DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Clasificación y Selección de Modelos

9 de marzo de 2024 — Laboratorio de Datos

Grupo 100

Integrante	LU	Correo electrónico
Chapana Puma, Joselin Miriam	1197/21	yoselin.chapana@gmail.com
Martinelli, Lorenzo	364/23	martinelli.lorenzo12@gmail.com
Padilla, Ramiro	1636/21	ramiromdq123@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

1. Introducción

1.1. Fuente de datos

A lo largo de este proyecto, trabajaremos con un conjunto de datos de imágenes denominado Sign Language MNIST[1], el cual se encuentra en formato csv, donde cada imagen del set de datos representa una letra en lenguaje de señas americano. El link al dataset se encuentra al pie de página.

1.2. Análisis exploratorio de datos

Antes de ponernos a trabajar con los datos, necesitamos saber mas acerca de ellos. Sabemos que, cada fila del dataset representa una imagen de 28x28 pixeles en escala de grises que corresponde a una letra en lenguaje de señas. Miremos a continuación un pequeño fragmento del mismo. En este, podremos ver las semejanzas y diferencias que tenemos entre las distintas letras. También, se puede ver en menor medida similitudes entre letras de la misma clase.

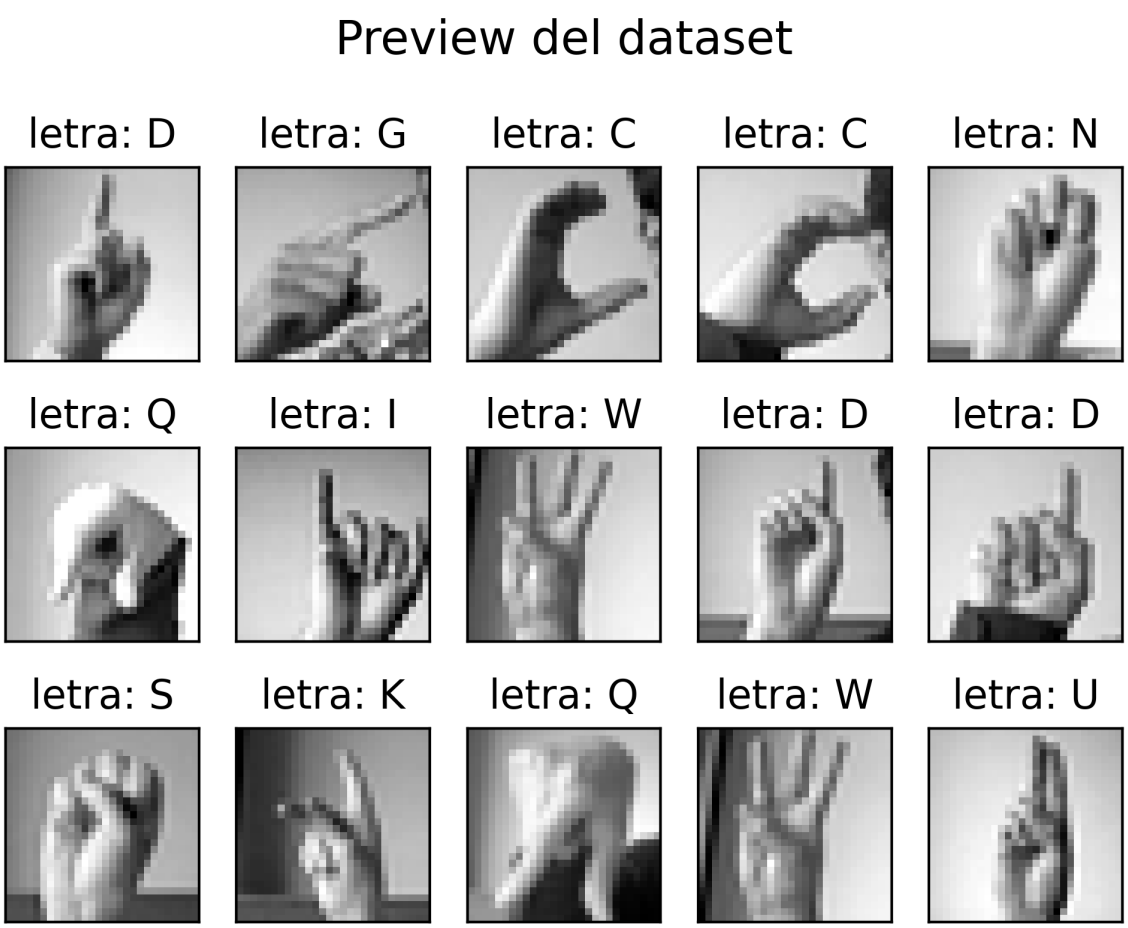


Figura 1: Primeras quince imagenes del dataset

A partir del gráfico anterior, podemos inferir que no todos los pixeles son realmente relevantes para diferenciar imágenes entre si. Por ejemplo, podemos ver que aquellos pixeles correspondientes al fondo no aportan información alguna. ¿Cuántos pixeles necesitaremos realmente para diferenciar las imágenes?

Por otro lado, viendo por ejemplo, que las letras C y D aparecen más de una vez, también surge la pregunta. ¿Están balanceadas las distintas clases que identifican a las letras? En el gráfico a continuación, podremos notar que tenemos un dataset con una excelente distribución de clases. ¿Por qué me interesa tener una cantidad pareja de cada muestra? Esto, adquiere relevancia puesto que al entrenar nuestros modelos en un futuro, no queremos tener sesgos.

Obs: Las letras J y Z no poseen ningún ejemplar puesto que requieren de movimiento para su seña.

[1]Link al dataset https://www.kaggle.com/datasets/datamunge/sign-language-mnist

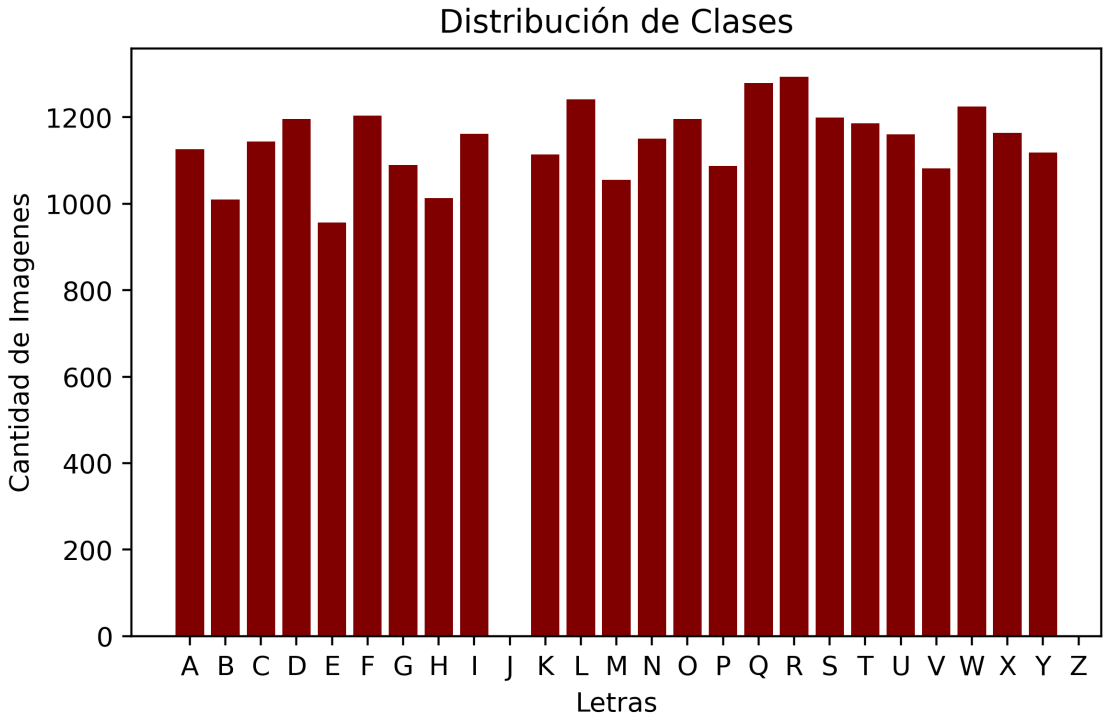
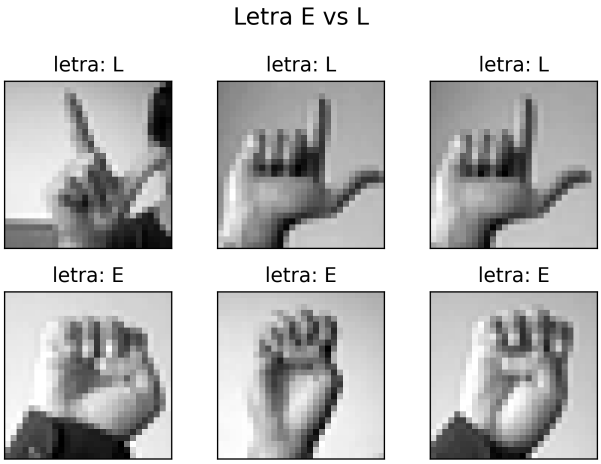
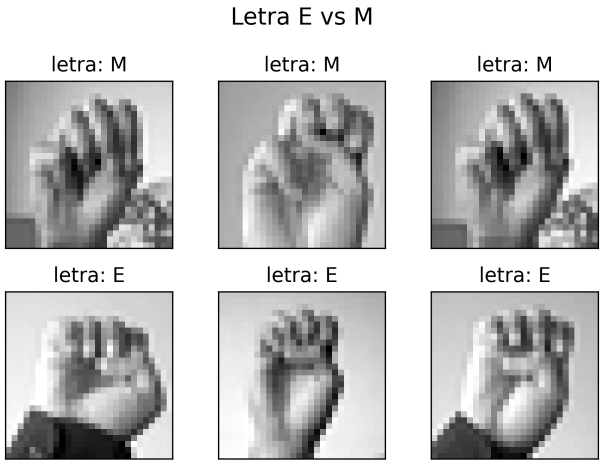


Figura 2: Cantidad de muestras de cada letra

Otra pregunta que naturalmente surge al explorar el dataset es que tan distintas son las letras entre sí, y si las diferentes muestras respecto a una misma letras presentan diferencias notorias o no. Para eso, comparemos por ejemplo, las letras E, L y M.

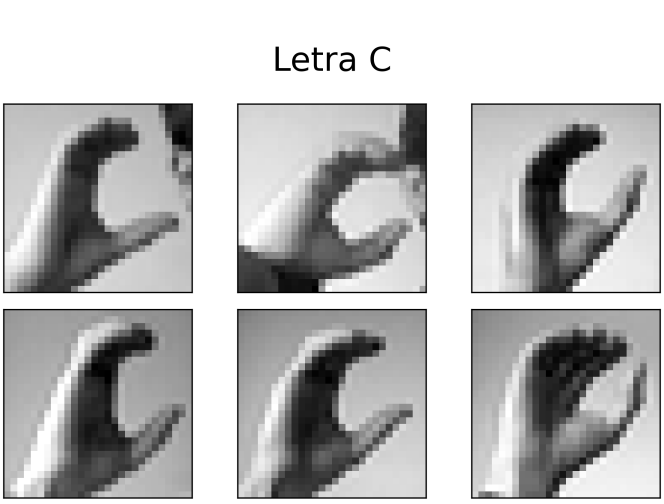


(a) Muestras de las letras E y L

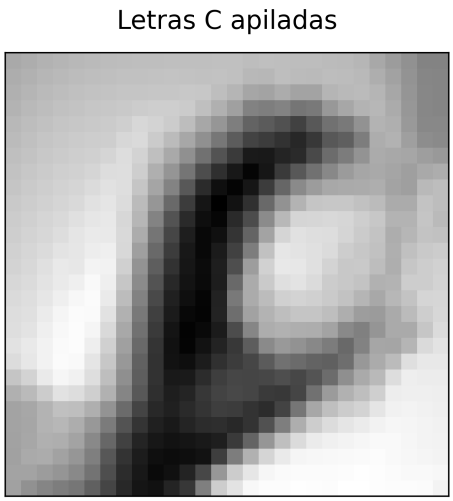


(b) Muestras de las letras E y M

En la imagen anterior, podemos apreciar que tanto la letra E como la M son muy distintas a la letra L, sin embargo, poseen muchas semenjanzas entre sí. Esto, es algo a tener en cuenta a la hora de plantear los modelos, cuales serán los pixeles mas representativos para imagenes similares. Miremos también, si en imagenes pertenencientes a una misma letra hay diferencias notorias.



(a) Distintas muestras de la letra C



(b) Todas las muestras de la letra C apiladas

Si bien, a simple vista, cuando comparamos algunos ejemplares, pareciera que las diferencias pueden llegar a ser significativas, a la hora de comparar todos los ejemplares, en este caso, apilandolos, notamos que forman una silueta que representa muy bien la seña a la que corresponde.

A partir de todo el análisis anterior, pudimos notar que no representa diferencias significativas el trabajar con imagenes respecto a datasets como el del titanic, respecto a su dificultad. Si, quizas, lleva algo mas de tiempo su análisis, pero con las herramientas indicadas no presenta mayores dificultades.

2. Modelo

2.1. ¿La imagen corresponde a una seña de la L o a una seña de la A?

En esta sección, trabajeremos con un subconjunto del dataset original, donde solo esten las letras A y L. Nuestro objetivo, será encontrar un modelo que diferencie estas letras con la menor cantidad de pixeles posibles. Comencemos entonces viendo cuantas muestras tenemos.

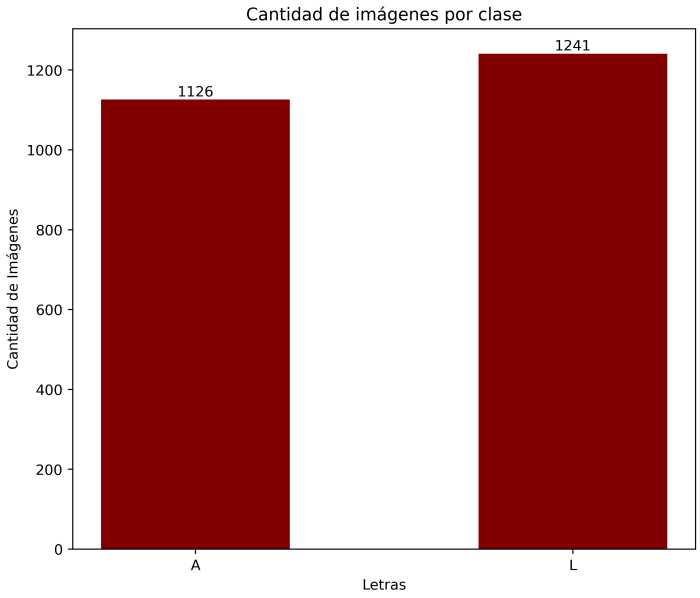
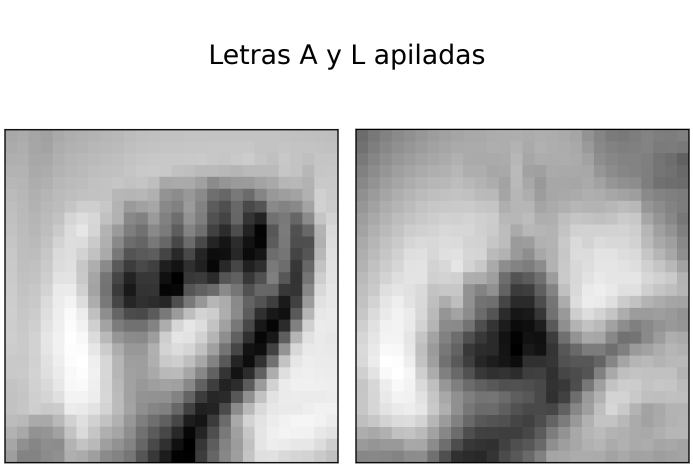
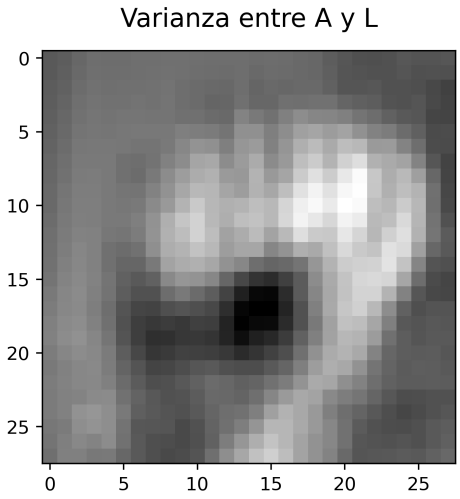


Figura 5: Distribución de Letras

Sabemos entonces, que tenemos buena distribución lo que nos ayuda al armado de los conjuntos de train y test. Veamos ahora, las diferencias entres estas letras en la imagen a continuación. Del lado izquierdo tenemos una comparativa entre todas letras L y las letras A apiladas, y del otro lado, tenemos la resta entre ambas, puesto que es de nuestro interes obtener las zonas de mayor varianza.



(a) Todas las letras A y L apiladas y reescaladas



(b) L - A

Notamos que, la zona con mayor varianza seria en la zona mas oscura que aparece en el medio de la imagen o, en su defecto, la zona mas blanca. Para saber esto con precisión utilizamos una función de pandas que nos devuelve en una lista las 10 posiciones con los valores mas altos. Destacando, que los pixeles mas representativos son el 301, 274 y 246.

2.2. Construcción del Modelo

Una vez, que conocemos bastante acerca de nuestro dataset, estamos en condiciones de armar nuestro clasificador Knn. A fines de afinar nuestro modelo, hicimos pruebas utilizando distintas cantidades de pixeles, la idea es, utilizar la menor cantidad posible para asi no usar datasets tan grandes. Veamos los resultados.

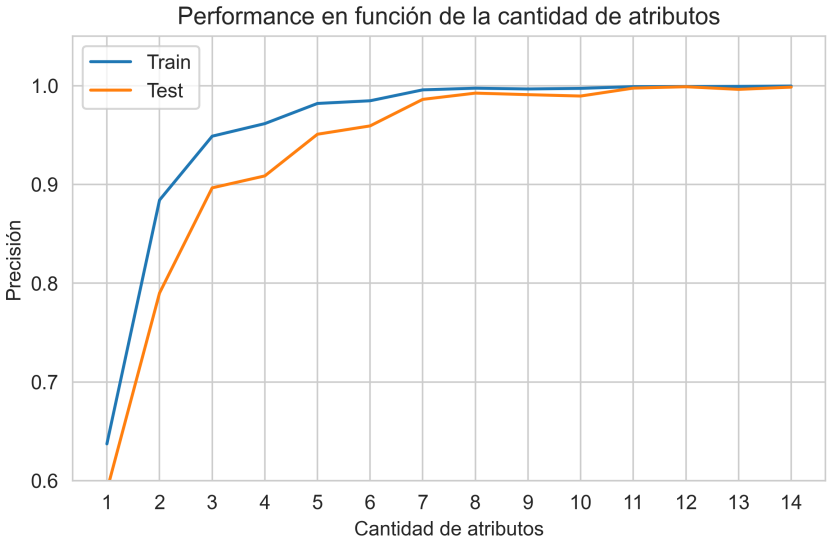
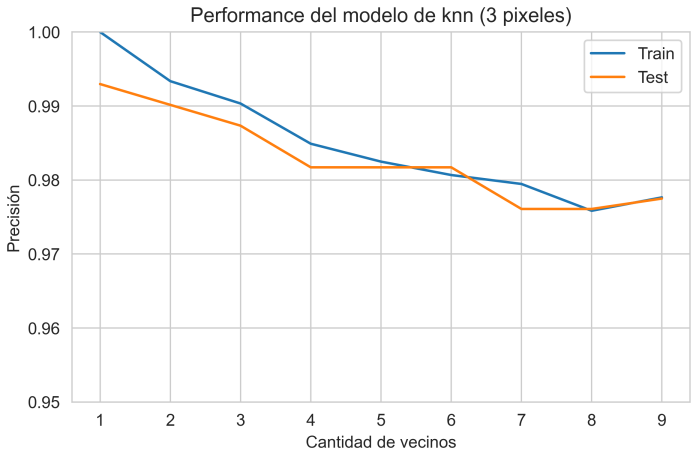


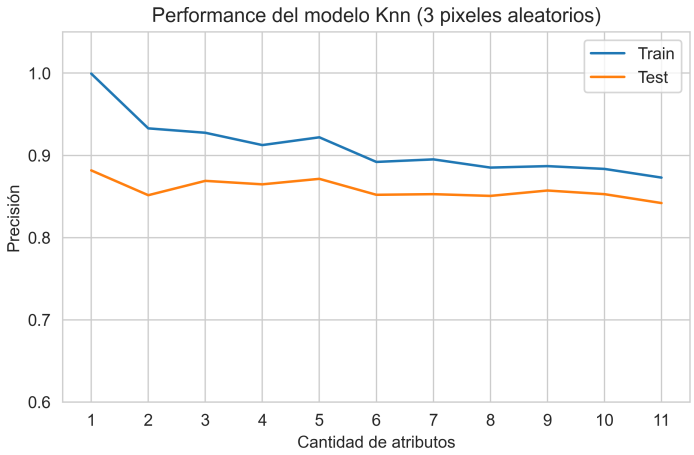
Figura 7: Pixeles vs Accuracy

Una observación, para tener un gráfico mas confiable, lo que hicimos fue realizar la prueba de precisión una cierta cantidad de veces, tomando en cada iteracion pixeles al azar, para luego tomar su promedio, de esta manera garantizamos cierta aleatoreidad.

Como se puede apreciar en la imagen anterior, la precisión aumenta de manera notable a partir de los 3 pixeles para luego estabilizarse alrededor de los 7. Veamos que ocurre ahora para algunos de estos valores si modificamos la cantidad de vecinos.

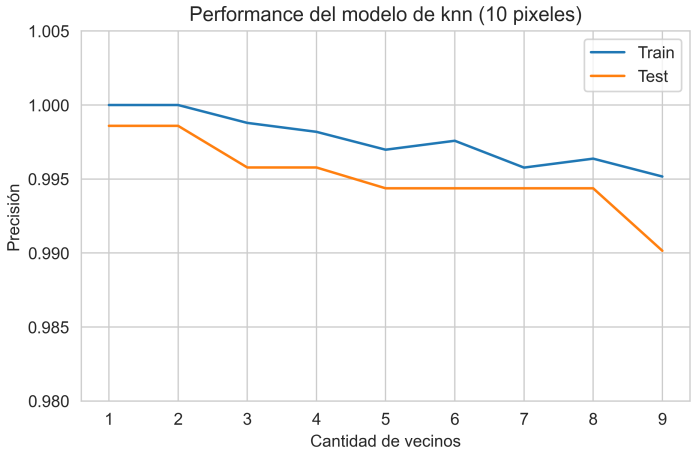


(a) Pixeles mas significativos

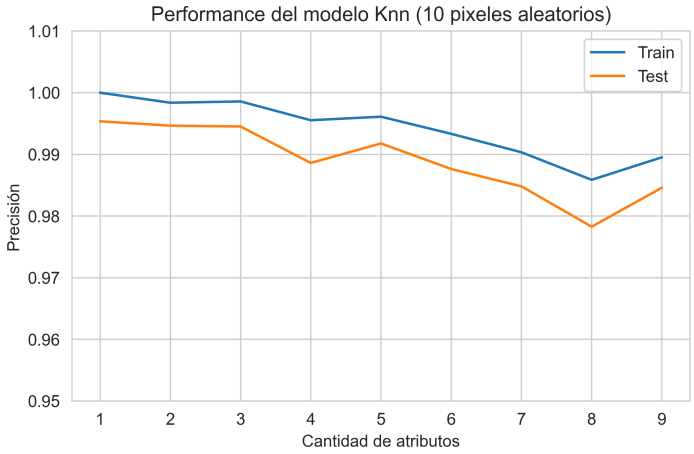


(b) Pixeles aleatorios

Podemos ver claramente que, contrario a lo que uno podria pensar, la performance baja a medida que tenemos mas vecinos, siendo el numero ideal exactamente 1. Notar que en estas pruebas tomamos por un lado los pixeles de mayor varianza que habiamos encontrados antes y, pixeles aleatorios. Veamos tambien, un caso con mas pixeles a ver que ocurre



(a) Pixeles de alta varianza



(b) Pixeles aleatorios

Al igual que cuando testeamos la performance en función de la cantidad de atributos, cuando hicimos la prueba con 3 y 10 pixeles aleatorios, la repetimos una cierta cantidad de veces y obtuvimos su promedio, con el cual realizamos los gráficos anteriores.

2.3. Conclusiones Clasificación Binaria

Pudimos destacar la importancia de comenzar con un analisis exploratorio de los datos correspondientes a las letras L y A, lo que nos permitió notar y diferenciar aquellos pixeles que presentan mayor varianza entre las dos letras, para luego mejorar la performance del modelo con una cantidad reducidad de datos. Una vez hecho esto, tambien observamos que la perfomance, en nuestras pruebas, baja al aumentar la cantidad de vecinos. Por lo tanto, podemos concluir que tomando un modelo que tenga entre 3 y 10 pixeles con alta varianza y con k = 1, tendriamos un modelo eficiente con una perfomance aproximada de 0,99.

3. Clasificador Multiclase

3.1. ¿A cuál de las vocales corresponde la seña en la imagen?

En esta sección, se intentará encontrar un modelo de árbol de decisión que prediga las vocales. Para esto, armaremos un dataset a partir del original el cual contenga unicamente estas.

A fin de encontrar el mejor modelo, realizamos distintas pruebas modificando los paramentros del modelo. En primer lugar, lo que hicimos fue testear alturas entre 1 y 14. Para esto, utilizamos dos métodos, por un lado dividimos el dataset original en train, validation y test, para luego entrenar el modelo con los datos de train y medirlo contra los datos de validation.

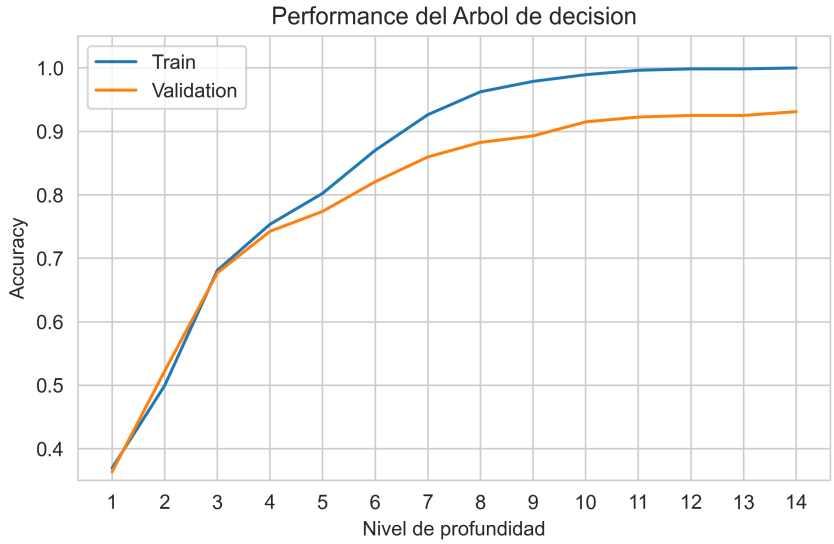
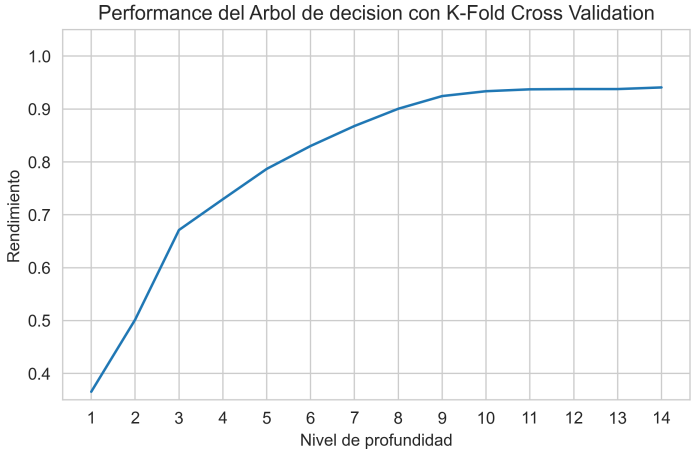
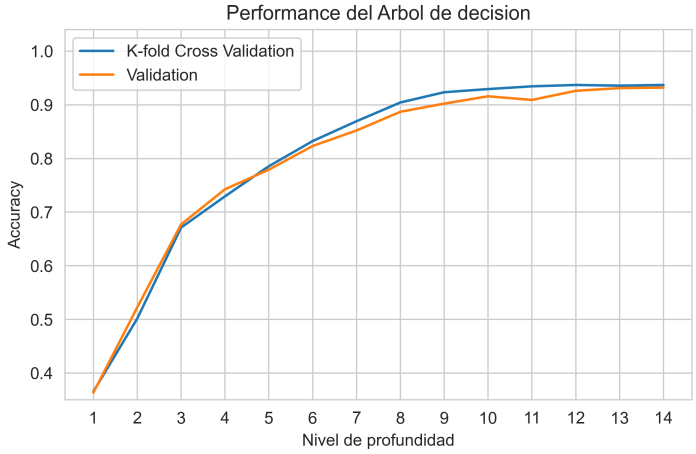


Figura 10: Prueba para distintas profundidades

Por otro lado,para poder comparar y seleccionar el mejor árbol de decisión, se empleo validacion cruzada con k-folding utilizando los datos de train, los cuales en este caso son la union entre los datos de train y validacion anteriores. Veamos entonces los resultados comparados con el test anterior.



(a) KFold



(b) Comparacion entre ambos tests

A raiz, de los gráficos anteriores, pudimos notar que a partir de la altura 10, el modelo tiende a mantener su precisión. Por eso, nos quedaremos con estos valores e iremos testeando que ocurre combinandolos con otros hiperparametros. Para eso, realizamos un Grid Search tomando en cuenta los profundidades anteriormente nombradas, y los criterios, entropia y gini.

3.2. Grid Search

En la imagen a continuación, la cual corresponde al Grid Search, podemos notar, primero que utlizar entropia es un poco mas eficiente que utilizar gini. Y, en segundo lugar, que el rendimiento entre profundidades practicamente no difiere. Por lo tanto, tomaremos aquel de menor profundidad en busca de tener un modelo mas simple. De esta manera, el modelo que escogeremos al final es un árbol de altura 10 con el criterio entropia.

Criterio	Altura	Precision
gini	10	0.9306
gini	11	0.937
gini	12	0.9334
gini	13	0.9362
gini	14	0.9367
entropy	10	0.9467
entropy	11	0.9467
entropy	12	0.9467
entropy	13	0.9441
entropy	14	0.9477

Figura 12: Resultados Grid Search

3.3. Conclusión

Para encontrar el mejor modelo, comenzamos testeando con el metodo de Kfold Cross Validation diferentes alturas, donde notamos que a partir de una altura 10 casi no variaba la precision de nuestro modelo. Luego, con esta información realizamos un Grid Search que incluia alturas entre 10 y 14 además de los criterios entropy y gini. A partir de este pudimos conlcuir que el modelo final es un árbol de altura 10 con el criterio entropia, el cual testeamos poniendo a prueba el modelo frente a los datos de test que separamos al principio de todo el proceso. En la página a continuación mostramos la matriz de confusión y los resultados finales.

3.4. Matriz de Confusión y Test

Para poder visualizar mejor donde acierta y donde no nuestro modelo, se recurrió a la matriz de confusión. Esta matriz es una herramienta que muestra de manera detallada cómo el modelo clasifica las muestras en cada una de las clases. Donde cada fila de la matriz representa la clase real, mientras que cada columna representa la clase predicha por el modelo.

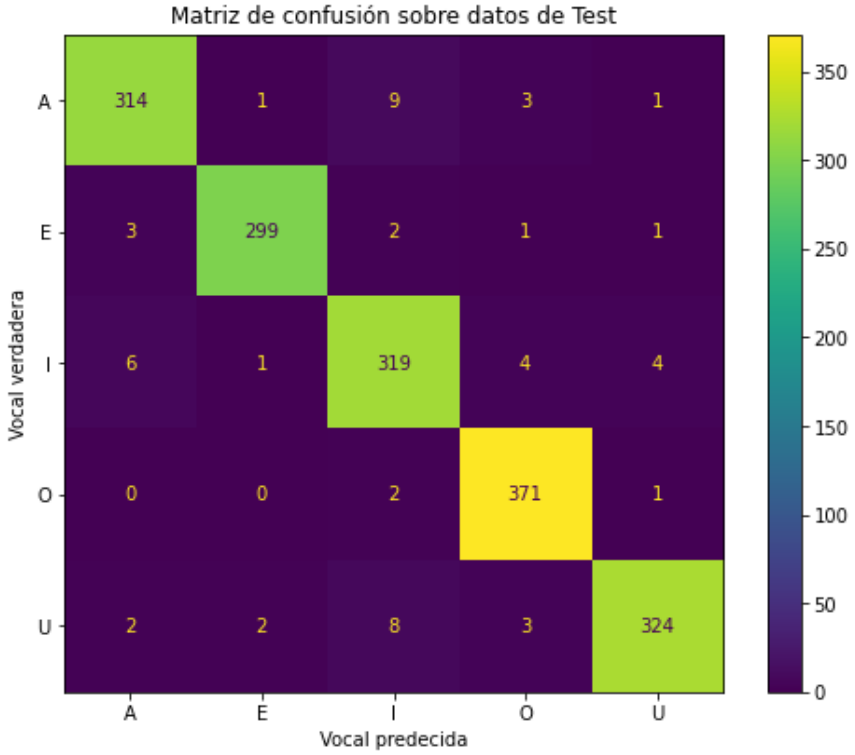


Figura 13: Matriz de confusión

Podemos notar, que nuestro modelo tiende a confundir las letras I y A por un lado, y las letras U e I. Sin embargo, tambien vemos que tiene un excelente desempeño mostrando una precisión del 0.9678.

Cabe aclarar, que tanto la matriz como la precisión fueron calculadas a partir de los datos de test, los cuales, habiamos reservado hasta este momento.