. Referencias

.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis https://www.aaai.org/Papers/Symposia/Spring/1994/SS-94-01/SS94-01-019.pdf