Aprendizaje Automático 1

Trabajo Práctico 1

Clasificación de expresiones genómicas

Grupo Data Sapiens

Cursada: Aprendizaje Automático 1, Primer Cuatrimestre 2025
Integrantes:

- Calderón, Lara
- Muschietti, Bruno
- Guerrero Schmidt, Mateo
- Kannemann, Vladimir
- Rancati, Hernán Gabriel

Fecha: 1 de mayo de 2025

Primer cuatrimestre de 2025

Introducción

La hiperplasia, es un fenómeno en el que las células experimentan un crecimiento anormal y descontrolado. Es de nuestro interés saber por qué algunas células que experimentan hiperplasia se convierten en células cancerosas mientras que otras no. Para eso, contamos con un dataset de 500 instancias correspondientes a 500 pacientes con el valor de 200 genes que se consideran relevantes para este problema. Además, cada instancia tiene una etiqueta que indica si el paciente tuvo nuevas incidencias de hiperplasias o no.

Nuestro trabajo consiste en encontrar si existe correlación entre estos genes y la hiperplasia. Para eso vamos entrenamos distintos modelos de clasificación para ver cual se desempeña mejor.

1 Separacion de Datos

1.1. Funcion de split en sets de entrenamiento y test

Comenzamos definiendo una función propia para dividir el conjunto de datos en subconjuntos de entrenamiento y evaluación. Dado que las clases no están balanceadas (el 70 % de las instancias son clasificadas negativas), haremos una partición estratificada, conservando la proporción original de clases tanto en el conjunto de entrenamiento como en el de evaluación. Además, incluimos un `shuffle` inicial para mezclar el dataset previendo la posibilidad de que exista un orden implícito en los datos que no queremos que los algoritmos aprendan.

Respecto a la partición de datos, elegimos un 80 % para entrenamiento y un 20 % de evaluación final (held-out).

Adicionalmente, exploramos también la posibilidad de preservar la distribución de los atributos en ambas particiones, aunque descartamos esa estrategia por dificultades técnicas. No obstante, consideramos que podría resultar valioso desarrollar una heurística que preserve la distribución original en futuras particiones.

2 Construcción de Modelos

Para este punto, construimos y evaluamos modelos basados en **árboles de decisión**, con el objetivo de obtener una **estimación realista de la performance** de los mismos.

Importante: de acá en más sólamente utilizaremos el score promedio cuando hagamos K-fold cross-validation.

2.1. Entrenamos un arbol de decisión y medimos su performance

Comenzamos creando un árbol de altura 3, con el resto de sus hiperparámetros por default. Obtuvimos scores con diferentes métricas, utilizando validación cruzada con Stratified K-Fold Cross-Validation (K=5), lo cual asegura que cada subconjunto (fold) mantiene aproximadamente la misma proporción de clases que el conjunto original.

En la Figura 1 observamos que los scores por fold son consistentes y se mantienen cerca del promedio en entrenamiento. Los scores son ligeramente menores en validación respecto a entrenamiento para Accuracy. En AucROC y AucPRC los scores empeoran notoriamente en validación. Los Scores globales dieron resultados similares a los promedios obtenidos en los folds.

	Permutación	Accuracy (training)	Accuracy (validación)	AUCPRC (training)	AUCPRC (validación)	AUC-ROC (training)	AUC-ROC (validación)
0	1	0.821875	0.7375	0.682198	0.394770	0.836924	0.604167
1	2	0.8375	0.7125	0.699528	0.344841	0.849961	0.492932
2	3	0.853125	0.7250	0.694402	0.372037	0.791272	0.597098
3	4	0.8375	0.6625	0.739077	0.464251	0.852596	0.689732
4	5	0.821875	0.7875	0.685031	0.535909	0.805455	0.688727
5	Promedios	0.834375	0.7250	0.700047	0.422362	0.827241	0.614531
6	Globales	(NO)	0.7250	(NO)	0.417926	(NO)	0.607068

Figura 1: Scores de arbol default en cross-validation.

2.2. Exploramos hiperparámetros de árboles

En la Figura 2 puede verse que la mejor combinación se alcanzó con un arbol de altura 3. Al aumentar la profundidad a 5, se observó un comportamiento de sobreajuste, evidenciado por una mejora en el conjunto de entrenamiento y una disminución del score en validación. Ambas métricas, gini y entropy muestran desempeños similares, obteniéndose de entropy valores ligeramente inferiores en validación. Se observa, además que, para el árbol sin cota de altura, el modelo memorizó completamente el set de entrenamiento, confirmando un sobreajuste extremo.

	max_depth	criterion	Accuracy(training)	Accuracy(validación)
0	3.0	gini	0.834375	0.7225
1	5.0	gini	0.925625	0.7075
2	NaN	gini	1.000000	0.6825
3	3.0	entropy	0.811250	0.6750
4	5.0	entropy	0.901250	0.6675
5	NaN	entropy	1.000000	0.6575

Figura 2: Scores según la exploración de hyper-parámetros con ParameterGrid.

3 Comparación de Algoritmos

Para poder encontrar un modelo que generalice bien los datos, nos gustaría ver varios algoritmos distintos. Cada uno de ellos tiene su propio set de hiperparámetros por lo que también queremos explorar sus distintas configuraciones. Como no queremos probar muchos modelos distintos y muchos hiperparámetros son continuos, sería muy costoso hacer un GridSearch que recorra todo el espacio. Por este motivo, es útil usar un RandomSearch cercando los valores de los hiperparámetros según sea conveniente.

Por último, se comparó los modelos obtenidos con el RandomSearch con LDA y GaussianNaiveBayes en sus versiones default.

3.1. Elección de hiperparámetros

Para la búsqueda de hiperparámetros con RandomSearch se eligieron tres algoritmos distintos: Árboles de decisión, KNN y SVC. El método de evaluación de los modelos fue haciendo un k-fold cross validation con 5 folds y usando la métrica AUC-ROC.

Respecto a La cantidad de iteraciones del RandomSearch, regulamos el valor dependiendo del algoritmo, dado que las velocidades de ejecución de los algoritmos son muy dispares. Particularmente, KNN es significativamente más lento que los otros dos algoritmos.

3.1.1. Árboles de decisión

En este caso, los hiperparámetros elegidos para variar fueron: `criterion, max_depth, min_impurity_decrease` y `min_sample_split`. Para el criterio se consideraron las opciones Gini y Entropy ya que son las más utilizadas. Para la altura máxima, se consideró desde 3 hasta la raíz cuadrada de la cantidad de atributos, o sea, 15. El `min_impurity_decrease` indica la mínima variación que tiene que tener un corte para ser considerado; para elegir su valor se tomó en consideración los ejemplos de la documentación y luego se ajusto haciendo un par de iteraciones de prueba y viendo que valores óptimos tomaba. Finalmente, se eligió evaluar en una escala logarítmica desde 0.003 y 0.5. El valor del `min_sample_split` se eligió arbitrariamente entre 2 (menor valor posible) y un valor no tan grande, en este caso, 15.

También se probó el hiperparámetro `max_features` que limita la cantidad de atributos a ser considerados para el modelo. Pero este parámetro introduce aleatoriedad y consigo, mucha varianza, por lo que se decidió descartarlo.

Finalmente, luego de 1000 iteraciones, el óptimo encontrado tuvo un score de **0.662** con la siguiente configuración: {`'criterion': 'gini', 'max_depth': 13, 'min_impurity_decrease': 0.001056, 'min_samples_split': 7`}.

3.1.2. KNN

En este caso, los hiperparámetros elegidos para optimizar fueron: `n_neighbors, p` y `weights`. El rango de la cantidad de vecinos se eligió arbitrariamente entre 2 y 50, siendo 50 un valor grande pero razonable teniendo en cuenta que cada modelo lo entrenamos con 320 datos. El valor de `weights` indica como son pesados cada punto dentro del rango; se probó con los dos valores por default que son 'uniform' y 'distance'. Por último, el parámetro p indica la potencia usada para calcular la distancia de los puntos; se eligió variar con una distribución uniforme entre 1 y 3.

Finalmente, luego de 200 iteraciones, el óptimo encontrado tuvo un score de **0.870** con la siguiente configuración: {`'n_neighbors':13, 'p':1.151, 'weights': distance'`}.

3.1.3. SVC

En este caso, los hiperparámetros elegidos para variar fueron: `C, gamma` y `kernel`. El valor de C fue primero elegido en base a ejemplos de la documentación y luego ajustada en base a pruebas y, particularmente, con la curva de aprendizaje realizada en la próxima sección (Figura 3), que se puede ver en la figura 3. Con esto, el rango elegido fue de 1 a 50 con una distribución loguniforme. El valor de gamma

se ajustó a base de pruebas y ver que óptimos tomaba y se llego a un rango de 1×10^{-6} a 1×10^{-3} con una distribución loguniforme. Para el kernel se probó entre los siguientes: 'linear', 'poly', 'rbf' y 'sigmoid'.

Finalmente, luego de 1000 iteraciones, el óptimo encontrado tuvo un score de **0.925** con la siguiente configuración: {`'C': 37.71, 'gamma': 0.0001249, 'kernel': 'rbf'`}.

3.2. Comparativa de los modelos entrenados

Para comparar los modelos obtenidos entrenamos 3 modelos más. Estos son LDA, Naive Bayes y Dummy Classifier. Los scores del auc-roc obtenidos usando Stratified K-fold cross validation con 5 folds fueron: Naive Bayes Default (0.812), LDA Default (0.682), Dummy most-frequent (0.5). El motivo de no hacer RandomSearch en estos modelos es que no tienen hiperparámetros que permitan modificar la complejidad del algoritmo.

Lo primero que podemos ver es que el Dummy Classifier obtuvo un score de 0.5, ésto se debe a que, a pesar de que las clases están desbalanceadas, la métrica es invariante frente a estos desbalances. Por ende, podríamos pensar que esta métrica es redundante, pero nos sirve como base de comparación para las otras métricas.

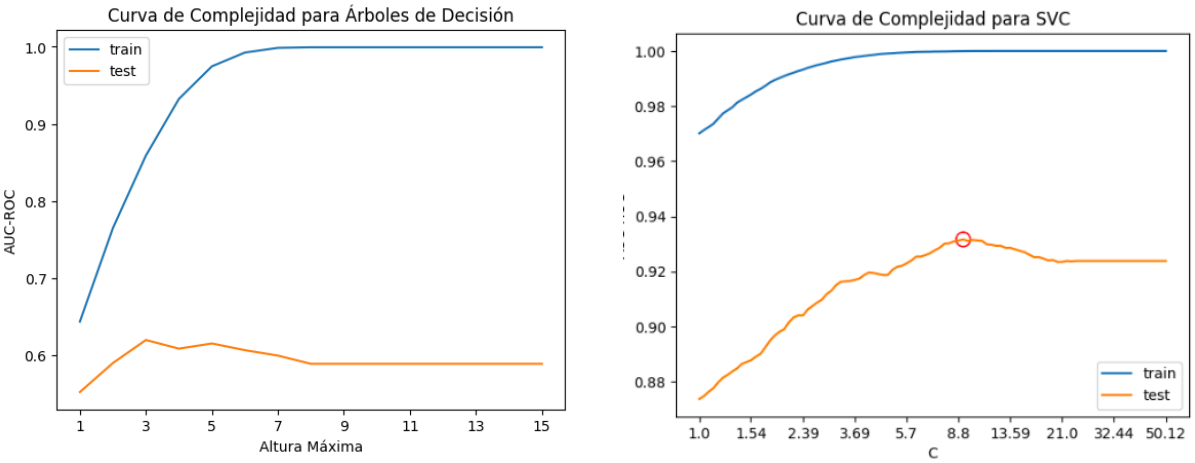
Otra cosa que podemos notar es que los árboles de decisión no son un buen clasificador para este problema ya que no superan a los modelos de LDA y NB. Por otro lado, tanto KNN como SVC superan significativamente estos modelos, siendo este último el que obtiene scores más altos.

Observando que los tres mejores modelos son SVC, KNN y NB ganamos cierta intuición de cómo puede ser la distribución en los datos. El mejor modelo de SVC fue con el kernel 'rbf' que es de tipo radial, KNN también funciona para estructuras radiales. Por otro lado, NB supone distribuciones gaussianas. Con todo esto podemos asumir que las distribuciones parecieran tener cierta distribución radial o gaussiana con matrices de covarianza distintas para cada clase(debido al bajo rendimiento de LDA), pero seguramente la frontera sea no lineal (dados los bajos scores de Árboles, Random Forest y LDA).

4 Exploración de los mejores modelos

4.1. Curvas de complejidad

En esta sección analizamos la capacidad de generalización de los modelos a través de curvas de complejidad para árboles de decisión y SVC (Figura 3). Para los árboles de decisión, se observa que profundidades menores a 3 generan alto sesgo (subajuste), ya que los scores de entrenamiento y validación mejoran al incrementar la profundidad. A partir de una profundidad mayor a 5, se evidencia alta varianza (sobreajuste): el rendimiento en entrenamiento continúa mejorando mientras que en validación comienza a degradarse. En el caso del modelo SVC, el parámetro que regula la complejidad es **C**. A partir de las curvas obtenidas, se observa que el rendimiento óptimo se alcanza aproximadamente en C=9.15.



(a) Árboles (b) SVC

Figura 3: Comparación de la complejidad en modelos

4.2. Curvas de aprendizaje

También generamos curvas de aprendizaje (Figura 4) para los modelos de árboles de decisión, SVC, e incluimos LDA. En todos los casos se observa que las curvas de test no han alcanzado una meseta, lo cual sugiere que el modelo podría beneficiarse de más datos de entrenamiento. Esto indica que aún existe capacidad de mejora en el desempeño general.

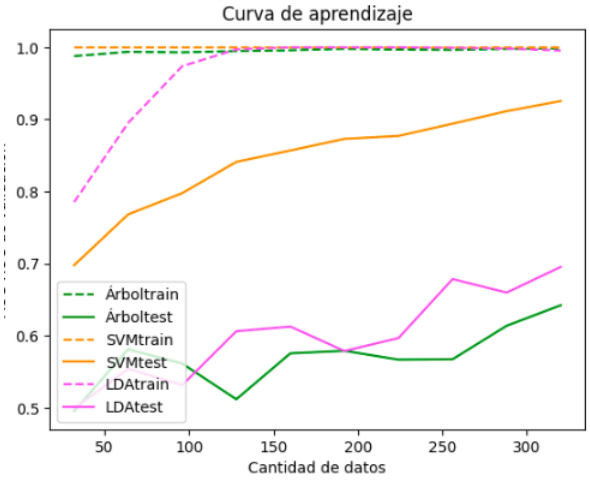


Figura 4: Curvas de aprendizaje para Árboles de Decisión, SVC y LDA

4.3. Pruebas con modelos Random Forest

Como paso siguiente, nos propusimos estudiar el funcionamiento de los métodos de ensamble, con el objetivo de evaluar si lográbamos disminuir la varianza obtenida y, por ende, mejorar el desempeño de los árboles de decisión. Por esto, construimos un modelo de Random Forest con 200 árboles.

A continuación presentamos una figura que muestra la relación entre el parámetro Max Features (que determina cuántos atributos se consideran al generar cada división en un nodo) y el puntaje AucROC. Este parámetro tiene un efecto directo en la decorrelación entre los árboles del bosque, lo que impacta en

la capacidad de generalización del modelo. Se alcanza un máximo de performance en test con 90 atributos pero es relativamente parejo en el rango a partir del logaritmo de la cantidad total de atributos.

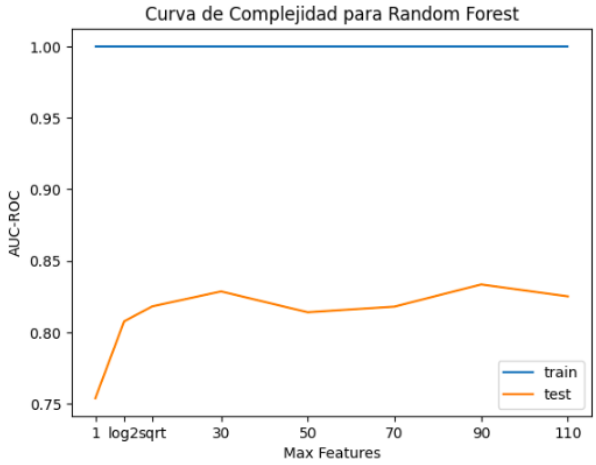


Figura 5: Relación entre el parámetro `max_features` y el AUC-ROC en Random Forest

Finalmente,construimos una curva de aprendizaje para Random Forest: no muestra señales claras de haber alcanzado una meseta. Por el contrario, se observa una tendencia creciente en el rendimiento a medida que se incrementa la cantidad de datos de entrenamiento. Esto sugiere que el modelo aún está aprendiendo y que incorporar más datos podría seguir mejorando su desempeño.

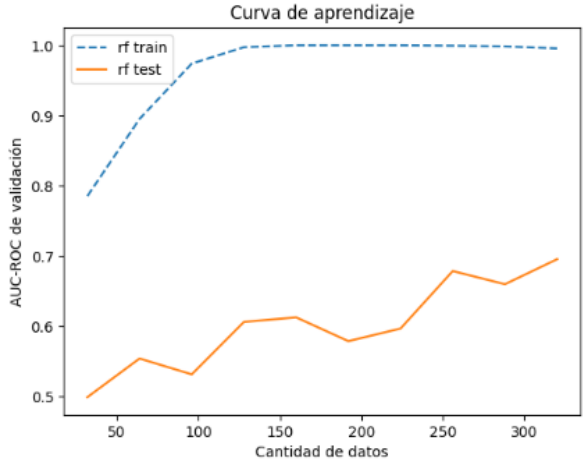


Figura 6: Curva de aprendizaje del modelo Random Forest

5 Evaluación de Performance

Nuestro mejor modelo fue un `SVC` (Support Vector Classifier) con los siguientes hiperparámetros: `C= 9.15`, `kernel= rbf`, `gamma= 0.0001`. El C lo obtuvimos de las curvas de complejidad mientras que el gamma y kernel los obtuvimos de hacer Random Search.

Sin embargo, observamos que el modelo posiblemente esté sobreajustando y memorizando el training set, ya que el *AUC ROC* sobre los datos conocidos de entrenamiento nos dio `0.9999`, que es mucho mayor

que el del held out (datos desconocidos): `0.8786`.

Basándonos en el score reportado por `X_held_out`, debido a que no fue utilizado para la búsqueda de hiperparámetros ni para el entrenamiento, estimamos un score $AUC\ ROC$ de **0.8786**.

6 Conclusiones

A lo largo del trabajo, implementamos, analizamos y evaluamos distintos modelos de aprendizaje automático con el objetivo de predecir la aparición de nuevas hiperplasias a partir de diversas mediciones realizadas en 500 pacientes.

De los resultados obtenidos, podemos afirmar, como nos propusimos, que existe una correlación entre los datos de los 200 genes y los pronósticos de futuras apariciones de hiperplasias, dado que algunos de los modelos pudieron predecir en buena medida las etiquetas.

Más en profundidad, podemos intuir de cierta manera cómo se da esta correlación. El hecho de que los modelos con peores resultados —y, por ende, con más dificultades a la hora de capturar la estructura subyacente de los datos— hayan sido LDA y el Árbol de Decisión nos permite deducir que este problema no posee fronteras de decisión lineales, ya que los modelos mencionados se basan en este tipo de separación.

Por el contrario, las fronteras de decisión no lineales pueden ser bien interpretadas por SVC y KNN, casualmente los dos modelos que mejores resultados dieron. Más aún, al optimizar los hiperparámetros de SVC, encontramos que el kernel seleccionado fue `rbf` (un kernel radial), lo cual coincide con dicha afirmación, dado que este kernel no lineal permite proyectar los datos en un espacio de dimensión más alta para así encontrar fronteras de decisión curvas y complejas.

Finalmente, al observar el score reportado por el mejor modelo ($AUC\ ROC$: **0.8786**) y contextualizarlo dentro del análisis de las curvas de aprendizaje (Figura 4) se puede concluir que el desempeño obtenido aún presenta margen de mejora. No obstante, aunque no es posible probarlo de manera precisa, es probable la presencia de un error irreducible inherente al problema.

6.1. Algunos problemas con los que tropezamos:

Probamos usar QDA intentando ganar intuición sobre los malos resultados de LDA, con la esperanza de que la flexibilidad en la matriz de covarianza del primer modelo produjera mejores resultados pero no fue así, de hecho obtuvimos una advertencia del algoritmo indicando que poseíamos pocas muestras para la cantidad de atributos del dataset. Es posible que los malos resultados de este modelo se deban a un problema de estimación de la matriz de covarianza para la cantidad de datos en posesión.

Luego de la clase de métricas, nos dimos cuenta de que calculamos mal el AUCROC, ya que le pasamos entonces el vector de predicciones de clases en vez del vector de probabilidades para que calcule el umbral, por lo que tuvimos que volver a armar los gráficos. Durante el ejercicio de K-fold cross validation utilizamos de manera incorrecta la API de SciKitLearn y finalmente decidimos implementar el algoritmo. En general, nos dio la impresión de que hay que tener muchísimo cuidado en el uso de los datos y las APIs puesto que fácilmente se puede obtener un resultado que parezca razonable pero que no corresponda a

aquello que se intenta hacer; a diferencia de lo que sucede con problemas de solución exacta de pequeño tamaño donde puede prepararse una batería de tests para comprobar los resultados o incluso pruebas formales.