Clasificación de alimentos mediante la detección de estímulos aplicando técnicas de aprendizaje automático

Luis Fernández Freire
Raúl Martín Hernández
Alejandro Benimeli Miranda
Rodrigo Juez Hernández

Abstract. Este estudio investiga un conjunto de sensores de gas que generan datos en forma de series de tiempo en las cuales deberemos evaluar cuando se introduce un estímulo y clasificar dicho estímulo entre las clases plátano, vino y ruido de fondo. Los datos se han tratado y modificado para adaptarlos al problema de clasificación. Para la clasificación se han probado diversos métodos de machine learning tradicionales usando optimización de hiperparámetros. Nuestro análisis ha concluido que el clasificador más robusto es Random Forest con una precisión de 0.9962 ± 0.0005, y es también al que menos afecta reducir el tamaño del dataset.

1 Introducción

El problema que vamos a afrontar en este documento consiste en clasificar instantes de tiempo entre tres clases: vino, plátano y ruido de fondo. Los datos proporcionados consisten en series de tiempo de 8 sensores de tipo MOX, además de la temperatura y humedad. Los sensores están colocados dentro de una habitación y miden cuando se introduce uno de los dos objetos mencionados, o cuando no hay nada y entonces se considera ruido de fondo. Los objetos se introducen siempre una hora después de comenzar el experimento, pero pueden ser eliminados en cualquier momento.

El dataset [1] esta contenido en dos archivos .dat, uno de datos, que contiene todas las mediciones de todas las instancias de tiempo, y otro de metadatos, en el que se indica a qué clase pertenece cada serie, y durante cuánto tiempo se introdujo el alimento. Ambos datasets se relacionan mediante una id, donde cada serie tiene un conjunto de instantes de tiempo asociados.

2 Análisis exploratorio de los datos

Antes de empezar a crear atributos y seleccionar los más importantes vamos a analizar los datos que se han extraído en la fase de experimentación. Para empezar, en los datos contamos con las 3 clases mencionadas anteriormente, que son Plátano, Vino o Ruido (Banana, Wine o Background en inglés).

Para cada clase hay un número diferente de series, que se puede ver en la Figura 1. Para cada una de estas clases hay una proporción diferente de muestras de

tiempo, visible en la Figura 2. Este desbalance se debe a que en cada serie donde se introduce plátano o vino, hay un tiempo antes y después donde la muestra no está presente, por lo que hemos considerado que dichas instancias de tiempo deben ser de la clase Ruido. Hemos tenido esto en cuenta en la Sección 3, donde se trata el preprocesado de datos.

Hemos encontrado que aunque en el fichero metadata figure una serie número 95, no existen instancias de tiempo asociadas a dicha serie en el fichero dataset. Esto se ha tenido en cuenta a la hora de generar la Figura 1.

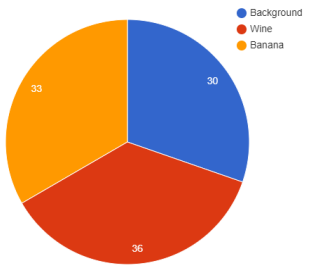

Fig. 1: Número de series de cada clase.

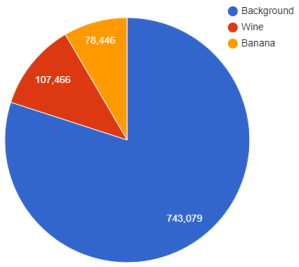

Fig. 2: Número de instantes de tiempo de cada clase.

Hemos llevado a cabo un análisis estadístico de los valores recogidos por cada sensor, y hemos conseguido unas gráficas que nos han dado una información muy relevante, presentes en la Figura 3. En estas gráficas podemos ver cómo todos los sensores tienen un comportamiento razonable dentro de un umbral. En cambio el sensor 5 tiene unos extraños picos que seguramente se deban a que es defectuoso, y podrían afectar a la tarea de clasificación.

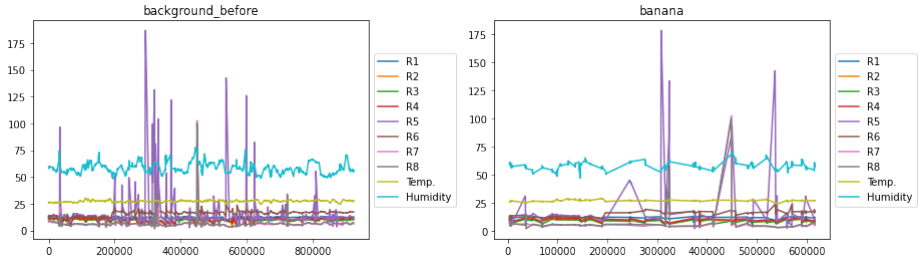

Fig. 3: Valores de cada sensor por cada instancia de tiempo.

Gracias a los datos del experimento anterior, hemos calculado estadísticas para cada clase, por cada sensor. En la Figura 4 podemos ver las medias y varianzas de cada sensor para cada clase, generada con matplotlib [2]. Es importante remarcar que se han anotado dos clases más, en las que se consideran diferentes

tipos de riudo antes y después de meter los estímulos. Esto es así ya que en algunas observaciones hemos percibido que después de insertar el estímulo algún tipo de residuo afecta a los sensores. Esto se puede observar en la Figura 4, en la que en ciertos sensores la varianza de background es menor a la de background_after.

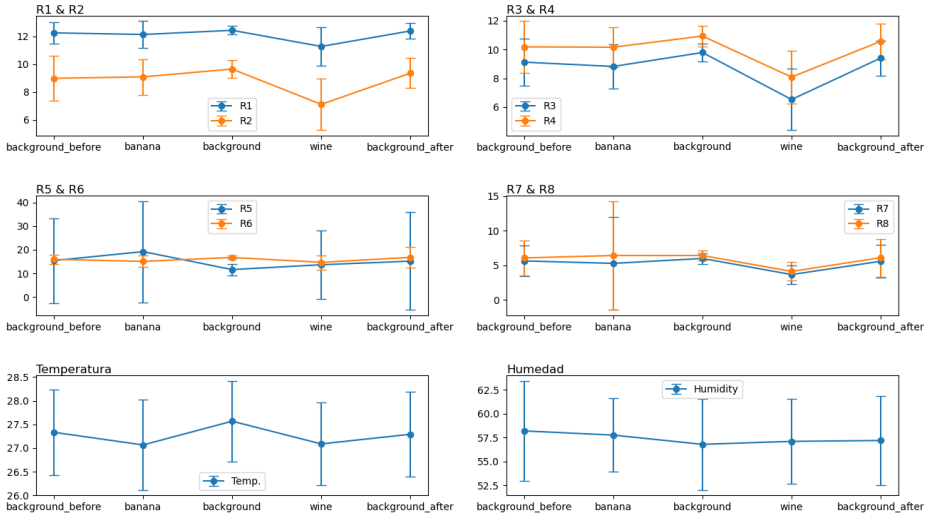

Fig. 4: Medias y varianzas de cada sensor.

Por último podemos remarcar que este análisis resulta útil sobre todo para fijarnos en que existe algún tipo de correlación entre algunos sensores de tipo MOX, algo en lo que indagamos con más profundidad en la Sección 3.2.

3 Preprocesado de datos y propuestas de nuevos atributos

Antes de añadir o quitar atributos hemos generado el dataset principal en el que cada instante de tiempo tiene asociada una clase, juntando la información presente en ambos datasets. Hemos considerado que las instancias de tiempo antes de introducir la muestra y después de retirarla son background. Esto hace que el dataset quede muy desbalanceado, por lo que se hace un muestreo aleatorio para que las clases tengan el mismo número de filas a la hora de clasificar.

Una vez hecho esto, hemos adaptado el dataset para poder clasificar instantes de tiempo, aprovechando la técnica de ventanas de tiempo. Estas ventanas constan de al menos 3 muestras de tiempo anteriores a cada instancia a evaluar, y dentro de un margen de tiempo que, después de una rápida experimentación hemos determinado que debe ser de 3 minutos. Las ventanas que no cumplen estas condiciones se establecen como NaN, y posteriormente se eliminan, junto

con las columnas de tiempo e id. Todo el procesado de datos se ha realizado usando las librerías Numpy [3] y Pandas [4].

3.1 Nuevos atributos

Gracias a los resultados observados en la Figura 4, hemos considerado que añadir los siguientes atributos estadísticos facilitarían la tarea de clasificación. Para cada sensor hemos añadido los siguientes atributos:

- Media: Este atributo se utiliza para representar el valor medio que ha tenido la ventana durante los últimos minutos.
- Desviación Típica: Este valor representa la variación le las mediciones en la ventana de tiempo ya que, como se puede ver en la Figura 5 (generada gracias al código extraído de [5]), cuando se introduce la muestra de vino la varianza de las mediciones se ve incrementada.
- Min y Max: Estos valores apoyan a la desviación típica para indicar el umbral que ha alcanzado la ventana.
- Pendiente: Este atributo resulta de alta importancia ya que consideramos que las ventanas que se encuentran en los cambios entre background y muestras de alimentos pertenecen a la clase del alimento.

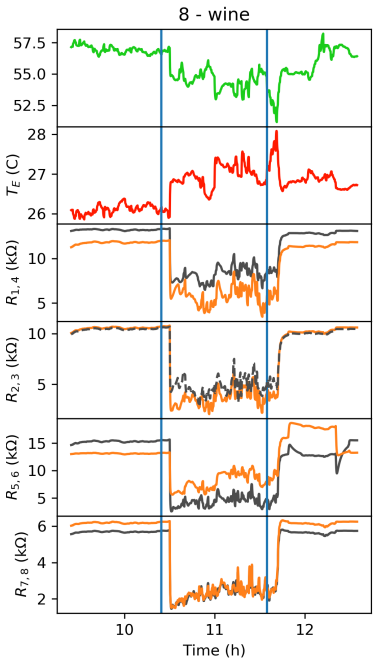

Fig. 5: Visualización de la serie 8 por cada sensor.

3.2 Eliminación de atributos

Añadir 5 atributos por cada sensor resulta en unos 60 atributos. Esto puede resultar problemático en lo que a tiempo de ejecución respecta, y por ello hemos reducido la dimensionalidad gracias a las conclusiones extraídas en las secciones anteriores.

En la Sección 2 hemos visto que el sensor 5 está defectuoso, por lo que hemos eliminado todos sus atributos (R5, R5mean, R5std...).
Una vez hecho esto, vamos a probar a visualizar la matriz de correlación de todos los atributos. Esta información está presente en la Figura 6. Aquí los píxeles

con valores muy cercanos a 1 o -1 indican una alta correlación entre atributos, lo que significa que éstos son redundantes entre sí.

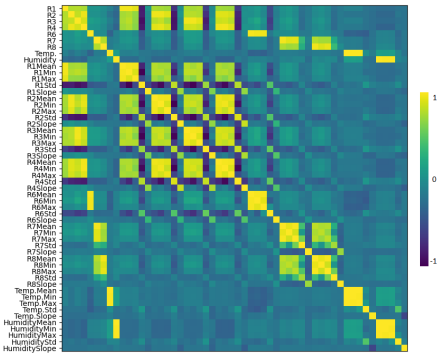

Fig. 6: Matriz de correlación antes de borrar atributos.

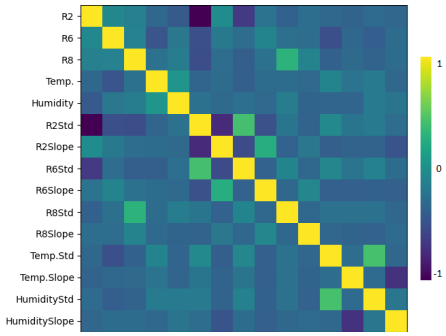

Fig. 7: Matriz de correlación después de borrar atributos redundantes.

Podemos ver que los sensores R1, R2, R3 y R4 están muy fuertemente correlacionados. Lo mismo ocurre entre los sensores R7 y R8. Por ello, hemos elegido quedarnos únicamente con los sensores R2, R6 y R8. Existen otras redundancias entre algunos de los atributos estadísticos introducidos y sus respectivos sensores. En particular podemos fijarnos que cada sensor es redundante con sus atributos de Mean, Min y Max, por lo que se van a eliminar del dataset. Estos arreglos resultan en una matriz de correlación ideal (Figura 7), en la que las únicas correlaciones altas son entre un atributo y él mismo.

Haber pasado de 54 atributos a 15 favorece entrenamientos más efectivos y reduce considerablemente los tiempos de ejecución para cualquier clasificador a probar. La eliminación de estos atributos no afecta a la precisión de los clasificadores ya que la información que pueden extraer de ellos es redundante. Los resultados se pueden ver en la Sección 5 y su análisis en la Sección 6.

4 Modelos utilizados, descripción del protocolo experimental y estimación de parámetros

Hemos usado los modelos estudiados en clase de teoría: Naive Bayes [6], K Nearest Neighbors [7], Logistic Regression [8], Red Neuronal [9] y Random Forest [10] de SKLearn [11]. Los escogimos tanto por estar familiarizados con ellos, como porque son capaces de tratar con problemas multiclase (excepto la regresión lineal, pero SKLearn. tiene un parámetro que permite usar el algoritmo 'one vs rest' para hacer una clasificación multiclase). De los diferentes tipos de Naive Bayes que existen escogimos el gausiano, ya que todos los atributos de los datos que vamos a tratar son continuos.

El protocolo experimental usado ha consistido en probar en cada clasificador sus hiperparámetros más importantes:

- Naive Bayes: Ninguno.
- KNN: Número de vecinos.
- Logistic Regresion: Ninguno.
- Red Neuronal: Número de capas ocultas (1-2) y neuronas por capa (10-40). Por simplicidad siempre usamos el mismo número de neuronas en ambas capas.
- Random Forest: Número de estimadores.

Para el cálculo de la precisión se utiliza la función cross_val_score de SKLearn para hacer una validación cruzada de 5 iteraciones. Esto se realiza para evitar el sobreajuste. Cross_val_score se encarga de hacer la separación de los datos de entrenamiento y de prueba en cada iteración automáticamente. Para los clasificadores para los que es necesario (o aconsejable) normalizar los datos, usamos una pipeline de SKLearn que normaliza las particiones de datos de entrenamiento y prueba en cada iteración de la validación cruzada antes de clasificar.

Por último, para acelerar el tiempo de ejecución del fichero que prueba los parámetros usamos el módulo Multiprocessing de Python [12] para distribuir la carga entre varios núcleos.

5 Resultados obtenidos

Gauss. Naive Bayes	
Accur.	Desv
0.565277	0.002953

Regresión Logística	
Accur.	Desv
0.678291	0.001224

KNN		
K	Accur.	Desv
3	0.9863	0.000628
8	0.9739	0.000723
13	0.9601	0.000957
18	0.9542	0.000892
38	0.9313	0.001110

Red Neuronal			
Capas	Neuronas	Accur.	Desv
1	10	0.813982	0.004911
	20	0.863128	0.009472
	30	0.892082	0.005235
	40	0.912408	0.003810
2	10	0.864355	0.009236
	20	0.946296	0.002155
	30	0.969213	0.003137
	40	0.979955	0.001634

Random Forest		
N	Accur.	Desv
10	0.995373	0.000482
30	0.995975	0.000407
70	0.996090	0.000383
90	0.996182	0.000313
110	0.996233	0.000522
130	0.996063	0.000412
170	0.996154	0.000454

Table 1: Precisiones y desviaciones típicas de los experimentos de ajuste de hiperparámetros.

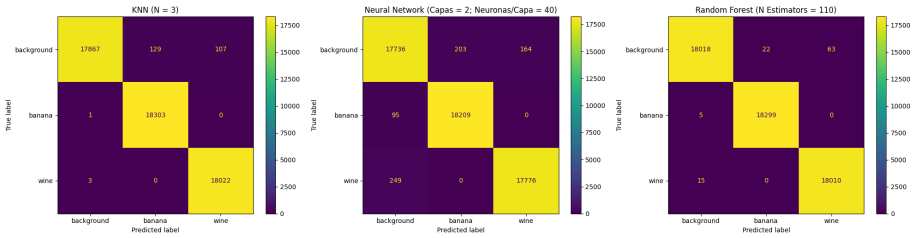

Fig. 8: Matrices de confusión de los 3 mejores clasificadores.

KNN	
K	Accur.
3	0.948
8	0.921
13	0.901
28	0.869

Red Neuronal		
Capas	Neuronas	Accur.
1	20	0.846
	40	0.887
2	20	0.915
	40	0.952

Random Forest	
N	Accur.
10	0.971
30	0.973
110	0.974
170	0.974

Table 2: Precisiones de los mejores clasificadores usando 1/10 de los datos.

En las Tablas 1 y 2, y en la Figura 8, tenemos los resultados de los experimentos realizados.

6 Discusión de los resultados obtenidos y conclusiones

Como se puede ver en la tabla 1, los 3 clasificadores que mejor precisión consiguieron (independientemente de sus hiperparámetros), fueron los clasificadores KNN, Neural Network y Random Forest. Las tablas expuestas son los algoritmos más representativos de los que probamos, hicimos más pruebas pero no son relevantes y ocuparían espacio de más.

Viendo todos los clasificadores y como varía la precisión según su funcionamiento o la variación de los hiperparámetros deja entrever que nuestro dataset requiere de un modelo muy complejo con muchos parámetros y una frontera de decisión que dista mucho de ser simple o necesitar regularización.

La variación de la precisión cuando vamos aumentando la cantidad de vecinos en KNN confirma nuestra hipótesis, cuantos más vecinos, la frontera de decisión se simplifica (hace como un efecto de regularización). Eso quiere decir que 3 vecinos es el más apropiado porque permite un modelo mucho más complejo.

Logistic Regression requiere que los datos sean linealmente separables, esa es la razón por la que se comporta tan erráticamente. Naive Bayes, aun que no separa los datos linealmente también genera un modelo demasiado simple.

El mejor resultado en las redes neuronales se obtiene con la configuración más compleja probada, con 2 capas y 40 neuronas por capa.

El caso de Random Forest es diferente. Aunque el que mejor rendimiento da es el que tiene 110 estimadores, la diferencia al variar el hiperparámetro es despreciable y podría ser debida a la aleatoriedad del algoritmo. Un árbol de decisión ya de por si puede generar una frontera de decisión muy compleja, pero si esto lo combinamos con que en Random Forest los árboles votan usando bagging y se contrarrestan los errores entre ellos tenemos que es el que mejor resultado da repetidamente. Si nos fijamos en la Figura 8 vemos cómo, además de ser el que mejor precisión obtiene, es muy bueno distinguiendo entre vino y plátano y se comporta mejor que la red neuronal.

Como ya hemos dicho, este dataset requiere de modelos muy complejos, pero aún así nos parece que obtenemos un rendimiento más que aceptable y es difícil imaginar una situación en la que Random Forest no sea el mejor, ya que es rápido, prácticamente no requiere de ajuste de hiperparámetros y obtiene resultados muy fiables.

KNN también es un buen clasificador con 3 vecinos. En cambio, su complejidad computacional es mucho mayor a Random Forest. Si quisiéramos construir un sistema que detecte en tiempo real qué alimento se encuentra en la sala, el uso de KNN no es adecuado por su elevado tiempo de ejecución. En su favor hay que decir que viendo la matriz de confusión (Figura 8), es el algoritmo que menos confunde los objetos a detectar (vino y plátano). KNN tiene un mayor error clasificando erróneamente el ruido de fondo, pero si le quisiéramos dar prioridad a detectar cualquier tipo de alimento frente a diferenciarlos entre sí, podría interesarnos más KNN. Aún así opinamos que un Random Forest es más adecuado para el problema que queremos solucionar.

El dataset que usamos tiene una cantidad muy elevada de datos, por lo que hemos comprobado cómo se comportarían los modelos elegidos con muchos menos datos. Ejecutando los mismos experimentos para 1/10 de los datos (salvo algunos hiperparámetros que inferimos que no van a rendir muy bien vistos los resultados anteriores), obtenemos las precisiones de la Tabla 2. Como se puede observar, Random Forest sigue dominando sobre el resto. De hecho, es el algoritmo que menos perjudicado se ve por la drástica reducción del volumen de los datos. En cuanto a las matrices de confusión, observamos que se mantiene la misma tendencia salvo que ahora Random Forest es también el mejor diferenciando entre Vino y Plátano, nulificando la única ventaja que KNN tenía sobre este.

En conclusión, vistos los resultados de la Sección 5 y el análisis de los mismos realizado en esta, el algoritmo que utilizaríamos para clasificar instantes de tiempo es Random Forest, con un número de estimadores igual a 110.

References

[1] F. Huerta, R. Huerta, T. Mosqueiro, J. Fonollosa, N. Rulkov, and I. Rodriguez-Lujan. "gas sensors for home activity monitoring data set", Jul. 2016. `https://archive.ics.uci.edu/ml/datasets/Gas+sensors+for+home+activity+monitoring`, Accedida 10/12/2021.

[2] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[3] C.R. Harris, K.J. Millman, and S.J. et al van der Walt. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[4] The pandas development team. pandas-dev/pandas: Pandas, February 2020. `https://doi.org/10.5281/zenodo.3509134`.

[5] R. Huerta, T. Mosqueiro, J. Fonollosa, N. Rulkov, and I. Rodriguez-Lujan. "online decorrelation of humidity and temperature in chemical sensors for continuous monitoring", Jul. 2016. `https://github.com/thmosqueiro/ENose-Decorr_Humdt_Temp`, Accedida 10/12/2021.

[6] David Hand and Keming Yu. Idiot's bayes: Not so stupid after all? *International Statistical Review*, 69:385 – 398, 05 2007.

[7] G Ritter, H Woodruff, S Lowry, and T Isenhour. An algorithm for a selective nearest neighbor decision rule (corresp.). *IEEE Trans. Inf. Theory*, 21(6):665–669, November 1975.

[8] Antônio Alves, Antônio Fernandes, Dalson Britto, Dalson Figueiredo, Enivaldo Rocha, Willber Da, and Silva Nascimento. Read this paper if you want to learn logistic regression. *Revista de Sociologia e Política*, 28:1, 12 2020.

[9] Peter Zhang. Neural networks for classification: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 30:451 – 462, 12 2000.

[10] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] Python Software Foundation. Python multiprocessing library. `https://docs.python.org/3/library/multiprocessing.html`.