DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico - Aprendizaje supervisado

Clasificación de expresiones genómicas

1 de mayo de 2025 Aprendizaje automatico I - 2025

Grupo Overfitters & Underpaids

Integrante	LU	Correo electrónico
Manuel Fernández Burda	1700/21	manufernandezbur@gmail.com
Santiago Nahuel Eliges	386/20	saneliges@gmail.com
Giancarlo Moroni	780/22	moronigiancarlo1@gmail.com
Nicolás Spisso	477/19	nicolasspisso@hotmail.com
Aaron Bernal Huanca	815/22	aaronbernal28@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

Resumen

Este informe usa 500 muestras de ARN de 200 genes, tomados de pacientes con lesiones pre-tumorales, y los analizamos con diversas técnicas de machine learning. Los datos fueron provistos por grupos de investigación del CONICET (y docentes de la cátedra) en el marco de un proyecto interdisciplinario sobre hiperplasias y pronóstico oncológico. El objetivo es entrenar modelos que ayuden a predecir nuevos casos de forma precisa y automática. Cada muestra fue etiquetada como buen pronostico, si no hubo indicios de nuevas hiperplasias o similares; contrariamente se etiquetaron como de mal pronóstico si hubo una recaída.

1. Separación de datos

De cara a realizar la separación de datos, proponemos mirar primero en qué momento serán empleados. Los datos de desarrollo serán usados en la etapas de Construcción de Modelos (Sección 2), Comparación de Algoritmos (Sección 3) y Diagnostico Sesgo-varianza (Sección 4). Por otro lado, los datos de evaluación serán empleados en la sección Evaluación de Performance (Sección 5) donde se pondrá a prueba nuestro mejor modelo.

Antes de realizar la separación, hicimos un análisis exploratorio de los datos. Buscamos la forma de poder tener en nuestro set de desarrollo y evaluación la misma distribución de los datos. Nos volcamos a intentar mantener no sólo la misma distribución de clases sino también la misma distribución de features. En un primer acercamiento, probamos mirando los promedios, desvíos, máximos y mínimos de cada una; buscando encontrar si había una diferencia en la importancia para determinar la clase de éstas a simple vista, de manera de poder incluirlas a la estratificación de datos para la división en cuestión (ver anexo en el ipynb: Análisis Exploratorio). Luego de plantear esta situación con lxs docentxs y entender que dada la escasez de datos esto podría ser una complicación mayor que no ayudaría al desarrollo de los algoritmos consiguientes, nos volvimos a la idea de separar en test y train aleatoriamente, manteniendo solamente la distribución de clases en ambos conjuntos.

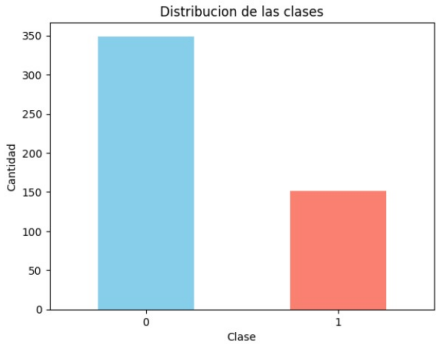

Figura 1: **Distribución de clases en el dataset original.** Donde **1** significa buen pronostico y **0** un mal pronostico

Los datos están visiblemente desbalanceados en clases, como se ve en la Figura 1 (como es de esperar en problemas biológicos reales). El corte que realizamos consiste de separar un 90 % para desarrollo y un 10 % para evaluación, pero asegurando que exista la misma distribución de cada clase en ambos sets. Esto nos permitirá obtener métricas representativas de performance en la realidad de los modelos que entrenemos, adecuando un conjunto de evaluación que respete escenarios lo más fieles posibles a la distribución real. Antes de hacer esto, nos cercioramos que no haya datos repetidos en el dataset.

Haberlo separado de forma completamente aleatoria (sin seguir ninguna regla al hacer el split, sólo sampleando instancias del dataset), podría haber generado que no tengamos ningún dato de la clase positiva en evaluación o entrenamiento, o tener muy pocos. Esto puede impactar negativamente en el aprendizaje del modelo, ya que el entrenamiento no estaría reflejando la distribución real de los datos. Adicionalmente, desde el punto de vista metodológico, la evaluación propuesta no sería significativa para comprobar la performance del modelo.

2. Construcción de modelos

En esta sección se construye y evalúa distintos modelos de tipo árbol de decisión.

2.1. Árbol de altura máxima 3

Sobre el dataset de entrenamiento fitteamos un árbol de decisión limitando la profundidad máxima a 3 nodos por rama. Adicionalmente, medimos la performance de este modelo usando el conjunto de evaluación, obteniendo una performance de 0.8 en accuracy.

2.2. K-fold cross validation

Para obtener mejores estimaciones del rendimiento del árbol de clasificación, se utilizó el método de validación cruzada k-fold con $K = 5$ folds, manteniendo la misma proporción de clases en cada uno (estratificado). Mediante este procedimiento

se estimaron las métricas: *accuracy*, *área bajo la curva Precision-Recall (AUPRC)* y *área bajo la curva ROC (AUC ROC)*, tanto para el conjunto de entrenamiento como para el de validación (observación; el conjunto de validación en este caso es distinto al de evaluación, pues está comprendido dentro de los datos de desarrollo).

Se utilizaron dos formas distintas para calcular estas métricas a partir de los folds. La primera consiste en computar las métricas individualmente para cada fold y luego promediar los resultados para obtener una estimación del resultado general.

La segunda consiste en concatenar todas las predicciones de cada fold para luego calcular la métrica global a partir de ese conjunto concatenado de predicciones.

Se dividió el dataset en 5 folds manteniendo la proporción de las clases en cada uno con el fin de conservar la representatividad de cada fold a la par de conseguir métricas más estables y realistas, ya que métricas sensibles al desbalance de clase, como la *AUPRC*, pueden variar mucho entre los folds si la proporción de clases varía.

Permutación	**Accuracy (T)**	**Accuracy (V)**	**AUPRC (T)**	**AUPRC (V)**	**AUC ROC (T)**	**AUC ROC (V)**
1	0.817	0.722	0.645	0.433	0.750	0.671
2	0.806	0.667	0.654	0.342	0.841	0.564
3	0.828	0.589	0.647	0.308	0.802	0.471
4	0.833	0.689	0.655	0.350	0.806	0.575
5	0.808	0.667	0.682	0.398	0.835	0.618
Promedios	0.818	0.666	0.657	0.366	0.807	0.580
Global	(NO)	0.667	(NO)	0.379	(NO)	0.585

Tabla 1: **Resultados de las métricas de evaluación por fold.** Se reportan Accuracy, AUPRC y AUC ROC tanto en entrenamiento (T) como en validación (V).

En nuestro trabajo observamos que ambas formas de estimar las métricas (sobre el promedio de los folds y la estimación global) produjeron estimaciones muy similares, difiriendo en casi una décima. Además, parece observarse que ninguna de las métricas calculadas muestra una gran variabilidad entre los folds si vemos solo el caso de entrenamiento. Pero en validación, vemos que para las 3 métricas existe una mayor variabilidad comparado con el caso de entrenamiento. Es por eso que utilizamos como resumen el promedio de los folds y el global para confirmar similitudes entre folds. Es evidente que el modelo tenga mayor variabilidad en la validación ya que son datos que no había visto anteriormente.

También observamos discrepancias entre las métricas obtenidas en los conjuntos de entrenamiento y validación, donde los valores correspondientes al conjunto de entrenamiento (calculados como el promedio entre los folds) resultan sistemáticamente más altos que los del conjunto de validación. Esto es algo esperable, ya que al evaluar las métricas sobre los conjuntos de entramiento, el modelo está siendo evaluado sobre datos que ya vió –y probablemente haya memorizado.

2.3. Combinación de parámetros para árboles

Para este inciso, se definió una grilla de valores para los hiperparámetros del árbol de decisión, variando la altura máxima entre 3, 5 e infinito (sin restricción) y el criterio de división entre Gini y Entropy, como se observa en la tabla 2.

Una vez definidos los valores posibles de los hiperparámetros, se ajustó un árbol de decisión para cada combinación de los mismos. Luego, se estimó la accuracy en los conjuntos de entrenamiento y validación utilizando validación cruzada con $K = 5$ folds (de la misma manera que en el inciso anterior).

En nuestra experimentación observamos poca diferencia entre los criterios de ganancia. A grandes rasgos, el criterio de Gini parece brindar resultados ligeramente superiores en general, independientemente de la altura del árbol.

Por otro lado, se evidencia una tendencia decreciente en la accuracy del conjunto de validación al aumentar la altura del árbol (especialmente cuando no se impone límite). Este patrón es opuesto al observado en el conjunto de entrenamiento, donde la accuracy aumenta con la altura del árbol, denotando un overfitting progresivo.

Altura	**Criterio**	**Validation Accuracy**	**Training Accuracy**
3	Gini	0.667	0.818
5	Gini	0.667	0.918
Infinito	Gini	0.656	1.00
3	Entropy	0.667	0.784
5	Entropy	0.629	0.903
Infinito	Entropy	0.620	1.00

Tabla 2: **Combinaciones de hiperparámetros para árboles de decisión.** Se reporta accuracy en validación y entrenamiento.

Esta tendencia a obtener mejores scores en el conjunto de entrenamiento mientras empeoran en el de validación sugiere que el aumento irrestricto de la altura de los árboles favorece el sobreajuste. Una explicación posible es que, al no limitar la complejidad del modelo, el árbol termina realizando divisiones demasiado específicas obteniendo una clasificación perfecta del conjunto de entrenamiento adaptándose al comportamiento aleatorio de los datos de entrenamiento (los memoriza) perdiendo poder predictivo para datos nuevos.

2.4.

Podemos concluir, a partir de los resultados obtenidos, que en la práctica es habitual que las métricas medidas sobre el conjunto de entrenamiento tiendan a ser más optimistas que las calculadas sobre el conjunto de validación. Sin embargo, cuando la diferencia entre ambas es significativa, puede ser un indicio claro de sobreajuste.

Una forma de evitar un modelo sobreajustado es detectar una configuración de hiperparámetros que reduzca el sobreajuste y optimice el desempeño sobre el conjunto de validación. En el caso de los árboles de decisión, observamos que la altura máxima es un factor causante de sobreajuste, por lo que encontrar un valor adecuado para este hiperparámetro será crucial.

También señalamos que calcular las métricas como el promedio de los valores obtenidos en cada fold no siempre proporciona estimaciones estables, especialmente cuando la cantidad de datos es reducida. Al dividir el conjunto en múltiples folds, puede ocurrir que cada uno contenga una baja cantidad de muestras de una clase en particular, lo que incrementa la variabilidad de las métricas obtenidas entre los folds. En contraste, al calcular la métrica sobre la concatenación de las predicciones de todos los folds, se incrementa la cantidad de ejemplos de la clase minoritaria, lo que permite obtener una medición más estable y realista.

Por último, concluimos que, en el conjunto de datos utilizado, las métricas *Accuracy, AUPRC y AUC ROC* parecen comportarse adecuadamente cuando se estiman como el promedio de los resultados obtenidos en cada fold mediante validación cruzada.

3. Comparación de algoritmos

La idea es comparar diversos algoritmos de marchine learning tales como arboles de decisión, KNN, SVM, LDA, Naïve Bayes y Random Forest, así como hallar una combinación optima de hiperparámetros para cada una.

3.1. Decision Trees, KNN & SVM

Model	Parameter	Values
Decision Tree	max_depth	`range(1, 20)`
	criterion	`[gini, entropy, log_loss]`
	min_samples_leaf	`range(1, 20)`
	max_features	`[sqrt, log2, x_desarrollo.shape[1], None]`
	splitter	`[best, random]`
	class_weight	`{0: 1, 1: x} for x in [1, 1.1, ..., 2]` y variantes
KNN	n_neighbors	`range(1, 75)`
	weights	`[uniform, distance]`
	p	`[1, 2, 3]`
	algorithm	`[auto, ball_tree, kd_tree, brute]`
	leaf_size	`range(1, 100)`
	metric	`[euclidean, manhattan, minkowski]`
SVM	kernel	`[linear, poly, rbf, sigmoid]`
	C	`np.logspace(-4, 2, 800)`
	degree	`range(1, 6)`
	gamma	`np.logspace(-4, 2, 300)`

Tabla 3: **Grilla de parámetros** para RandomizedSearchCV con $cv = 5$, $n\_iter = 1500$.

Los parámetros elegidos se encuentran la tabla 3. Con respecto al árbol de decisión, acotamos max_depth a 20 porque la performance no mejora con valores mas altos como se muestran el los gráficos de la sección Diagnostico Sesgo-varianza 4. Probamos distintas combinaciones de class_weight debido a que los datos están desbalanceados (aproximadamente el 30 % son de clase positiva). El resto de los parámetros están pensados para abarcar una gran cantidad de combinaciones posibles. Debido a la falta de información sobre los datos, este enfoque se aplica para los otros modelos.

El parámetro más relevante de KNN es n_neighbors, el cual, fijamos hasta 75 vecinos porque para valores más grandes, es esperable que una instancia de clase positiva (minoritaria) se encuentre cerca (en el sentido de vecindad) de instancias de clase negativa.

El algoritmo SVM se basa en un kernel que depende de la distribución de los datos respecto de las clases que no conocemos. Ademas, los datos tienen una dimensión muy alta ($p = 200$). Es por ello que decidimos testear todos los kernels disponibles. Elegimos escala logarítmica para gamma y C (cantidad de regularización) porque lo recomienda scikit-learn (véase https://scikit-learn.org/stable/modules/svm.html#svm-kernels).

Luego, ejecutamos RandomizedSearchCV con 1500 iteraciones cada algoritmo. Cabe mencionar que nos dimos el lujo de probar muchas configuraciones dado que el tamaño de dataset de desarrollo es chico ($\sim$ 450) y por ende, una complejidad temporal manejable ($\sim$ 2 minutos con una CPU Ryzen 5 5600G).

Modelo	Mejores Hiperparámetros	Mejor Score (AUCROC)
Decision Tree	{splitter: best, min_samples_leaf: 19, max_features: 200, max_depth: 9, criterion: entropy, class_weight: {0: 1, 1: 2}}	0.6900
KNN	{weights: distance, p: 1, n_neighbors: 17, metric: manhattan, leaf_size: 47}	0.8832
SVM	{kernel: rbf, gamma: 0.0001, degree: 3, C: 6.6226}	0.9257
LDA	Default	0.7475
GaussianNB	Default	0.7714
BernoulliNB	Default	0.7188
Dummy	Default	0.5

Tabla 4: **Comparación de algoritmos.** Se muestran los 7 algoritmos testeados junto con la mejor combinación de sus hiperparámetros y su score asociado (AUCROC). El modelo Dummy fue agregado como referencia del modelo mas simple posible

A partir de la tabla 4, observamos que el modelo SVM alcanza el mejor desempeño con un score de 0,9257, seguido por KNN con 0,8832, mientras que el árbol de decisión obtiene un score considerablemente menor de 0,6900. Esta diferencia puede atribuirse a que los modelos KNN y SVM capturan mejor la estructura subyacente de los datos, mientras que el árbol de decisión, con una profundidad acotada, presenta un sesgo mayor como se verá mas adelante en la curva de complejidad del árbol de decisión.

3.2. LDA & Naïve Bayes

Para comparar los resultados obtenidos con LDA y Naïve Bayes, es fundamental considerar el sesgo inductivo que introduce cada modelo. Sea $X = (X_1, \ldots, X_p)$ una vector aleatoria de dimensión $p = 200$ que representa a la secuencia de genes, y sea $k \in \{0, 1\}$ la clase del problema, donde 1 significa buen pronostico y 0 un mal pronostico. LDA asume que $X|_{Y=k} \sim \mathcal{N}(\mu_k, \Sigma)$ para cada k, es decir, los datos de cada clase provienen de una distribución normal multivariada con matriz de covarianza común entre clases, permitiendo correlación entre las variables.

En cambio, Naïve Bayes asume independencia condicional entre las variables, es decir, $X_i|_{Y=k} \perp X_j|_{Y=k}$ para todo i, j, k. Dado que en este caso las variables representan genes, es razonable esperar que no sean completamente independientes, lo que podría explicar el bajo desempeño observado en este modelo. Más específicamente, GaussianNB asume que $X_i|_{Y=k} \sim \mathcal{N}(\mu_{i,k}, \sigma^2_{i,k})$, mientras que BernoulliNB asume que $X_i|_{Y=k} \sim \mathcal{B}(p_{i,k})$. Claramente, esta última condición no se cumple, ya que los datos son continuos, lo cual puede justificar por qué GaussianNB obtuvo un mejor score que BernoulliNB (véase tabla 4).

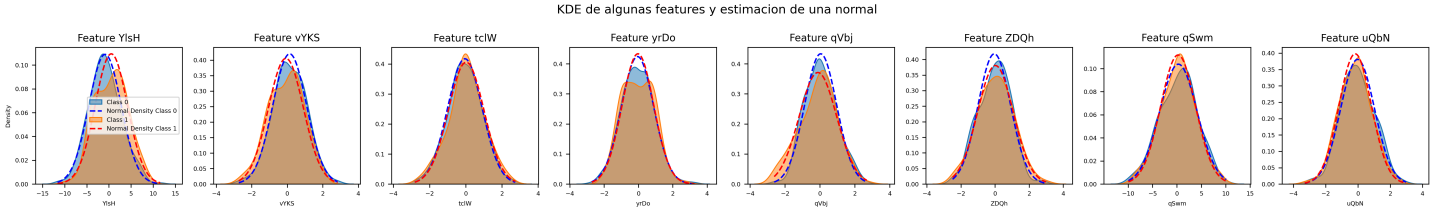

Figura 2: **Distribuciones de genes seleccionados**. Seleccionamos 8 features al azar y graficamos su kernel density estimate (KDE) correspondiente para visualizar su distribución por clase, superpuesta en lineas punteadas está la densidad de una normal con sus parámetros estimados. Muestra como algunas features (las primeras dos y la cuarta) no siguen estrictamente una distribución normal y tienen problemas en los picos. Este gráfico fue generado con los datos de desarrollo.

Volviendo a LDA, sabemos que si $X|_{Y=k}$ tiene distribución normal multivariada, entonces cada componente marginal también sigue una distribución normal. Sin embargo, esto no se verifica en la figura 2 para algunas componentes, como por ejemplo $\texttt{YisH}|_{Y=1}$ y $\texttt{vYKS}|_{Y=1}$, lo cual podría explicar por qué el modelo no obtuvo buenos resultados.

3.3. Mejor Modelo

En conclusión, el modelo SVM obtiene el mejor desempeño (Tabla 4), probablemente debido a su capacidad para modelar fronteras de decisión complejas mediante el uso del Gaussian kernel (*rbf*). Vimos en la figura 2 que un conjunto aleatorio de features siguen aproximadamente distribuciones normales (salvo los picos), lo cual fortalece que el Gaussian kernel esté en la mejor configuración de hiperparámetros. Además, el valor óptimo de $C = 6{,}6226$ sugiere que no más de 7 instancias fueron clasificadas incorrectamente o se ubicaron en el lado equivocado del margen de separación.

KNN también muestra un rendimiento notable, lo que sugiere que la información local entre observaciones similares es relevante para la clasificación en este problema. Es interesante notar que, a excepción de Decision Trees, un enfoque modelos discriminativos {SVM, KNN} resultó mas eficiente que los modelos generativos {LDA, Naïve Bayes} para estos datos.

4. Diagnóstico Sesgo-varianza

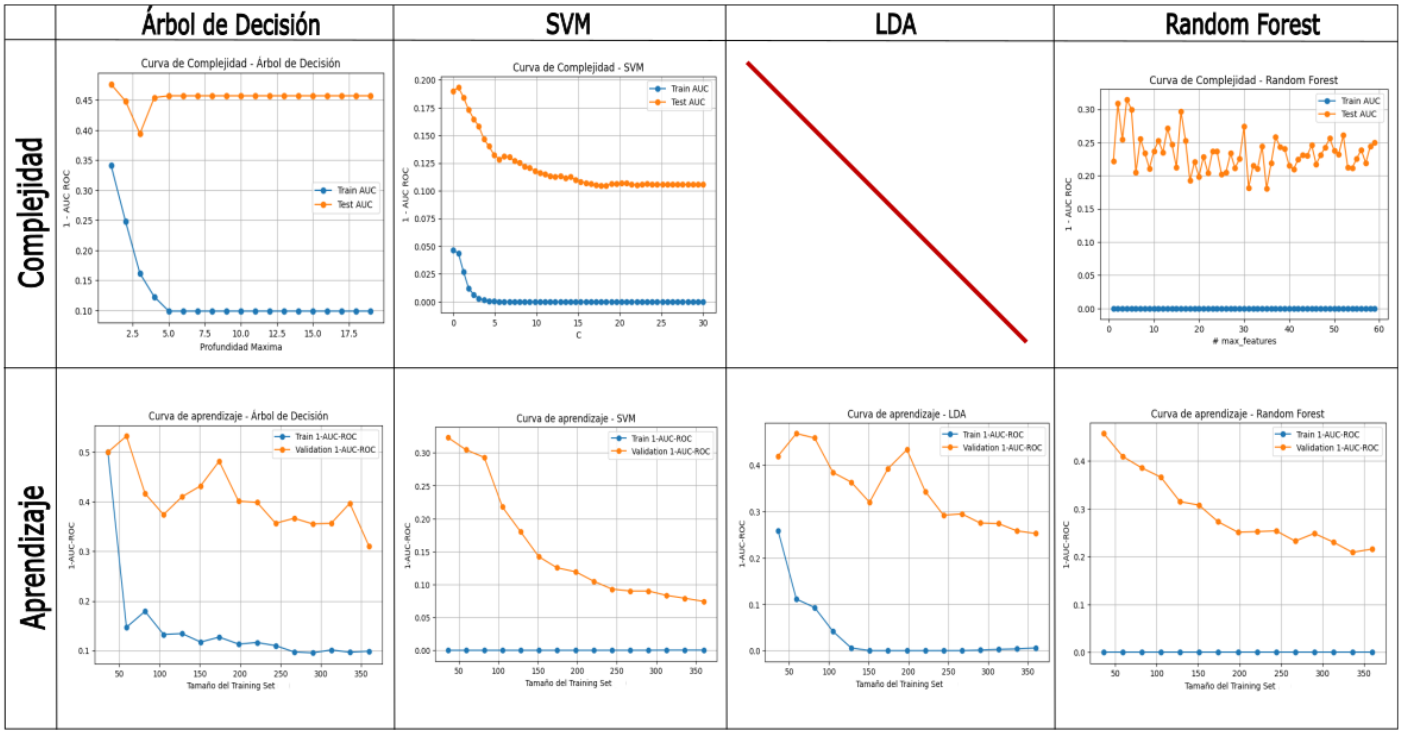

Figura 3: **Herramientas de diagnóstico para Árboles de Decisión, SVM, LDA y Random Forest.**

4.1. Curvas de Complejidad

Analizaremos para árboles de decisión y SVM, cómo los hiperparámetros `profundidad máxima` $\equiv$ `D` y `C` respectivamente afectan al sesgo y la varianza. Notemos que el sesgo en estos gráficos (Figura 3) se puede visualizar a través del error (que se comporta como 1 - AUCROC) en entrenamiento y en validación como función de la complejidad del modelo, de manera que modelos sencillos muestran baja varianza pero alto sesgo (alto error en validación y en training) y modelos más complejos muestran alta varianza pero bajo sesgo (bajo– o nulo error en training y alto en validación). Para árboles de decisión, vemos que a partir de `D = 5` tanto el error en validación como en entrenamiento no cambian. Esto es porque se alcanzó la profundidad máxima que el árbol podía aprender, y todos los árboles con el parámetro mayor o igual a 5, resultan en el mismo árbol con `D = 5`. Una vez notado esto, podemos analizar ese recorte del gráfico `1 <= D <= 5`. Vemos para valores chicos de este hiperparámetro (1-2) el error en validación y en entrenamiento es alto, mostrando un alto sesgo. Cuando el error en entrenamiento baja, el sesgo del modelo disminuye. En cuanto a la varianza, en estos valores donde el sesgo disminuye podemos ver que se acrecienta la misma, pues aunque el error en entrenamiento haya bajado drásticamente, el error en validación aumenta. Para SVM vemos algo parecido a lo que ocurre en el gráfico de árboles de decisión con `1 <= D <= 5`; las curvas de error varían de la misma manera al aumentar la complejidad. Para valores bajos de C, el modelo presenta baja varianza pero alto sesgo por los valores elevados de error en train y validación, mientras que al aumentar C, el sesgo del modelo disminuye pero aumenta la varianza.

4.2. Curvas de Aprendizaje

Si ponemos el foco en las 3 de curvas de aprendizaje (Figura 3) para árboles de decisión, SVM y LDA, vemos que cualquiera de ellas podría seguir mejorando su error en validación. No hay indicios de que hayan empezado a empeorar estos números, incurriendo en overfitting. Por lo tanto estaríamos en condiciones de afirmar de que podemos seguir agregando datos de entrenamiento para mejorar la performance de estos modelos hasta toparnos con un rebote en la curva de validación.

4.3. Caso Random Forest

El parámetro `max_features` funciona estableciendo un subconjunto aleatorio de igual tamaño que su valor, para features a las que puede explorar cada árbol dentro de sus cortes en cada iteración. Notamos en una primera instancia que este gráfico es distinto a los de SVM y decision trees, pues el error en validación no es monótonamente decreciente y luego creciente a partir de cierto punto. Sucede que esta curva presenta mayores variaciones, debiéndose en mayor parte a que el parámetro max features selecciona *al azar* las features que pueden ver los árboles, lo que puede provocar que pocos árboles (o ninguno) encuentren a las features (o su combinación) que sean más explicativas. En comportamiento macro, vemos que para un número bajo, `max features <=20` la performance en validación tiende a bajar (aunque no significativamente) mientras que

para `max features >=30` esta también muestra una leve tendencia a subir. Es a partir de estas secciones que podemos realizar un análisis similar al de los anteriores gráficos de complejidad; el rendimiento progresa hasta un rango de valores críticos del hiperparámetro, hasta que comienza a empeorar luego de pasar este rango. De manera que durante este progreso, el sesgo baja y la varianza aumenta. Al ver el gráfico de aprendizaje (Figura 3), notamos que también hay lugar para intentar agregar más datos al entrenamiento y así mejorar aún más la performance. Pues como podemos ver en el gráfico, aún no se ha alcanzado el rebote en validamiento. También habría indicios para pensar, debido al desaceleramiento del error en función de los datos de aprendizaje, que se estaría cerca de ese posible punto crítico.

5. Evaluación de performance

En esta etapa final, seleccionamos el modelo SVM por ser el que alcanzó el mayor AUC ROC en la comparación de algoritmos (Tabla 4) y por exhibir una estabilidad en sus métricas al ajustar el hiperparámetro de regularización C sin que se observe un deterioro significativo de la curva ROC a medida que cambia su complejidad (Figura 3). Entrenamos el SVM con su mejor configuración de hiperparámetros sobre el conjunto de desarrollo y, al evaluarlo en los datos de prueba, obtuvimos un AUC ROC de 0.9314 (Figura 4), demostrando así una sólida capacidad predictiva. Finalmente, usamos este valor para estimar las probabilidades de las instancias de competencia (es decir, usando los datos separados para test) y exportamos las predicciones a un archivo CSV para su posterior evaluación en el conjunto held-out.

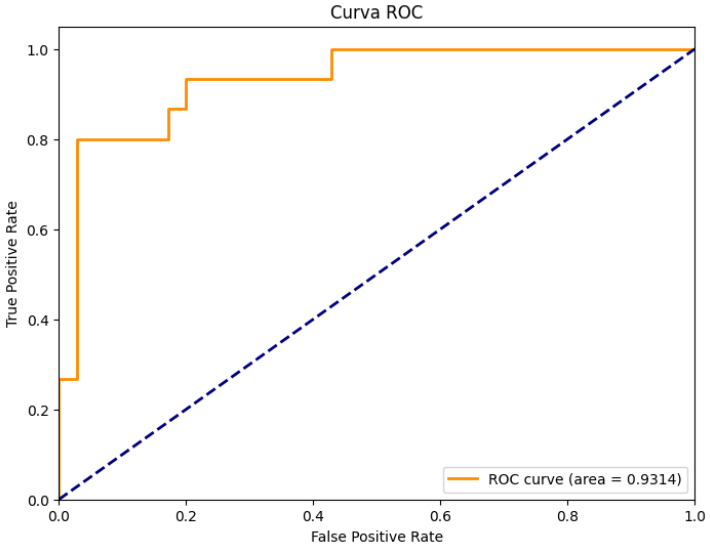

Figura 4: **Curva ROC para SVM con los mejores hiperparámetros.**

6. Conclusiones

6.1. Sobre las Métricas y la Evaluación

Del trabajo realizado se puede concluir que uno de los factores principales que complejizan la evaluación de modelos es la carencia de métricas sencillas y fácilmente generalizables para todo tipo de problema. Por ejemplo, en este trabajo, las métricas de accuracy pueden resultar poco representativas del rendimiento real de los modelos al evaluarse sobre una base de datos significativamente desequilibrada.

Por otro lado, resulta interesante analizar la dependencia de las métricas respecto al conjunto de datos de evaluación. Es esperable que si un modelo posee una buena comprensión de los patrones subyacentes en los datos, su rendimiento se mantenga relativamente estable al modificar el conjunto de datos de evaluación. Por lo tanto, es relevante considerar la variabilidad del rendimiento de los modelos sobre distintos conjuntos de datos empleando técnicas de diagnóstico como las curvas de aprendizaje utilizadas en la Sección 4 (véase la Figura 3). Allí se observó que los modelos de SVM y Random Forest exhiben una notable disminución en la variabilidad en comparación con los otros modelos analizados.

En nuestro trabajo también se evidencia la marcada dependencia de las métricas respecto al conjunto de datos en la Sección 2. En la Tabla 1 se puede apreciar la amplia diferencia entre las métricas obtenidas sobre el conjunto de prueba y sobre el conjunto de evaluación, producto del fenómeno de sobreajuste. En este escenario, el modelo logra identificar patrones muy específicos de los datos de entrenamiento, pero pierde capacidad para generalizar a conjuntos de datos nuevos.

Este fenómeno es particularmente notorio en la Tabla 2, donde se observa cómo, al aumentar la complejidad del modelo de árbol de decisión, aparentemente mejora su rendimiento al analizar la métrica medida sobre el conjunto de datos de entrenamiento. Sin embargo, su rendimiento parece deteriorarse al medirse sobre el conjunto de validación. Esto sugiere que

complejizar los modelos no necesariamente se correlaciona con una mejora en su capacidad predictiva (como puede apreciarse en la Figura 3 en la curva de complejidad del árbol de decisión y Random Forest), e incluso puede empeorar el rendimiento al disminuir su generalidad.

6.2. Sobre el Sesgo Inductivo y la Distribución de los Datos

Al trabajar con variables que representan genes, sabemos que muchas de ellas presentan correlaciones complejas y, por lo visto en la Figura 2, distribuciones aproximadamente normal en cada clase. Esta característica de la naturaleza de los datos ayuda en la justificación de la eficacia de modelos que asumen una estructura multivariante continua (e.g., LDA o SVM con kernel gaussiano) frente a métodos que imponen independencia condicional (Naïve Bayes), los cuales subestiman las dependencias entre genes y muestran peor desempeño o modelos que asumen fronteras de decisión rectangulares (Decision Trees). Notamos que resulta crucial conocer los bias inductivos de cada algoritmo, además de realizar análisis exploratorios sobre los datos, si no se tiene conocimiento de dominio, para comprender más profundamente el desempeño de los mismos.

El carácter desbalanceado del dataset dado —con una clase minoritaria y otra mayoritaria— obliga a realizar la partición de entrenamiento y evaluación de manera estratificada para preservar la proporción de clases. Este paso no solo potencia la representatividad de las métricas (evitando escenarios en los que una clase desaparezca del split), sino que también influye en el comportamiento de los algoritmos: aquellos con sesgos inductivos sensibles al balance (como árboles de decisión sin restricción de profundidad) tienden al sobreajuste cuando la partición (principalmente en los folds, como vimos en la Sección 2) no refleja la distribución real.

6.3. Sobre la Búsqueda de Hiperparámetros

Si bien una complejidad excesiva en los modelos puede perjudicar su rendimiento, es necesario contar con un modelo suficientemente complejo para poder reconocer patrones en los datos. Por esta razón, la elección óptima de hiperparámetros para un modelo dado resulta fundamental. Encontrar el rendimiento óptimo global en el espacio de hiperparámetros es una tarea compleja, pero es posible generar heurísticas eficientes que guíen hacia óptimos locales, como es el caso de lrandomized grid search empleada en el ejercicio 3 del trabajo. No obstante, este método también estará influenciado por las métricas de evaluación seleccionadas y el conjunto de datos sobre el que se construyó y evaluó el modelo.