Proyecto APA

Lorenzo Sabater Joseba Sierra

Enero 2020

1. Descripción

Este trabajo consiste en resolver el problema planteado en la BCI competition sobre el dataset V [1], un problema multiclase de imágenes mentales. Datos correspondientes a José del R. Millán, IDIAP Research Institute [5].

Los datos consisten en muestras tomadas de 3 pacientes realizando 3 tareas distintas cada 15 segundos (imaginar movimientos con la mano izquierda, con la derecha, y generación de palabras dada una letra). Entonces, se nos dan vectores de datos con 96 predictores distintos obtenidos de diferentes señales cerebrales medidas durante la realización del experimento. Nuestro objetivo es predecir una tarea dado un vector de entrada cada 0.5 segundos. Como los vectores de entrada que nos dan han sido calculados 16 veces por segundo, se nos pide explícitamente calcular la media de cada 8 vectores para crear uno nuevo, así que, nuestros datos después de este proceso corresponden a una predicción cada 0.5 segundos, o visto de otra manera, debemos generar una predicción por cada 8 vectores de entrada originales que nos lleguen.

Los mejores resultados de la competición [2] fueron obtenidos por un grupo de investigación de la Universidad de Barcelona [3], consiguiendo en los datos de test una precisión de 68.65 (sujeto 1 : 79.60, sujeto 2: 70.31, sujeto3 : 56.02), mediante Análisis Discriminante de Fisher y mediante un sistema para considerar las inconsistencias y transiciones entre tareas de los sujetos.

Para la implementación del trabajo, hemos usado Python3 junto con la libreria scikit-learn [6], debido a su facilidad de uso y a la trabajada documentación de la que dispone.

2. Tratamiento de los datos

Disponemos de unos 10000 datos por sujeto ordenados en el tiempo, sin valores nulos o no definidos. Primero de todo vamos a calcular la media cada 8 observaciones seguidas (cada grupo de 8 observaciones forma parte de la misma clase y el número de observaciones es múltiplo de 8) para obtener los vectores característicos con los que vamos a trabajar, que nos aportan información cada 0.5 segundos. Con este cambio pasamos de un total de 31216 datos a un nuevo conjunto de 3902, que vamos a tratar y observar a continuación.

2.1. Corregir 'skewness' y estandarizar datos

Todos los predictores de los vectores de los datos muestran una distribución como la de la figura 1, donde podemos observar un claro 'skewness' (o asimetría estadística). En la 2a imagen se muestran 3 predictores juntos para apreciar la diferencia de rangos que puede existir entre estos, pudiendo afectar negativamente a algunos modelos.

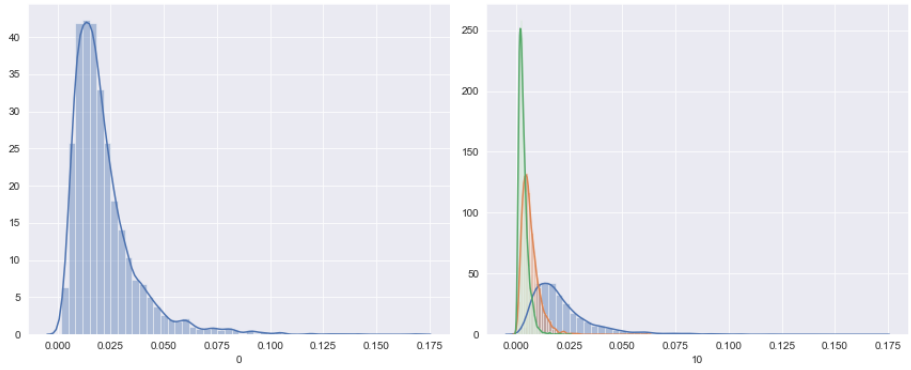

Figura 1: Distribución de los predictores

Para eliminar la falta de simetría de la distribución aplicaremos el logaritmo sobre los predictores, y los estandarizaremos(substrayendo la media y dividiendo entre la desviación estándar) para que el rango de valores de cada uno no influya negativamente en las predicciones. Podemos ver en la figura 2 el resultado obtenido.

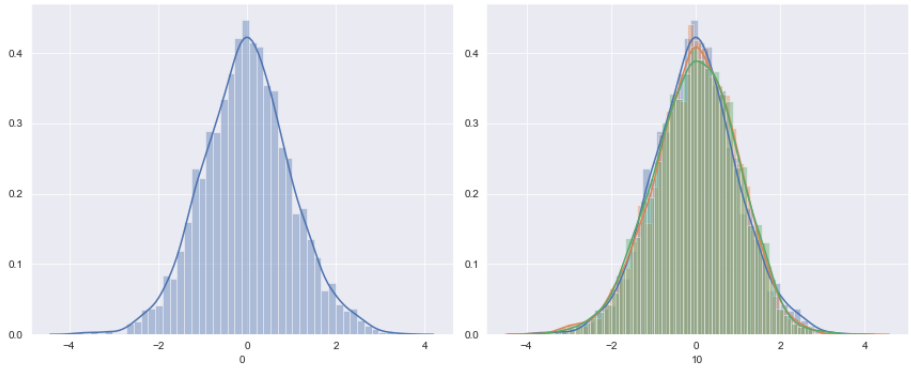

Figura 2: Distribución de los predictores después de aplicar log y estandarizarlos

2.2. Visualización con LDA

Aplicamos Linear Discriminant Analysis(LDA) para poder visualizar los datos y poder hacernos una idea inicial de la complejidad del problema, siempre recordando la precisión que se pierde teniendo en cuenta que se reduce un problema de 96 variables a una representación bidimensional.

Figura 3: Visualización de los datos con LDA

Podemos observar los datos de las 3 clases agrupados en 3 grupos, aunque bastante solapados entre si, esta característica nos hace pensar que va a ser difícil obtener buenas predicciones, por lo menos con modelos lineales.

2.3. Reducción de dimensionalidad con PCA

Debido a la dimensionalidad del problema (96 predictores), usaremos PCA para intentar eliminar algunos predictores. Si nos fijamos en la figura 4 podemos ver como los 2 primeros componentes no explican mucha de la varianza del target, en concreto explican un ratio de 0.21 y 0.06 (3) de la varianza.

Aun así, al seleccionar los componentes que llegan a explicar un 0.95 de la varianza, reducimos la dimensionalidad del problema de 96 a 65. Si no hubiésemos aplicado log a los datos, la varianza habría estado un poco más distribuida, ya que habríamos necesitado 66 predictores en vez de 65.

Figura 4: Visualización de los datos con PCA

3. Método de resampling

Partimos de un conjunto de datos de entrenamiento, y otro de test fijado por la competición. El tamaño de estos es de 3902 y 1308 respectivamente, una vez calculada la media de cada 8 vectores.

Para la selección de modelo/ajuste de hiper-parámetros hemos usado cross-validation con stratified 10-fold. Entonces, para cada modelo separamos los datos de entrenamiento en 10 grupos de tal manera que cada uno tenga un % similar de apariciones de la misma clase, y este se entrena 10 veces cada vez con 9 grupos diferentes y se evaluará con el grupo restante. A partir de de esta evaluación seleccionaremos el mejor modelo (tanto ajustar hiper-parámetros como comparar métodos distintos).

Mientras que la transformación 'log' de los datos la hacemos antes del resampling, ya que es independiente de otros datos, la selección de predictores con PCA y la estandarización debemos aplicarla dentro del cross-validate, en función de cada uno de los conjuntos de entrenamiento interiores, ya que sino, en cada iteración de cross-validate se habrían usado los datos de test (de las iteraciones de cross-validate) en el entrenamiento del modelo, alterando la estimación del error tal como se explica en la página 245 de 'The Elements of Statistical Learning' [4].

Una vez seleccionado el modelo final, evaluaremos su capacidad de predicción con los datos de test separados inicialmente, de manera que podamos obtener una estimación honesta y fiable de esta, ya que este conjunto de datos no ha sido usado en nada.

4. Modelización

A continuación vamos a presentar los modelos escogidos, junto con sus hiperparámetros finales seleccionados(si tienen alguno) y los resultados obtenidos usando cross-validation, que serán observados más a fondo una vez vistos todos los modelos, para seleccionar el definitivo de entre los distintos métodos.

Para facilitarnos la faena y abstraer las decisiones de cada modelo a la implementación, hemos definido una función execute(pipeline, grid) que recibe un pipeline de sklearn con el método de entrenamiento, y opcionalmente el conjunto de parámetros a explorar. Esta función implementa el método de resampling comentado. A la vez, también se permite guardar/cargar los modelos, y proporcionamos los ficheros de estos para ser cargados y probados sin tener que ejecutar el entrenamiento de nuevo.

4.1. Modelos lineales/cuadráticos

Los modelos seleccionados han sido: Naïve-Bayes, K-Neighbors y la red MLP single-layer.

4.1.1. Naïve-Bayes

Naïve-Bayes es un modelo que no tiene hiperparámetros dado que se basa en la teoría de Bayes de la probabilidad condicionada, y todos los parámetros necesarios para estimar mediante este modelo se calculan con los datos introducidos.

Los resultados de cross-validation obtenidos fueron los siguientes:

mean_test_score	std_test_score	mean_fit_time	std_fit_time	mean_score_time	std_score_time	model
0.507642	0.099283	0.075007	0.008971	0.002299	0.000483	Naive Bayes

Figura 5: Resultados Naïve-Bayes

Tiempo total ejecución cv: 0.92 seg.

4.1.2. K-neighbours

En el caso de K-neighbors hemos ajustado el hiper-parámetro número de vecinos entre 5 y 25 y el tipo de distancia a usar, entre manhattan y euclidea.

Hiper-parámetros seleccionados:

1. Número de vecinos = 22
2. Distancia = Euclidea

Resultados de cross-validation de estos hiper-parámetros:
Tiempo total ejecución cv: 13.45 seg.

mean_test_score	std_test_score	mean_fit_time	std_fit_time	mean_score_time	std_score_time	model
0.547347	0.114508	0.113435	0.009431	0.215276	0.02887	KNeighbors

Figura 6: Resultados K-neighbours

4.1.3. MLP Single-Layer

Para MLP hemos decidido explorar diferentes funciones de activación(relu, tanh, logistic), alphas entre 0.01 y 0.2, neuronas en 4 y 20 (reducimos bastante el número de neuronas a explorar debido al alto coste computacional que supone en la cpu). Usamos el solver por defecto de sklearn 'adam', que es el que recomiendan para datasets grandes(1000 datos o más) y asi podemos dedicar más tiempo a explorar los otros parámetros.

Los hiperparámetros seleccionados han sido:

1. Activation = 'relu'
2. Alpha = 0.13
3. Hidden Layer Size = 4

Resultados de cross-validation de estos hiper-parámetros:

mean_test_score	std_test_score	mean_fit_time	std_fit_time	mean_score_time	std_score_time	model
0.561976	0.103331	4.161161	1.579064	0.00246	0.004477	MLP

Figura 7: Resultados MLP Single-Layer

Tiempo total ejecución cv: 37 minutos.

4.2. Modelos no lineales

Los 2 modelos no lineales que hemos elegido son: SVMRBF (Support Vector Machines RBF) y Random Forest.

4.2.1. SVM-RBF

En el caso de las Support Vector Machines con kernel radial, hemos explorado para la selección de hiperparámetros tanto C :[0.001,100] como gamma: $2^{[-7,4]}$, (0.0078,0.0125, 0.25, 0.5, 1 ... 16).

Hiper-parámetros seleccionados:

1. C = 5.264
2. Gamma = 0.031

Resultados de cross-validation de estos hiper-parámetros:
Tiempo total ejecución cv: 7.68 minutos.

mean_test_score	std_test_score	mean_fit_time	std_fit_time	mean_score_time	std_score_time	model
0.57042	0.100677	1.89018	0.035622	0.132781	0.00781	SVM RBF

Figura 8: Resultados SVM-RBF

4.2.2. Random Forest

Random Forest es un algoritmo muy versátil, que se adapta a la gran mayoría de problemas, sobretodo de clasificación, pero tiene algunos problemas cuando hay muchas variables de poca relevancia "noise", como se explica en la página 596 de 'The Elements of Statistical Learning' [4]. Aún así, la proporción de ruido debe ser muy alta y si el modelo se comportase bastante mal ya lo observaremos en los resultados de la validación cruzada. En este caso hemos decidido no aplicar el preproceso con PCA, y que se encargue Random Forest de seleccionar las variables.

Los hiper-parámetros a ajustar han sido: número de árboles(500,1000,1500) y el número de predictores a tener en cuenta para las decisiones ($\sqrt{n}$, 25, 30, 35, 40, 45, 50) .

Hiper-parámetros seleccionados:

1. Número de árboles = 500
2. Número de variables/predictores = 35

Resultados de cross-validation de estos hiper-parámetros:

mean_test_score	std_test_score	mean_fit_time	std_fit_time	mean_score_time	std_score_time	model
0.606292	0.111367	46.562545	8.216287	9.295965	2.085388	Random Forest

Figura 9: Resultados Random Forest

Tiempo total ejecución cv: 28 minutos.

5. Modelo definitivo

A la hora de seleccionar el modelo final, vamos a comparar los resultados que hemos obtenido en la validación cruzada. Para ello será importante no solo comparar cuales son las medias de precisión obtenidas, sino también su varianza, ya que a veces puede ser más importante garantizar unos mínimos superiores, un modelo que nos garantice ser más estable aunque la media sea inferior. También observaremos los tiempos de los datos tanto de entrenamiento como los de predicción, siendo muy importantes estos últimos si nuestro modelo tiene que hacer predicciones en tiempo real, como es nuestro caso.

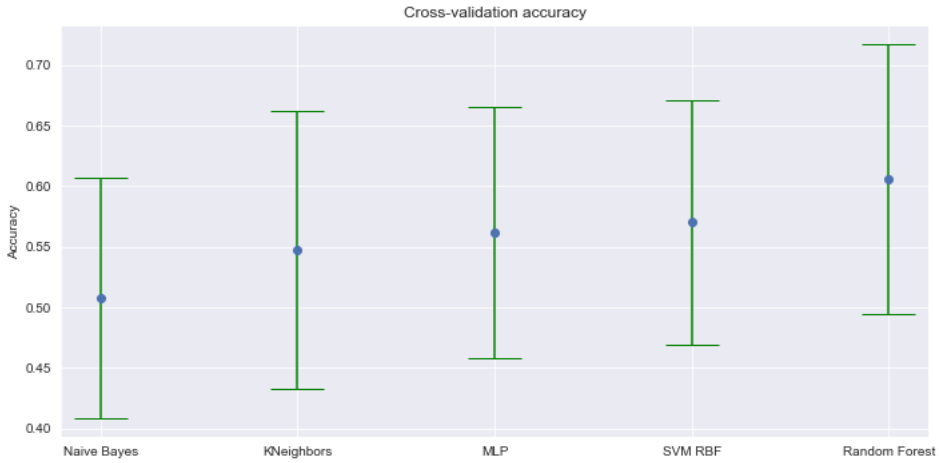

Figura 10: Precisión obtenida en las iteraciones de cross-validation.

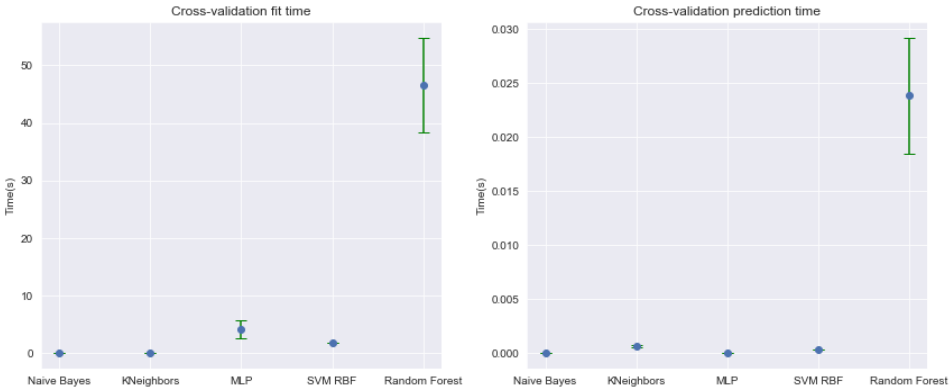

Figura 11: Tiempos de entrenamiento y predicción.

- Claramente Random Forest es el modelo con una precisión mayor en media, y a la vez con unos valores mínimos superiores al resto .
- Aunque Random Forest tiene un tiempo de entrenamiento mayor, debemos tener en cuenta que su espacio de búsqueda de hiper-parámetros no es tan grande como por ejemplo MLP, y no es lo mismo entrenar un random forest de 500 árboles, que una red de solo 20 neuronas.
- Por contra, Random Forest tiene unos costes de predicción muy altos, aunque en nuestro caso es suficiente ya que debemos hacer una predicción cada 0.5 seg.

Debido a los motivos citados previamente, hemos seleccionado como modelo definitivo Random Forest, en la figura 12 podemos ver los resultados finales obtenidos con los datos de test (únicamente usado con este propósito), incluyendo cada sujeto individualmente, y en la figura 13 los resultados de los 10 primeros puestos de la competición.

```
Test accuracy:
        subject1 ->  0.8127853881278538
        subject2 ->  0.6751152073732719
        subject3 ->  0.5137614678899083
        GENERAL  ->  0.6674311926605505
```

Figura 12: Resultados finales de nuestro modelo.

#.	contributor	psd	acc	s1	s2	s3	research lab
1.	Ferran Galan	y	**68.65**	79.60	70.31	56.02	University of Barcelona
2.	Xiang Liao	y	**68.50**	78.08	71.66	55.73	University of Electronic Science and Technology of China (UESTC)
3.	Walter	y	**65.90**	77.85	66.36	53.44	???
4.	Xiaomei Pei	y	**65.67**	76.03	69.36	51.61	Institute of Biomedical Engineering of Xi'an Jiaotong University
5.	Irene Sturm	y	**64.91**	78.08	63.83	52.75	Fraunhofer FIRST (IDA), Berlin
6.	Stephan Uray	y	**64.60**	81.05	73.04	39.68	TU Graz
7.	Julien Kronegg	y	**64.04**	76.06	64.83	51.18	University of Geneva
8.	John Q. Gan	y	**63.91**	77.40	63.83	50.46	University of Essex, Colchester
9.	Shiliang Sun	n	**62.83**	74.31	62.32	51.99	Tsinghua University, Beijing
10.	J. Ignacio Serrano M. D. del Castillo	y	**62.61**	75.80	61.75	50.23	Instituto de Automatica Industrial. CSIC. Madrid

Figura 13: Resultados finales de la competición.

6. Conclusiones/reflexiones

Con los resultados observados, podemos observar como los datos que tenemos entre manos seguramente son de naturaleza no lineal, ya que se ven favorecidos por modelos no lineales.

Es interesante ver la diferencia en el acierto de predicciones entre sujetos, que puede ser debido tanto al estado de concentración de los sujetos durante las pruebas, a las propias diferencias biológicas o a la interpretación/representación diferente de la información que hacen estos, en función de sus vivencias, experiencia, etc.

Todo parece indicar que se podrían hacer modelos personalizados para cada sujeto con un buen rendimiento, pero obtener un modelo general que de respuesta a todas las personas con un acierto alto es otro tema.

Mientras que estamos satisfechos con el trabajo realizado y los resultados obtenidos, hemos sentido como muchas veces nos faltaba conocimiento para llevar a cabo una decisión, sin saber si esta era la óptima.

A la vez somos conscientes que no hemos sacado máximo provecho de por ejemplo un método como el Multilayer Perceptrón (MLP), debido a que lo hemos

limitado a 1 capa y pocas neuronas debido al alto coste computacional que supone entrenarlo y explorar sus hiper-parámetros si no se usa la GPU para su cálculo.

También, los datos tratados son una serie temporal, ya que están ordenados en el tiempo, y existen métodos [7] para tratar esta característica y sacarle provecho.

Referencias

[1] *BCI competition dataset V description.* URL: `http://www.bbci.de/competition/iii/desc_V.html`.

[2] *BCI competition dataset V results.* URL: `http://www.bbci.de/competition/iii/results/index.html`.

[3] Joan Guàrdia Ferran Galán Francesc Oliva. *BCI Competition III. Data Set V: Algorithm Description.* URL: `http://www.bbci.de/competition/iii/results/martigny/FerranGalan_desc.pdf`.

[4] Trevor Hastie, Robert Tibshirani y Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction.* 2.ª ed. Springer, 2009. URL: `http://www-stat.stanford.edu/~tibs/ElemStatLearn/`.

[5] J. del R Millán. "On the need for on-line learning in brain-computer interfaces". En: (Proc. Int. Joint Conf. on Neural Networks., 2004.).

[6] F. Pedregosa y col. "Scikit-learn: Machine Learning in Python". En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830.

[7] *Time Series - Wikipedia.* URL: `https://en.wikipedia.org/wiki/Time_series#Classification`.