Proyecto final de Aprendizaje Automático

Alejandro Martín, Daniel Mohedano y Silvia Sopeña

Abstract. Usando distintos sensores de temperatura, humedad y 8 de resistencia trataremos el conjunto de datos usando distintas técnicas de aprendizaje automático. Con el objetivo final de tratar de predecir los diferentes estímulos de plátano y vino, con la mayor precisión que sea posible. En esta memoria se resumirá el proceso que se ha seguido para lograr el resultado final.

1 Introducción

El objetivo del problema es el siguiente: Estando los sensores constantemente recabando información, se deberán distinguir los diferentes estímulos (o ausencia de ellos) en cada instante de tiempo. Para ello se ha obtenido el conjunto de datos del *UCI Machine Learning Repository*[1].

2 Análisis exploratorio de los datos

En nuestro conjunto de datos tenemos 100 series de tiempo de experimentos, divididas entre las 3 clases: `plátano`, `vino` y `background`. Cada una de esas series se divide en 12 atributos diferentes: id del experimento, etiqueta temporal, 8 resistencias (R1-R8), temperatura y humedad.

Las series en las que figuran `plátano` y `vino` se estructuran de la siguiente manera: Medición de `background` antes del estímulo, medición del estímulo de la clase que corresponda (`plátano` o `vino`) y medición de `background` después del estímulo.

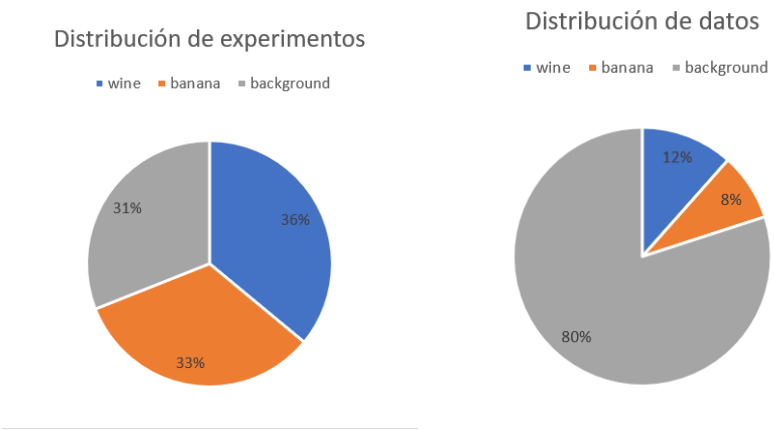

Fig. 1: Distribución del conjunto de datos

En total hay una cantidad de 928991 datos de instantes temporales. En la Figura 1 se representa tanto la distribución de las clases en los experimentos como la distribución de clases en los datos temporales.

Aunque los experimentos si que están bien equilibrados, claramente los datos no. El `background` es la medición más abundante de todas, haciendo que sea más complicada la clasificación de los estímulos de `plátano` y `vino` por el desbalance en el conjunto de datos. Esto se debe a que en todos los experimentos hay mediciones de `background` previas a la introducción del estímulo y posteriores a la extracción del mismo.

Para ayudar en la visualización de los datos también mostramos en figuras todos los datos de cada experimento, ayudándonos a llegar a algunas conclusiones al inspeccionar las gráficas [2]. Como se puede ver en la Figura 2, a simple vista parece que el `vino` genera mucho más ruido que el `plátano` al ser introducido. Este comportamiento fue observado en muchos de los otros experimentos.

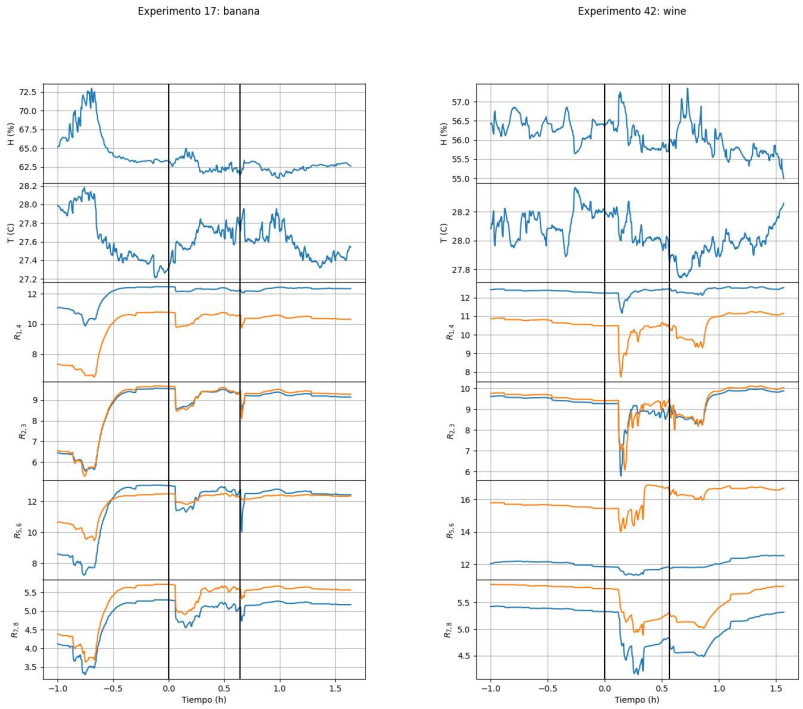

Fig. 2: Datos para experimento de `vino` y `plátano`.

Otro aspecto importante que observamos en las figuras fue la imprecisión de la medida `R5`. Esta parecía tener mucho ruido en comparación con el resto de sensores, sobretodo después del experimento 31, lo que podría indicar alguna

avería o algo similar. Esto se puede comprobar viendo la descripción de los atributos que se muestra en la Tabla 1 [3]. La desviación típica del sensor es claramente superior al resto de sensores.

	count	**mean**	**std**
R1	928991	12.185	0.868
R2	928991	8.958	1.558
R3	928991	8.945	1.748
R4	928991	10.130	1.711
R5	928991	15.154	18.391
R6	928991	16.052	3.303
R7	928991	5.390	2.889
R8	928991	5.912	3.304
Temp.	928991	27.283	0.904
Humidity	928991	57.568	4.821

Table 1: Descripción de los datos.

También hemos encontrado algunos datos no válidos. Del experimento 95 no hay datos temporales (no demasiado relevante puesto que según el metadata era simplemente ruido de `background`). Apreciamos también que en el experimento 47 hay algún tipo de error o efecto externo ya que algunos de los sensores tienen un pico mucho mayor de lo esperado. Y finalmente en el experimento 76 no hay datos temporales entre los límites indicados por el metadata.

3 Descripción de los atributos propuestos y su obtención

Una de las primeras decisiones que tomamos fue seleccionar solo los datos entre los límites de estímulo. Es decir, desde el instante de introducción del estímulo hasta la extracción. De esta forma el conjunto de datos estará mucho más equilibrado, a diferencia del original que como habíamos visto tenía una proporción mucho más grande de `background` que del resto de clases.

En segundo lugar, decidimos eliminar por completo los datos del experimento 47 por no parecer fiables. Finalmente, decidimos eliminar también el atributo `R5` ya no sería posible simplemente eliminarlo de las medidas temporales en las que empezaba a mostrar errores. También debatimos añadir algunos atributos extra pero decidimos en contra por varios motivos:

- El aumento considerable del tiempo de ejecución, lo que lo hacía casi inviable.
- Ya que el problema que intentamos solucionar es clasificar instantes de tiempo, no nos parecía del todo correcto añadir datos calculados a partir de instantes de tiempo previos. Se podría dar el caso de intentar clasificar un dato temporal del que se carece de datos previos, por lo que medidas que indiquen la tendencia de los datos no parecen razonables.

Una vez analizados los datos y escogidos cuales serían relevantes para el estudio, procedimos a elegir los modelos candidatos para la clasificación. Algo que se debe de tener en cuenta es que el problema de clasificación que estamos manejando es multiclase. Esto añade un nivel de complejidad al entrenamiento si el clasificador no está preparado para tratar con este tipo de problemas. Por este motivo, dimos preferencia a clasificadores que nativamente pudiesen manejar problemas de carácter multiclase [4]. También dimos preferencia a los modelos vistos en clase, puesto que poseemos un mayor conocimiento de su funcionamiento. A continuación se muestra el listado de clasificadores probados junto con las variaciones en sus hiperparámetros:

- **Naive-Bayes**: Para este modelo probamos las dos implementaciones que mejor nos funcionaron en la práctica, `MultinomialNB` [5] y `GaussianNB` [6].
- **K Vecinos Próximos**: Utilizando la implementación de `KNeighborsClassifier` [7] modificamos el hiperparámetro del número de vecinos.
- **Regresión logística**: Para este modelo se utilizaron dos implementaciones distintas. En primer lugar, `LogisticRegression` [8] con la opción `multi_class='multinomial'` para permitir tratar con problemas multiclase. En segundo lugar, también probamos la implementación `SGDClassifier` [9], que implementa regresión logística utilizando *Stochastic Gradient Descent*. Esta implementación no tiene soporte para problemas multiclase por lo que la solución fue aplicar un enfoque *OneVsRest* [10]. Para esta última se utilizó una configuración muy similar a la utilizada en la práctica. Se usó la función de pérdida logarítmica, y se modificó el hiperparámetro de la constante de aprendizaje desde 0.1 a 1.
- **Random Forest**: Uno de los modelos más complejos que utilizamos. Se basa en construir diversos árboles de decisión para luego hacer la media de los resultantes. La implementación que utilizamos fue `RandomForestClassifier` [11] y el único hiperparámetro que se modificó fue el del número de árboles en el forest.
- **Redes Neuronales**: Para este modelo utilizamos la implementación de `MLPClassifier` [12], o perceptrones multicapa. Las configuraciones utilizadas fueron de 1 y 2 capas variando la cantidad de neuronas por capa.

Aunque podríamos haber probado tanto más clasificadores como configuraciones de hiperparámetros para cada clasificador, los tiempos de ejecución empezaban a crecer ya exponencialmente, por lo que no era del todo asequible.

Por último, para realizar el proceso de entrenamiento y validación de cada uno de los modelos se utilizó validación cruzada, obteniendo de esta forma resultados más fiables. En concreto usamos validación cruzada de tipo *K-Fold*, puesto que la implementación es sencilla y tiene gran utilidad. Las pruebas se hicieron con una cantidad de divisiones de 5, el valor predeterminado en el constructor

de la clase. De nuevo, se podría haber probado con una mayor cantidad de divisiones pero esto habría causado un crecimiento en los tiempos de ejecución y además correría el riesgo de generar conjuntos de prueba demasiado pequeños, por lo que los resultados obtenidos no serían los más fiables. Puesto que los datos se encuentran en orden temporal en las tablas, utilizamos la opción de `shuffle=True` para evitar causar algún tipo de bias en los clasificadores. Aún así, nos aseguramos de siempre inicializar el `random_state` con el mismo valor para que todos los clasificadores utilicen siempre las mismas divisiones y la comparación entre ellos pueda ser justa [13].

4 Resultados obtenidos

En la Tabla 2 se muestran los resultados de la validación cruzada. Se muestran tanto los valores de la precisión media como de la desviación típica, para cada uno de los clasificadores utilizados.

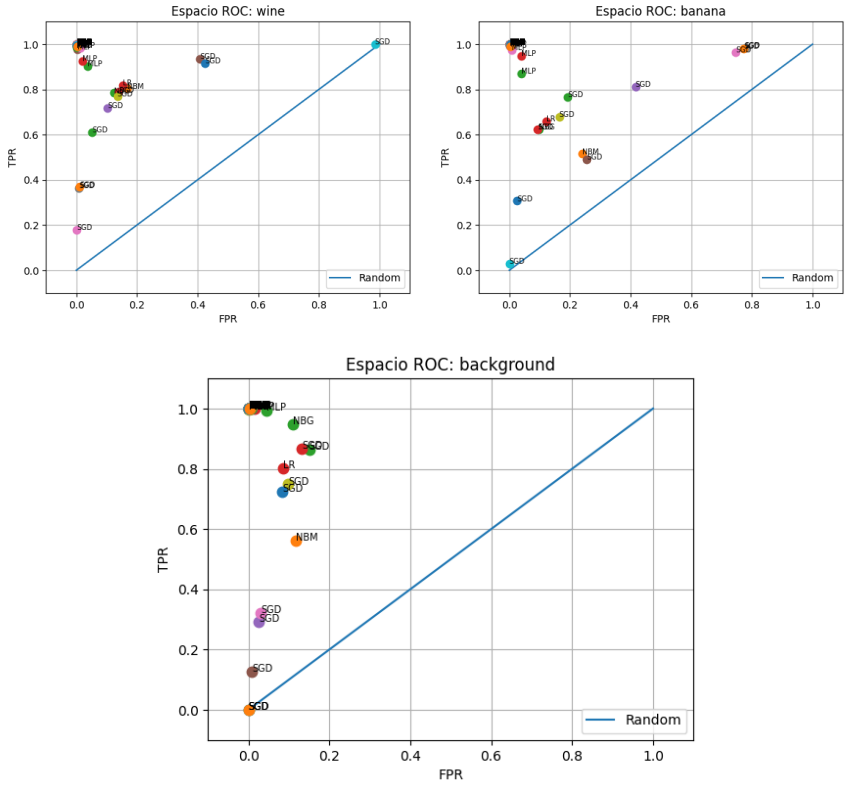

Fig. 3: Espacios ROC para cada clase.

En la Figura 3 se muestran los espacios ROC calculados para cada una de las clases. Estos fueron calculados con los resultados de la matriz de confusión

Naive Bayes		
Tipo	**Precisión**	**Desv.**
Multinomial	0.649884	0.002006
Gaussian	0.778738	0.001009

Random Forest		
N	**Precisión**	**Desv.**
10	0.999877	0.000038
35	0.999909	0.000063
60	0.999925	0.000044
85	0.999909	0.000046
110	0.999913	0.000048
135	0.999921	0.000050
160	0.999925	0.000044
185	0.999917	0.000049

KNN		
K	**Precisión**	**Desv.**
5	0.999885	0.000063
15	0.999663	0.000074
25	0.999437	0.000113
35	0.999290	0.000092
45	0.999124	0.000122
55	0.998854	0.000069
65	0.998553	0.000159
75	0.998378	0.000125
85	0.998105	0.000216
95	0.997867	0.000164

Regresión Logística		
	Precisión	**Desv.**
	0.764829	0.003179
Regresión Logística SGD		
c	**Precisión**	**Desv.**
0.1	0.660019	0.075157
0.3	0.642637	0.094687
0.5	0.583213	0.065870
0.7	0.599605	0.116481
0.9	0.579930	0.118764

Multilayer Perceptron		
Neuronas	**Precisión**	**Desv.**
25	0.924792	0.007318
50	0.967091	0.015553
75	0.984382	0.005693
100	0.989306	0.002850
(25, 25)	0.982792	0.007527
(25, 50)	0.993057	0.002102
(50, 25)	0.994655	0.003016
(50, 50)	0.998045	0.000934
(75, 25)	0.971718	0.026802
(75, 50)	0.996737	0.001266
(100, 25)	0.997066	0.001095
(100, 50)	0.994861	0.005239

Table 2: Resultados obtenidos.

de cada uno de los clasificadores. En la Figura 4 se muestran las matrices de confusión de las mejores configuraciones para cada clasificador.

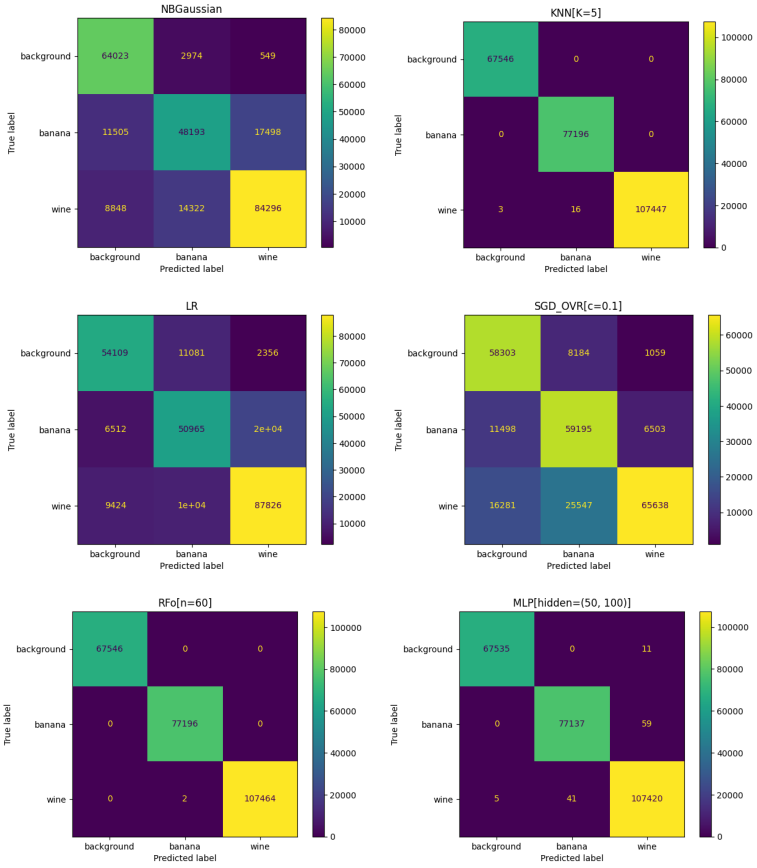

Fig. 4: Matrices de confusión para las mejores configuraciones.

5 Discusión de los resultados obtenidos y conclusiones

El primer resultado evidente que se observa es la eficiencia de cada modelo de clasificador. KNN, Redes Neuronales y Random Forest obtienen unos resultados mucho mejores que los obtenidos por los modelos basados en Naive Bayes y Regresión Logística. El clasificador con mejor resultado de todos fue Random Forest con una cantidad de árboles de 60. Como se puede ver en la Figura 4, este únicamente obtiene 2 fallos en la clasificación de todos los datos, lo cual es un resultado excelente. En comparación además con los siguientes dos mejores modelos, KNN y Redes Neuronales, su tiempo de ejecución era aceptable. Pero, la utilización de uno u otro podría depender mucho de la situación. Mientras que KNN no necesita ningún tiempo de entrenamiento, si necesita un considerable tiempo de ejecución para la clasificación. Esto quizás se podría resolver definiendo regiones de clasificación una vez cargados los datos de entrenamiento, de tal forma que el proceso de clasificación se limitase a detectar la región a la que se pertenece. En cambio, tanto Redes Neuronales como Random Forest tienen un tiempo de clasificación muy bajo en comparación con el de entrenamiento. De cara a utilizar un sistema en tiempo real estos podrían ser mejores opciones.

Para el clasificador KNN se puede observar una correlación entre la precisión del clasificador y el número de vecinos calculados. Se obtienen mejores resultados cuantos menos vecinos se tienen en cuenta. Esto podría indicar que las fronteras de clasificación son muy complejas, con muchas regiones individuales marginadas entre ellas. Al aumentar la cantidad de vecinos, estas regiones acaban desapareciendo y por lo tanto puede afectar negativamente a los resultados.

En el caso de Random Forest no parece haber una correlación entre los hiperparámetros probados y la precisión del clasificador. Lo mismo sucede en las Redes Neuronales, aunque si que es verdad que las de dos capas obtienen resultados ligeramente superiores a las de una.

Como hemos mencionado, los resultados para los tres mejores tipos de clasificadores son excelentes. La obtención de más de un 99% de acierto de forma constante incluso con distintos hiperparámetros puede indicar dos cosas: La gran precisión de los clasificadores para este problema o la facilidad del problema de clasificación. Otro aspecto a destacar es la posibilidad de estar realizando overfitting sobre los datos, lo cual podría causar este gran pico de precisión. Pero la utilización de validación cruzada debería al menos haber aliviado este efecto, por lo que los resultados si que se pueden considerar fiables.

Otro resultado curioso que se observa a partir de las matrices de confusión mostradas en la Figura 4 es que la clase `background` es la clase que menos se tiende a confundir con el resto. En comparación, las clases `plátano` y `vino` tienden a confundirse mucho más entre ellas. Esto coincide con lo que se observa en los Espacios ROC de la Figura 3. En el caso de `background` todos los clasificadores tienden a tener mucho menor False Positive Rate en comparación con las otras dos clases.

References

[1] Ramon Huerta, Thiago Mosqueiro, Jordi Fonollosa, Nikolai Rulkov, and Irene Rodriguez-Lujan. UCI Machine Learning Repository: Gas sensors for home activity monitoring Data Set. https://archive.ics.uci.edu/ml/datasets/Gas+sensors+for+home+activity+monitoring. Acceso: 2022-16-01.

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

[3] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[4] Scikit-learn Documentation. Multiclass and multioutput algorithms. https://scikit-learn.org/stable/modules/multiclass.html. Acceso: 2022-16-01.

[5] Scikit-learn Documentation. sklearn.naive_bayes.MultinomialNB. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html?highlight=multinomialnb#sklearn.naive_bayes.MultinomialNB. Acceso: 2022-16-01.

[6] Scikit-learn Documentation. sklearn.naive_bayes.GaussianNB. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html?highlight=gaussiannb#sklearn-naive-bayes-gaussiannb. Acceso: 2022-16-01.

[7] Scikit-learn Documentation. sklearn.neighbors.KNeighborsClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html?highlight=kneighborsclassifier#sklearn.neighbors.KNeighborsClassifier. Acceso: 2022-16-01.

[8] Scikit-learn Documentation. sklearn.linear_model.LogisticRegression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logisticregression#sklearn.linear_model.LogisticRegression. Acceso: 2022-16-01.

[9] Scikit-learn Documentation. sklearn.linear_model.SGDClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html?highlight=sgdclassifier#sklearn.linear_model.SGDClassifier. Acceso: 2022-16-01.

[10] Scikit-learn Documentation. sklearn.multiclass.OneVsRestClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html?highlight=one%20vs%20rest#sklearn.multiclass.OneVsRestClassifierr. Acceso: 2022-16-01.

[11] Scikit-learn Documentation. sklearn.ensemble.RandomForestClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=random%20forest#sklearn.ensemble.RandomForestClassifier. Acceso: 2022-16-01.

[12] Scikit-learn Documentation. sklearn.neural_network.MLPClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html?highlight=mlp#sklearn.neural_network.MLPClassifier. Acceso: 2022-16-01.

[13] Scikit-learn Documentation. Cross-validation: evaluating estimator performance. https://scikit-learn.org/stable/modules/cross_validation.html. Acceso: 2022-16-01.