DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico: Aprendizaje Supervisado

Clasificación de expresiones genómicas

Aprendizaje Automático

Grupo II

Integrante	LU	Correo electrónico
Montserrat, Luz	541/21	luzmont@ymail.com
Calloni, Sol	473/21	solcicalloni@gmail.com
Sanchez, Jhosept	813/21	levyt22@gmail.com
Duran, Valentina	974/21	valentinad01@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

Índice

0. Introdución

El siguiente trabajo consiste en la detección temprana y el pronóstico preciso de enfermedades como el cáncer a través de modelos de machine learning. El enfoque principal recae en un conjunto de datos único que abarca mediciones de ARN de 200 genes, obtenidas de pacientes con hiperplasia, una condición precursora del cáncer.

La hiperplasia es un fenómeno marcado por un crecimiento celular anormal y descontrolado que representa el punto de partida clave de este análisis. La pregunta fundamental que guía nuestra investigación es: ¿Qué determina la transición de células hiperplásicas a células cancerosas en algunos casos, mientras que en otros no?

Para abordar esta interrogante, se ha llevado a cabo un estudio exhaustivo que recopila muestras de diversos tipos de hiperplasias, provenientes de 500 pacientes con antecedentes familiares y lesiones pre-tumorales. A lo largo de un período de cinco años, se ha monitoreado la evolución de este grupo de pacientes en busca de indicios de neoplasias o hiperplasias más agresivas. El resultado de este último trabajo fue la creación de un biobanco de datos que almacena mediciones detalladas, clasificadas según el pronóstico de cada muestra: favorable en ausencia de recurrencia, y desfavorable en caso de recaída.

El foco de nuestro análisis se centra en un panel específico de 200 genes cuya expresión se cree que desempeña un papel crucial en la transformación tumoral, lo que ofrece nuevas perspectivas para comprender y abordar la progresión del cáncer.

1. Ejercicio 1: Separación de datos

En esta primera parte del trabajo lo que hicimos fue realizar dos particiones del conjunto de datos dado, llamadas desarrollo y evalución, es decir, los datos que usaremos para entrenar y los que usaremos para testear repectivamente.

Estamos asumiendo que la distribución de cada clase representa la realidad. Por lo tanto, es conveniente tanto entrenar como evaluar nuestros modelos en datos que mantengan esta proporción, lo que garantiza una evaluación más precisa del rendimiento del modelo. Observamos entonces la proporción de cada clase obteniendo lo siguiente:

Etiqueta	**Proporción**
0	0.686
1	0.314

Cuadro 1: Proporción de cada clase

Debido a que observamos un desbalance en las clases de los datos (ver Cuadro 1), se tomó la decisión de separarlos con la técnica *Stratified*, es decir, asegurarse de que las clases queden balanceadas con las proporciones originales al separar los datos.

Por otro lado, decidimos separar el 85 % de los datos para desarrollo (425 datos) y el 15 % (75 datos) para evaluación. Para elegir los datos, fuimos tomando instancias de manera aleatoria seteando una semilla para que sea reproducible. Esto lo hacemos para evitar mantener patrones que se puedan encontrar en los datos.

Una vez seleccionadas las instancias para desarrollo y evaluación, guardamos los dataframes correspondientes para su uso en las etapas siguientes del proceso de modelado.

2. Ejercicio 2: Construcción de modelos

A partir de este punto, siempre que mencionemos a "los datos" nos estaremos refiriendo a los datos de desarrollo.

En este ejercicio, el inciso 4 estará incluido en las conclusiones de las tablas.

2.1. Inciso 1

Para empezar a abordar el objetivo de encontrar el mejor modelo predictor, empezamos por entrenar un árbol de decisión con altura máxima 3. Para ello, separamos los datos de desarrollo en entrenamiento y test y calculamos los accuracies para los datos, lo que nos dió lo siguiente.

Datos	**Accuracy**
Entrenamiento	0.7906
Test	0.5814

Cuadro 2: Accuracy del modelo en cada conjunto de datos

Podemos observar que en los datos de entrenamiento el accuracy nos dió más que en los datos de test. Esto se debe a que estamos calculando el accuracy con los mismos datos que entrenamos el modelo.

2.2. Inciso 2

A continuación estudiamos más a fondo el modelo de árboles de decisión teniendo en cuenta diferentes métricas. Para ello, usamos todos los datos de desarrollo, es decir, juntamos los datos de entrenamiento y de test que separamos en el inciso anterior. Análogamente, decidimos utilizar stratified cross-validation pues las clases están desbalanceadas.

Para cada fold en cross-validation calculamos el accuracy, el área bajo la curva Precision-Recall ($AUPCR$) y el área bajo la curva ROC ($AUROC$) para los datos de entrenamiento y los de test. Luego, hicimos un promedio para cada métrica. Paralelamente, fuimos almacenando las predicciones de cada fold utilizando los restantes para entrenar el modelo. Con esto, conseguimos una predicción para cada instancia y así calculamos las métricas globalmente.

En la siguiente tabla podemos observar los valores de cada métrica promedio y global.

fold	accuracy_train	accuracy_test	aupcr_train	aupcr_test	aucroc_train	aucroc_test
0	0.835294	0.682353	0.460067	0.24724	0.78633	0.6206
1	0.835294	0.635294	0.460067	0.144322	0.78633	0.543677
2	0.826471	0.694118	0.446895	0.242079	0.775883	0.617497
3	0.841176	0.705882	0.469634	0.245316	0.786567	0.61622
4	0.838235	0.741176	0.464955	0.349095	0.794751	0.711367
promedio	0.835294	0.691765	0.460324	0.24561	0.785972	0.621872
global	**NaN**	0.691765	**NaN**	0.247725	**NaN**	0.622155

Cuadro 3: Resultados de stratified cross-validation

Observamos nuevamente que para las tres métricas evaluadas el rendimiento de los datos de entrenamiento superó consistentemente a los de test. Esto se debe a que al modelo le resulta más fácil predecir datos que fueron usados para su entrenamiento.

Se puede ver que el *accuracy* global y el promedio dieron igual en un valor de 0.69. Comparado con *AUCROC* y *AUPCR*, podemos observar que en los datos de entrenamiento el Accuracy dió mejor. Sin embargo, no vamos a tener tan en cuenta el valor dado por esta métrica ya que no nos habla sobre el tipo de error que está cometiendo el modelo.

Por otro lado, si recordamos el desbalance de las clases, una de ellas tenía 0.686 de proporción y el Accuracy en los datos de test dió en promedio 0.69, por lo que podría pensarse que el modelo está asignando a casi todos los datos esa misma clase. En consecuencia, consideramos que va a ser más informativo tener en cuenta otras métricas.

El *AUCROC* y el *AUPCR* globales y promedios también dieron muy similares, pero con una leve mejora en el global. En particular, el *AUPCR* para los datos de test en todos los *folds* se mantuvo consistentemente por debajo de 0.35, lo que indica un desempeño deficiente en la capacidad de clasificación. De manera similar, los valores del *AUPCR* para los datos de entrenamiento estuvieron cercanos a 0.46 en todos los casos. Esta observación sugiere que el modelo no logró alcanzar altos niveles de *precision* y *recall* al mismo tiempo. Dado que nuestro enfoque se centra en la clasificación de mal pronóstico versus buen pronóstico, es crucial minimizar los falsos positivos (clasificaciones incorrectas de buen pronóstico cuando en realidad era mal pronóstico), lo que implica maximizar el *precision*.

El *AUROC* para el conjunto de entrenamiento se mantuvo bastante constante en todos los *folds*, alrededor de 0.78. Sin embargo, observamos una variabilidad significativa en el conjunto de test, donde el valor máximo fue de 0.711367 y el mínimo de 0.543677. Dado que estamos tratando con un conjunto de datos desbalanceado, el *AUROC* parece ser una métrica más robusta y adecuada para evaluar el rendimiento del modelo en la clasificación binaria.

Comparado con el *AUPCR*, el *AUROC* ofrece una visión más completa del rendimiento del modelo. Mientras ambas métricas consideran los falsos positivos, la curva precision-recall no tiene en cuenta los verdaderos negativos. Esto significa que en un conjunto de datos desbalanceado, la curva precision-recall podría no reflejar adecuadamente la distribución de las clases. En contraste, la curva ROC tiene en cuenta tanto los verdaderos positivos como los verdaderos negativos, así como los falsos positivos y los falsos negativos. Por lo tanto, si el conjunto de datos está desbalanceado, esta información se reflejará en la curva ROC, brindando una evaluación más completa del rendimiento del modelo.

2.3. Inciso 3

Seguimos estudiando el caso de árboles de decisión variando los siguientes hiperparámetros:

- Altura máxima: 3, 5 y None.
- Criterios: gini y entropía.

Por lo tanto, creamos todos los pares posibles de altura máxima y criterios de corte anteriores utilizando la función ParameterGrid.

La función Grid Search implementada por nosotros toma cada uno de estos pares y ejecuta Stratified Cross Validation con k = 5. En este caso, como métrica utilizamos el accuracy promedio.

Veamos la siguiente tabla de resultados.

Altura máxima	Criterio de corte	Criterio	Accuracy (training)	Accuracy (test)
1	3	Gini	0.835294	0.691765
2	5	Gini	0.938824	0.654118
3	Infinito	Gini	1.0	0.642353
4	3	Entropy	0.805882	0.689412
5	5	Entropy	0.905882	0.661176
6	Infinito	Entropy	1.0	0.675294

Cuadro 4: Resultados de la clasificación según el criterio y la altura máxima

Se puede observar que en el caso particular de altura máxima 3 y criterio de corte gini, los valores de *accuracy* son iguales al *accuracy* promedio de la tabla anterior, tanto para el conjunto de entrenamiento como para la evaluación. Esto tiene sentido porque se utilizó la misma semilla, lo que implica que los *folds* son los mismos en ambas tablas.

Observamos que al aumentar la altura máxima de los árboles, el *accuracy* en el conjunto de entrenamiento aumenta para ambos criterios. Este incremento se debe a que a medida que la altura crece, el modelo se ajusta más a los datos.

Por otro lado, notamos que el *accuracy* en el conjunto de test permanece relativamente constante para los distintos modelos, a diferencia del *accuracy* en el conjunto de entrenamiento. Sin embargo, a diferencia de este último, no observamos un aumento en el *accuracy* de test a medida que crece la altura máxima del árbol para el caso de criterio gini. Esto sugiere que, en lugar de mejorar la capacidad del modelo para generalizar a nuevos datos, aumentar la altura del árbol puede simplemente conducir a un sobreajuste a los datos de entrenamiento. Para el criterio entropy, sorprendentemente observamos que en los datos de test, el accuracy primero disminuye levemente pero luego aumenta.

3. Ejercicio 3: Comparación de algoritmos

En este ejercicio exploramos distintos algoritmos de aprendizaje con diferentes configuraciones con el objetivo de encontrar el mejor modelo de cada familia. Como métrica de performance usamos AUROC resultante de 5-fold stratified cross-validation.

- **Árboles de decisión**

  Los valores de hiperparámetros probados para este algoritmo fueron los siguientes:

  - Altura máxima: uniforme_entera(1,10).
  - Criterios: gini, entropía, función de pérdida logarítmica.

  Dado que la altura máxima es un número entero, nos pareció conveniente utilizar una distribución uniforme de valores enteros entre los valores 1 y 10, dado que luego de probar muchos valores superiores notamos que los valores no variaban tanto luego de superar la altura 10. Además, sabemos que una altura máxima lógica para un árbol de decisión es $\log_2 425 = 8.73$ pues sería ir dividiendo al espacio en 2 en cada paso, es decir, crear una región para cada uno de los datos. Luego, para los criterios de corte, simplemente consideramos todos los que ofrecía sk-learn.[1]

  Veamos los mejores 5 modelos de árboles de decisión haciendo 15 iteraciones. La razón por la cual usamos esta cantidad se debe a que hay 30 pares posibles y no estamos buscando hacer Grid Search ni repetir pares innecesarios.

Promedio de AUROC	Hiperparámetros
0.646762	{'criterion': éntropy', 'max depth': 3}
0.646762	{'criterion': éntropy', 'max depth': 3}
0.637520	{'criterion': 'log loss', 'max depth': 3}
0.636871	{'criterion': 'gini', 'max depth': 3}
0.624257	{'criterion': 'gini', 'max depth': 3}

Cuadro 5: Los mejores 5 modelos para árboles de decisión

  Vemos que el primer y segundo mejor modelo es un árbol con altura máxima 3 y criterio de corte entropía. No nos sorprende que en los primeros 5 mejores modelos la altura que mejor dió sea 3 ya que concuerda con lo que vimos en el ejercicio anterior, a pesar de que sea otra métrica la que tuvimos en cuenta en ese caso. Se podría pensar que el mejor criterio de corte dió entropía ya que esta puede ser más efectiva en manejar conjuntos de datos con clases desbalanceadas. Debido a la forma en que se calcula, puede dar un peso más equitativo a todas las clases, lo que puede ser beneficioso cuando hay desequilibrios en la distribución de clases, como es en nuestro caso.

- **KNN**

  Los valores de hiperparámetros probados para este algoritmo fueron los siguientes:

  - Cantidad de vecinos: uniforme_entera(1,20)
  - Pesos: todos los vecinos tienen igual peso al votar o se le da peso al voto según la distancia
  - p : 1, 2. Es cómo se mide la distancia de minkowski. Cuando p = 1 es la distancia manhattan y cuando p = 2 es la euclidiana

[1] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Promedio de AUROC	Hiperparámetros
0.842864	{$'n\_neighbors'$: 6, 'p': 1, 'weights': '$distance$'}
0.839378	{$'n\_neighbors'$: 7, 'p': 1, 'weights': '$uniform$'}
0.830942	{$'n\_neighbors'$: 5, 'p': 1, 'weights': '$distance$'}
0.829063	{$'n\_neighbors'$: 16, 'p': 1, 'weights': '$uniform$'}
0.829063	{$'n\_neighbors'$: 16, 'p': 1, 'weights': '$uniform$'}

Cuadro 6: Los mejores 5 modelos para KNN

Dado que la cantidad de vecinos es un número entero, nos pareció conveniente nuevamente utilizar una distribución uniforme de valores enteros entre los valores 1 y 20. Para la forma de medir la distacia y los pesos simplemente probamos todas las opciones ofrecidas por sk-learn.[2]

En el Cuadro 6 vemos los mejores 5 modelos para KNN dados haciendo 30 iteraciones.

Observamos que el mejor modelo considera 6 vecinos, utiliza la distancia Manhattan y la votación se hace según la distancia. Se puede apreciar que en todos estos modelos siempre se considera la distancia Manhattan como distancia elegida. Esta calcula la distancia entre dos puntos sumando las diferencias absolutas entre las coordenadas de los puntos. Es especialmente útil cuando la dimensionalidad de los datos es alta, ya que la distancia Manhattan es menos sensible a estos factores en comparación con la distancia Euclidiana. En nuestro caso, tenemos 200 atributos, por lo que tenemos una dimensión bastante alta.

Con respecto a la cantidad de vecinos, vemos que en los tres mejores modelos no se aleja de 6.

Por otro lado, los primeros dos modelos son muy similares entre sí salvo por la forma de medir los pesos, y el promedio de AUROC dió muy parecido. Por lo tanto, no tenemos suficiente información para preferir una forma por sobre la otra.

- **Support Vector Machines**

  Los valores de hiperparámetros probados para este algoritmo fueron los siguientes:

  - C: uniforme(1,10)
  - Kernel: lineal, polinomial, radial y sigmoideal
  - Degree: aclara el grado del kernel polinomial. Tomamos desde 2 a 10, ya que en el kernel lineal tiene en cuenta el polinomial de grado 1.

  Veamos los mejores 5 modelos de SVM dados haciendo 40 iteraciones:

Promedio de AUROC	Hiperparámetros
0.891996	{'C': 4.343, 'degree': 4, 'kernel': rbf}
0.891876	{'C': 4.087, 'degree': 7, 'kernel': rbf}
0.891727	{'C': 5.685, 'degree': 8, 'kernel': rbf}
0.891490	{'C': 3.982, 'degree': 4, 'kernel': rbf}
0.879929	{'C': 1.686, 'degree': 6, 'kernel': rbf}

Cuadro 7: Los mejores 5 modelos para SVM

Podemos observar que el kernel que da en los primeros 5 mejores modelos es el kernel radial, por lo que el parámetro degree no se tiene en cuenta pues este se usaba junto al kernel polinomial. Con respecto

[2]https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

a los C, este hiperparámetro controla cuantas instancias se va a permitir que estén del lado equivocado del margen y/o hiperplano. Podemos ver que los mejores modelos tienen un valor cercano al 4, por lo que es un modelo bastante restrictivo.

- **Linear Discriminant Analysis**

  LDA no tiene hiperparámetros. Además, al probar con los parámetros que ofrecía sk-learn[3] no notamos diferencia en la performance de los modelos. Con lo cual, trabajamos con los parámetros dados por default por lo que solo hicimos un solo modelo que requirió una sola iteración. En este caso, el promedio de AUROC que dió fue 0.753419.

- **Naïve Bayes**

  Utilizamos el Naïve Bayes Gaussiano para trabajar. Los valores de hiperparámetros probados para este algoritmo fueron los siguientes:

  - Var Smoothing: uniform(0.000001, 0.001)
  - Priors: [0.68,0.32],[0.32,0.68],[0.85,0.15],[0.5,0.5],[0.15,0.85],[0.7,0.3],[0.3,0.7]

  Veamos los mejores 5 modelos de Naïve Bayes dados haciendo 20 iteraciones:

Promedio de AUROC	**Hiperparámetros**
0.822782	{'var_s': 0.000918, 'priors': [0.5, 0.5]}
0.820462	{'var_s': 0.000803, 'priors': [0.68, 0.32]}
0.819687	{'var_s': 0.000765, 'priors': [0.15, 0.85]}
0.818144	{'var_s': 0.000678, 'priors': [0.85, 0.15]}
0.818144	{'var_s': 0.000698, 'priors': [0.15, 0.85]}

Cuadro 8: Los mejores 5 modelos para Naive Bayes

El parámetro var_smoothing suaviza la matriz de covarianza, lo que ayuda a evitar problemas numéricos cuando hay características con varianzas muy pequeñas. Por otro lado, las priors son las probabilidades a priori de pertenecer a cada clase. Si no se especifican, estas probabilidades se ajustan automáticamente según los datos. En nuestro caso, probamos con distintas probabilidades, tanto las que se encuentran en nuestro conjunto de datos como otras opciones.

Dado que obtuvimos valores alrededor de 0.81 para el promedio de AUROC y este modelo asume que los atributos son independientes entre sí dada la clase, y con distribución normal, podríamos concluir que nuestros datos cumplen con estas hipótesis.

[3]https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

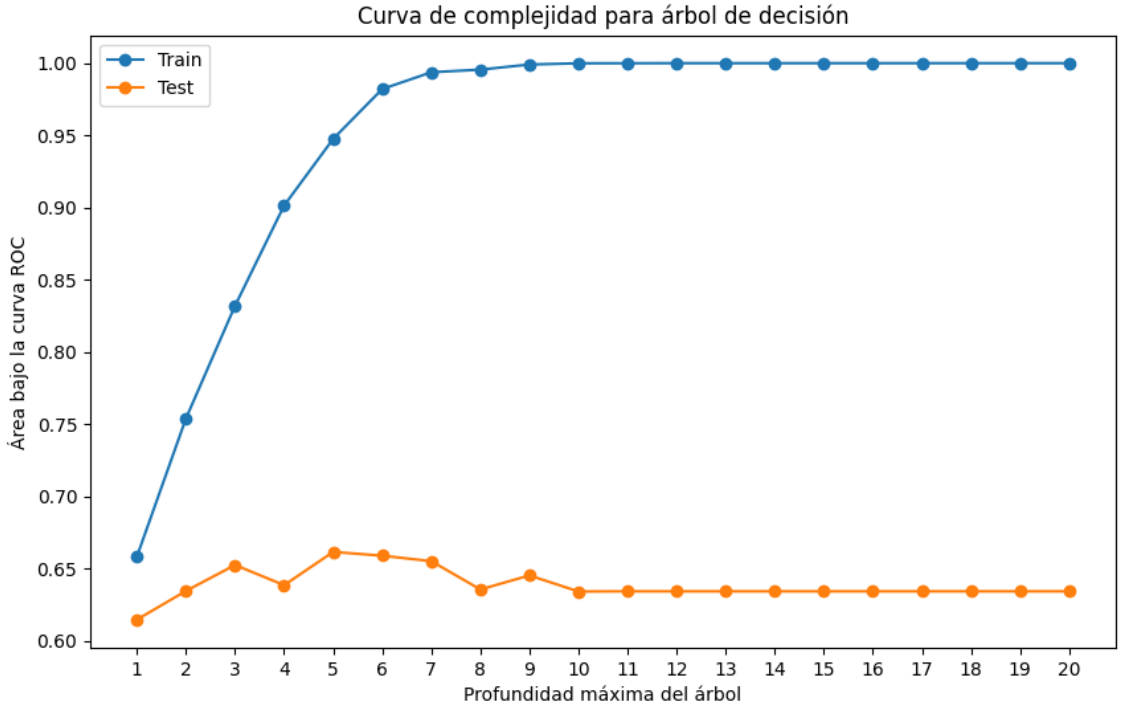
4. Ejercicio 4: Diagnóstico Sesgo-Varianza

4.1. Inciso 1 y 2

En estos dos incisos se nos pide por un lado graficar las curvas de complejidad para árboles de decisión y SVM, variando la profundidad en el caso de árboles, y el hiperparámetro C en el caso de SVM. Por otro lado, debemos graficar las curvas de aprendizaje para cada modelo. En base a estas curvas, sacar conclusiones sobre si los algoritmos parecen haber alcanzado su límite, o bien si aumentar la cantidad de datos debería ayudar.

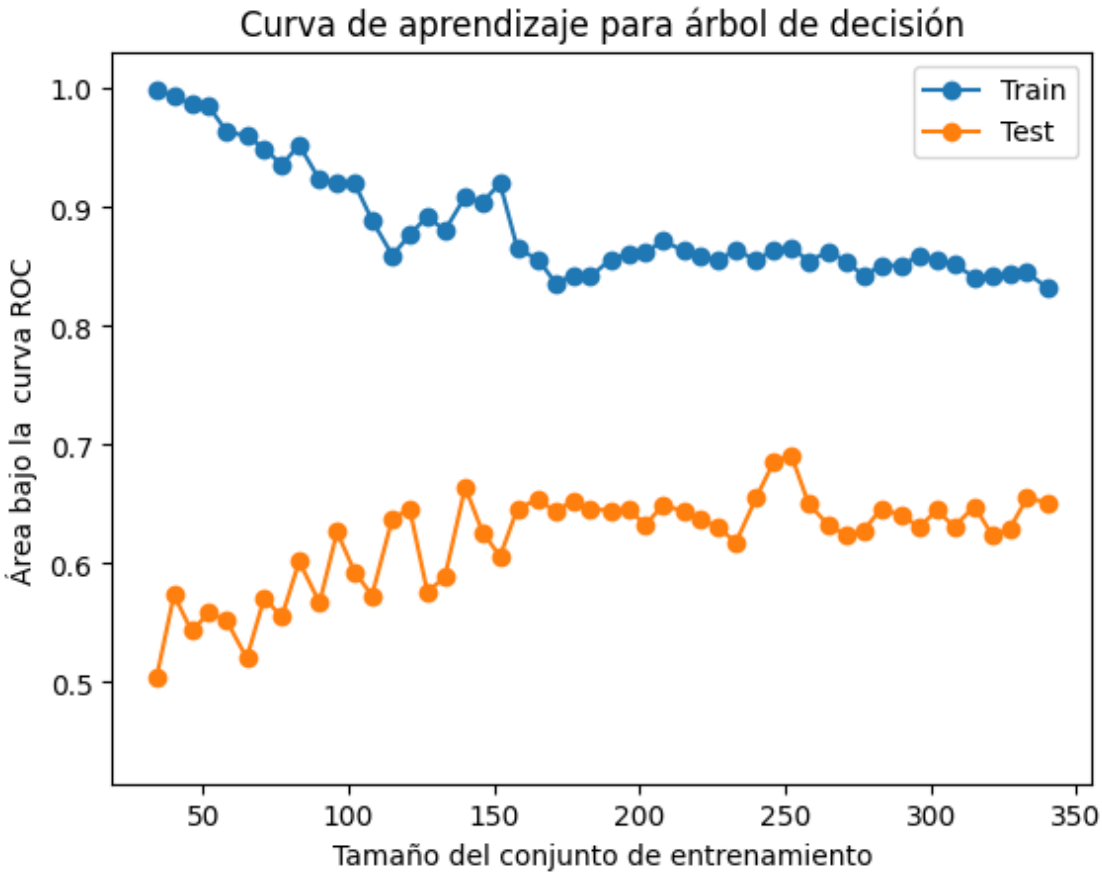
4.1.1. Árboles de decisión

Veamos primero la curva de complejidad de los árboles de decisión cuando vamos aumentando el valor de la profundidad máxima. Como nos había dado que el mejor modelo en este caso usa el criterio de corte entropy, fue el que usamos para el siguiente gráfico.



Podemos observar que tanto en los datos de entrenamiento como en lo de test, al llegar a una altura máxima de 11 las curvas se planchan. Esto se debe a que a partir de esta altura máxima, el árbol clasificó correctamente a todos los datos de entrenamiento (se ajusta demasiado a los datos y generaliza poco), por lo que al aumentar el parámetro de máxima altura nuestro árbol no cambia, y así se consigue la misma performance en los datos de test.

Por otro lado, veamos la curva de aprendizaje cuando aumentamos el tamaño del conjunto de entrenamiento siempre utilizando un árbol de altura máxima 3 y criterio de corte entropy, que es el que nos dió el mejor modelo para el caso de árboles de decisión.



Podemos observar que a medida que van aumentando los datos de entrenamiento, el área bajo la curva ROC en estos datos va disminuyendo, aunque a partir de cierto punto se empieza a planchar. Esto se debe a que debemos aproximar cada vez más datos, a pesar de que la complejidad del modelo no cambia. En el momento en el que se estanca, esto se debe a que los datos tienen una cierta representación, y estamos agregando datos ya similares, entonces no vamos a notar tanto la diferencia.

Por otro lado, podemos ver que el área bajo la curva en datos de test aumenta, y luego, al igual que con los datos de entrenamiento, se plancha la curva. Esto quiere decir que a pesar de aumentar la cantidad de datos utilizados para el entrenamiento, no nos produce un mejor modelo para la predicción de los datos de test.

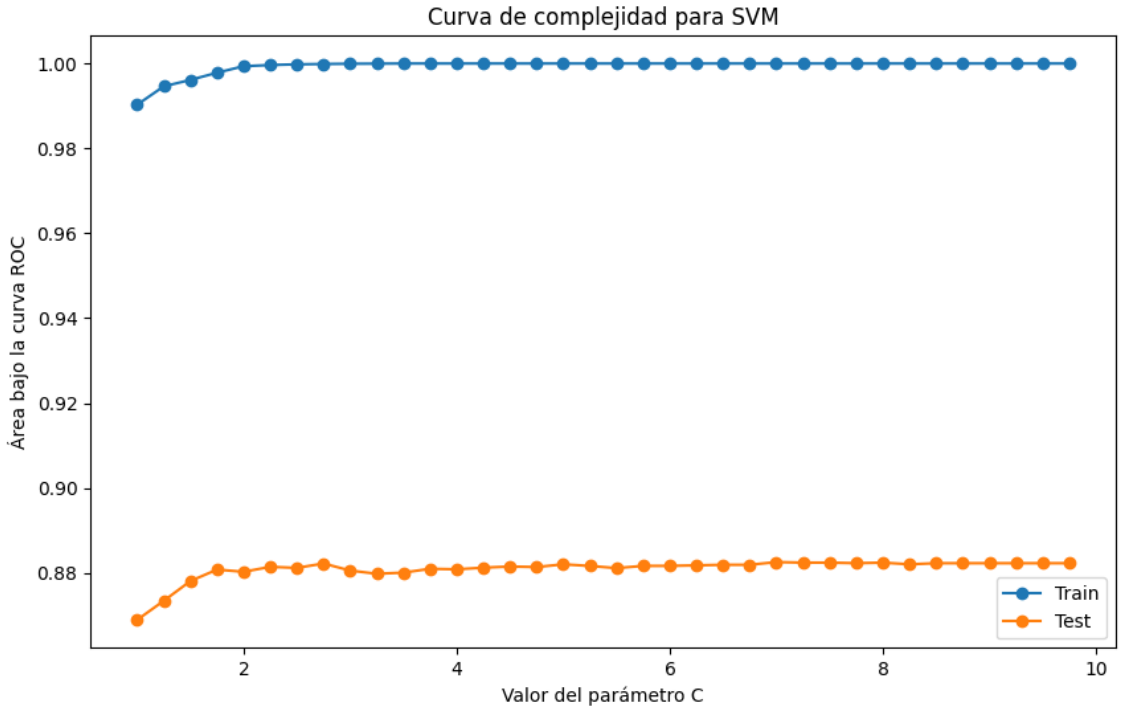
Mirando ambos gráficos, podemos hacer algunos análisis:

Si consideramos el sesgo, se podría decir que en este caso tenemos bastante, ya que mirando la curva de aprendizaje, las curvas de train y de test a pesar de plancharse al mismo tiempo, siguen separadas. Además, si miramos la curva de complejidad y tenemos en cuenta una profundidad máxima específica, notamos que hay una gran distancia entre el punto correspondiente a la curva de datos de entrenamiento y el punto de la de test.

Por otro lado, si consideramos la varianza, vemos que al principio de las curvas de aprendizaje pareciera haber bastante, pero a medida aumentamos la cantidad de datos, esto deja de ser así. Por lo tanto, podemos pensar que la varianza del comienzo se debe a la poca cantidad de datos de entrenamiento que tenemos.

4.1.2. Support Vector Machine

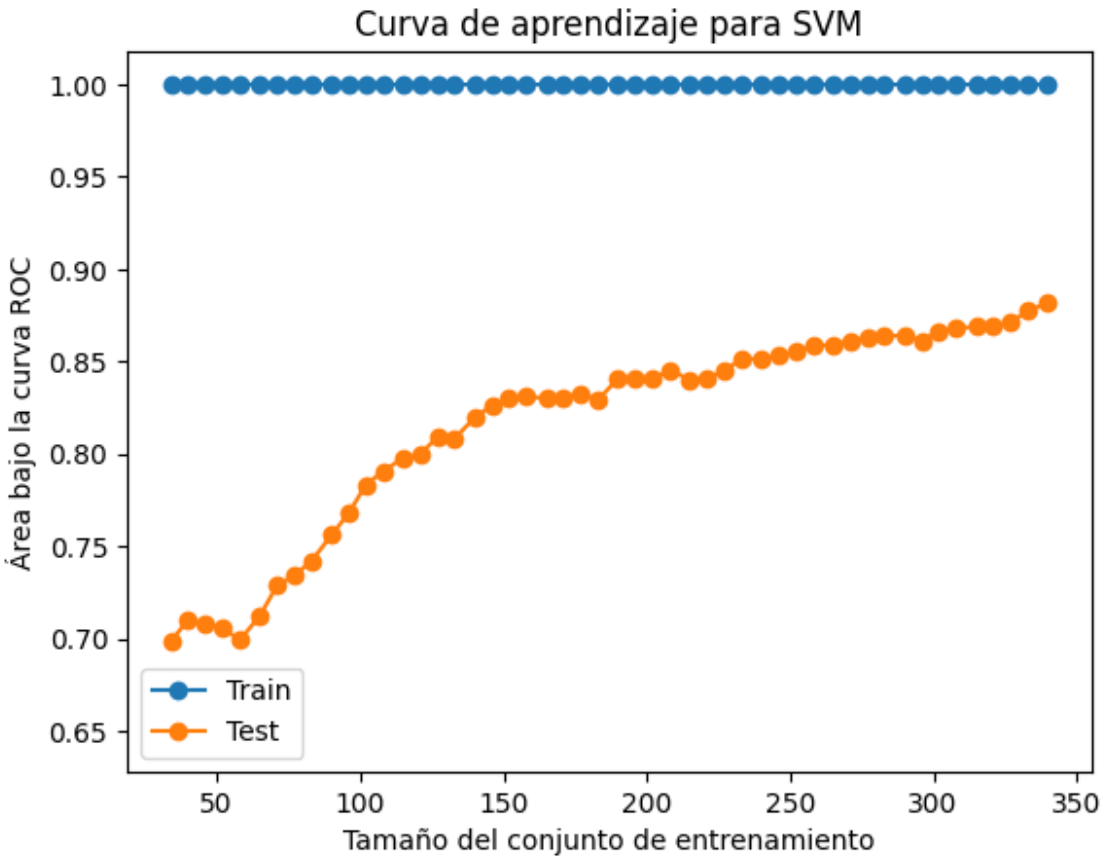
Veamos primero la curva de complejidad de SVM cuando vamos aumentando el valor del prámetro C. Como nos había dado que el mejor modelo en este caso usa el kernel radial, utilizamos este.



Podemos observar que ambas curvas, tanto la de entrenamiento como la de test, tienen una leve mejora al comienzo y luego se planchan rápidamente. La curva de entrenamiento se plancha en 1 aproximadamente con un valor del parámetro $C = 2$.

El parámetro C controla el compromiso entre sesgo y varianza en la técnica de aprendizaje estadístico. Cuando C es pequeño, buscamos márgenes estrechos que rara vez se violan; esto resulta en un clasificador altamente ajustado a los datos, lo que puede tener un sesgo bajo pero una varianza alta. Por otro lado, cuando C es mayor, el margen es más amplio y permitimos más violaciones; esto implica ajustar menos los datos y obtener un clasificador potencialmente más sesgado pero con una varianza menor.

Veamos ahora la curva de aprendizaje de SVM cuando tomamos un kernel radial y un valor del hiperparámetro C = 4.43 que es el del mejor modelo.

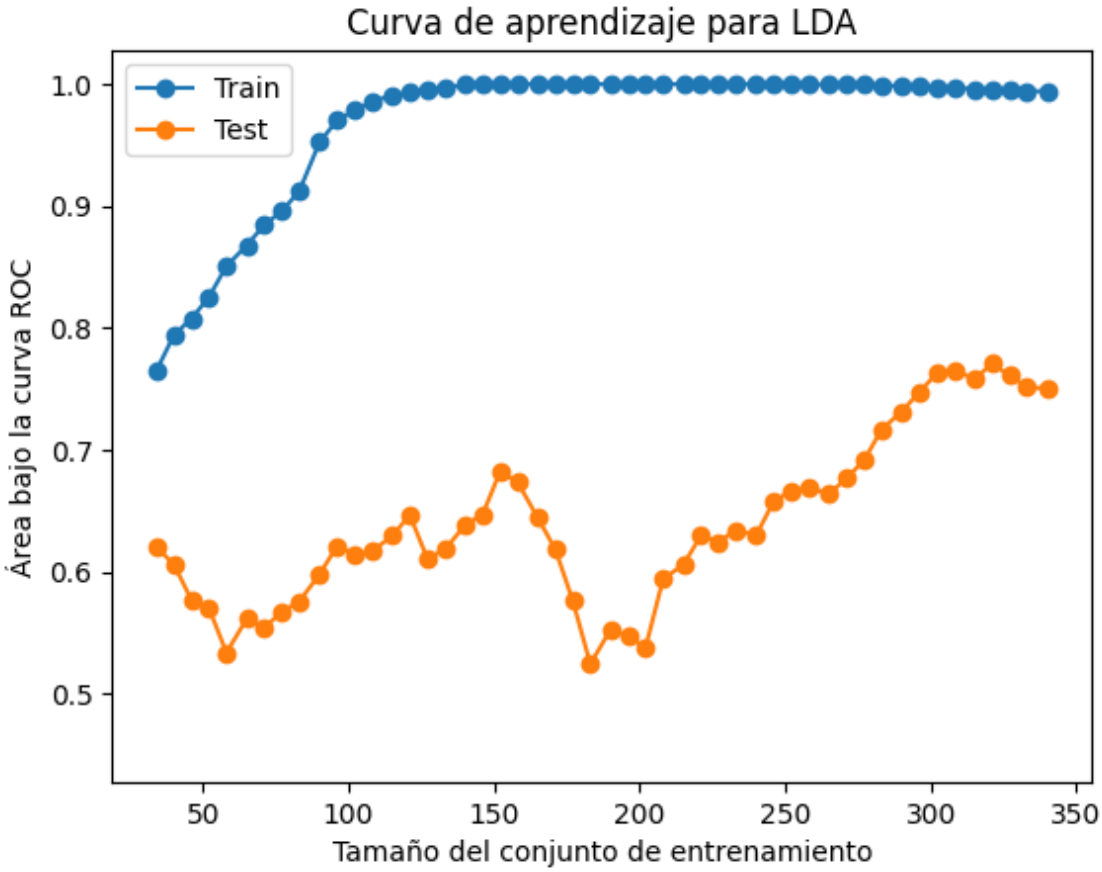


Podemos observar que para la curva de los datos de test, pasa lo mismo que con los árboles de decisión, empieza aumentando con más pendiente y luego se va planchando. Sin embargo, pareciera que podría seguir aumentando si agregamos más datos al conjunto de entrenamiento. Para la curva de los datos de training, vemos que esta está estancada en 1, es decir, no tiene error. Por lo tanto, a medida que aumentamos los datos de entrenamiento, el valor del área bajo la curva ROC es 1 siempre.

Por último, vemos que el gráfico coincide con lo que habíamos dicho anteriormente sobre el parámetro C. Como tenemos un valor relativamente chico, vemos que tenemos poco sesgo y, sin embargo, tenemos poca varianza.

4.1.3. Linear Discriminant Analysis

Para LDA no hicimos curva de complejidad porque no teníamos un hiperparámetro para aumentar. Sin embargo, si hicimos el gráfico de la curva de aprendizaje. Veamoslo a continuación.



Podemos observar que para la curva de los datos de test vemos una gran variación, ya que hay rangos del dominio donde la curva disminuye y algunos donde aumenta. Sin embargo, si vemos la curva completa podríamos decir que a la larga aumenta y luego se plancha. Por lo tanto, consideramos que no deberíamos obtener resultados mucho mejores si seguimos aumentando la cantidad de datos.

Para la curva de los datos de entrenamiento, podemos ver que al principio, cuando tenemos pocos datos de entrenamiento, la curva va aumentando con bastante pendiente pero luego se estanca en 1.

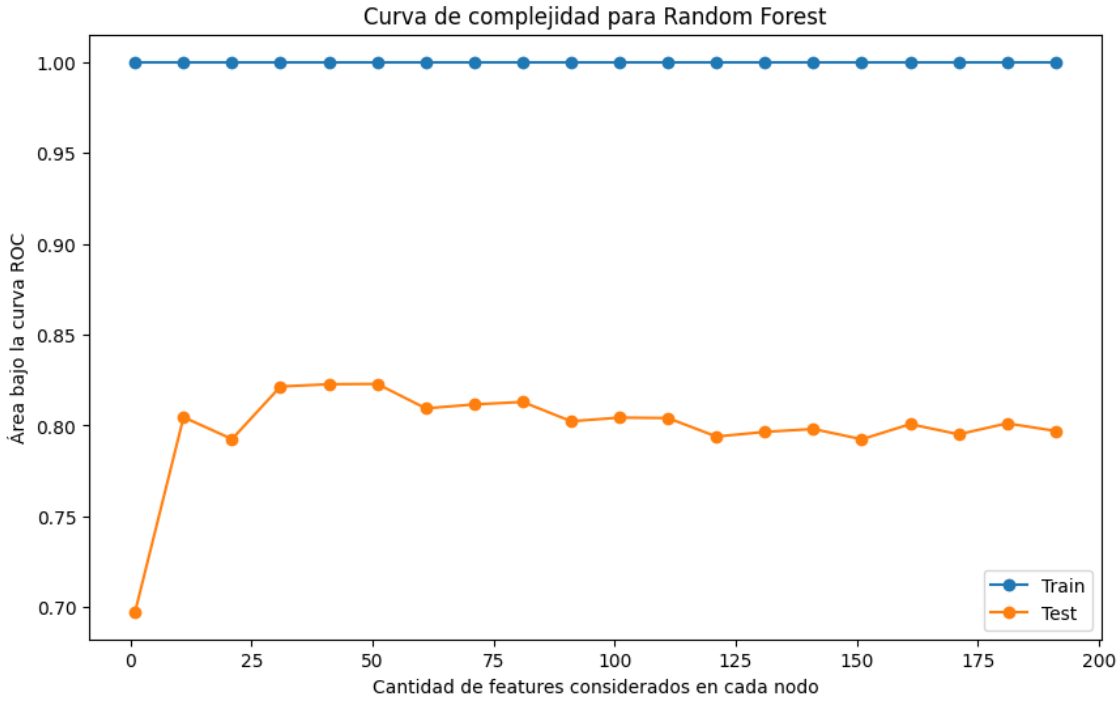
4.2. Inciso 3

En este inciso se nos pide construir un modelo RandomForest con 200 árboles. Además, exploramos para qué sirve el hiperparámetro max_features[4] y cómo afecta a la performance del algoritmo mediante una curva de complejidad. Por último, graficamos una curva de aprendizaje sobre los parámetros elegidos para determinar si sería útil o no conseguir más datos.

El parámetro max_features en Random Forest controla el número de características (features) que se consideran al dividir un nodo durante la construcción de los árboles en el bosque. Puede tomar diferentes valores que afectan la forma en que se construyen los árboles y, por lo tanto, el rendimiento del modelo.

Veamos la curva de complejidad de Random Forest variando la cantidad de features considerados en cada nodo.

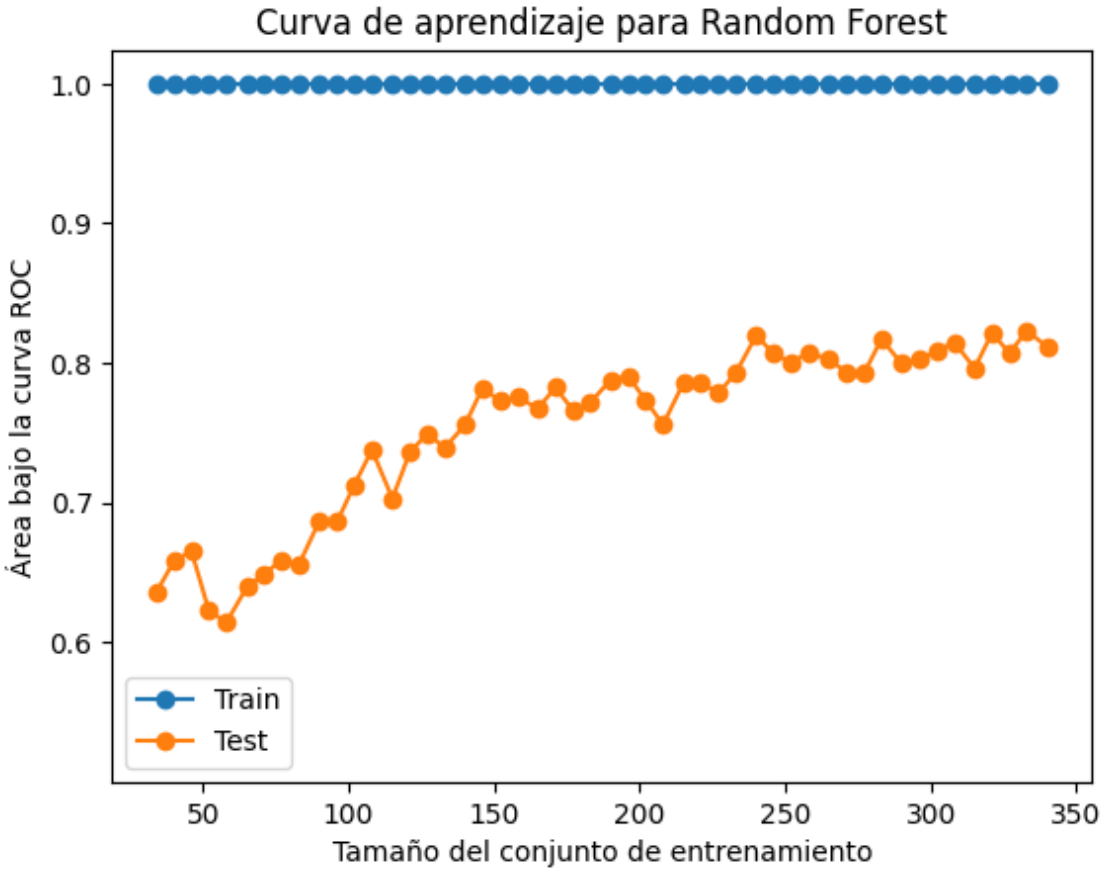
[4]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html



Para la curva de los datos de test, vemos que hay un aumento del área bajo la curva ROC cuando la cantidad de features considerados es pequeña. Sin embargo, a medida que seguimos aumentando esta cantidad, vemos que se plancha la curva y no vemos una mejora. Usualmente, la cantidad de features que se suele utilizar es la raíz de la cantidad total de features que se tienen. En este caso tenemos 200 features, por lo tanto, como $\sqrt{200} = 14.14$ entonces la cantidad de features estándar sería 14. Podemos ver que en el gráfico si tomamos cantidad de features igual a 14 tenemos un área bajo la curva ROC de aproximadamente 0.8, que es cercano al valor del resto de la curva.

Para la curva de los datos de entrenamiento, vemos que el área bajo la curva ROC se mantiene siempre en 1 a medida que aumenta la cantidad de features que tenemos. Esto se debe a que los árboles de decisión se ajustan mucho a los datos pues no le pusimos altura máxima. Por lo tanto, tiene sentido que en ningún caso tenga error.

Veamos ahora la curva de aprendizaje de Random Forest para una cantidad de features considerados en cada nodo igual a 14, por lo explicado anteriormente.



Podemos observar que en la curva de los datos de test, vemos un aumento pero con una pendiente que va disminuyendo a medida que utilizamos más datos para el entrenamiento del modelo. Por lo tanto, pareciera que no va a seguir aumentando mucho el área, por lo que no le vemos mucho sentido en seguir aumentando la cantidad de datos de entrenamiento.

Para esta curva, podemos ver que cuando tenemos pocos datos de entrenamiento tenemos más varianza, pero a medida que aumentamos estos, esta va disminuyendo. Por otro lado, vemos que ambas curvas están bastante separadas, por lo que al igual que en los árboles de decisión, tenemos bastante sesgo.

Si consideramos la curva de los datos de entrenamiento, vemos el mismo comportamiento que en la curva de complejidad.

5. Ejercicio 5: Evaluación de performance

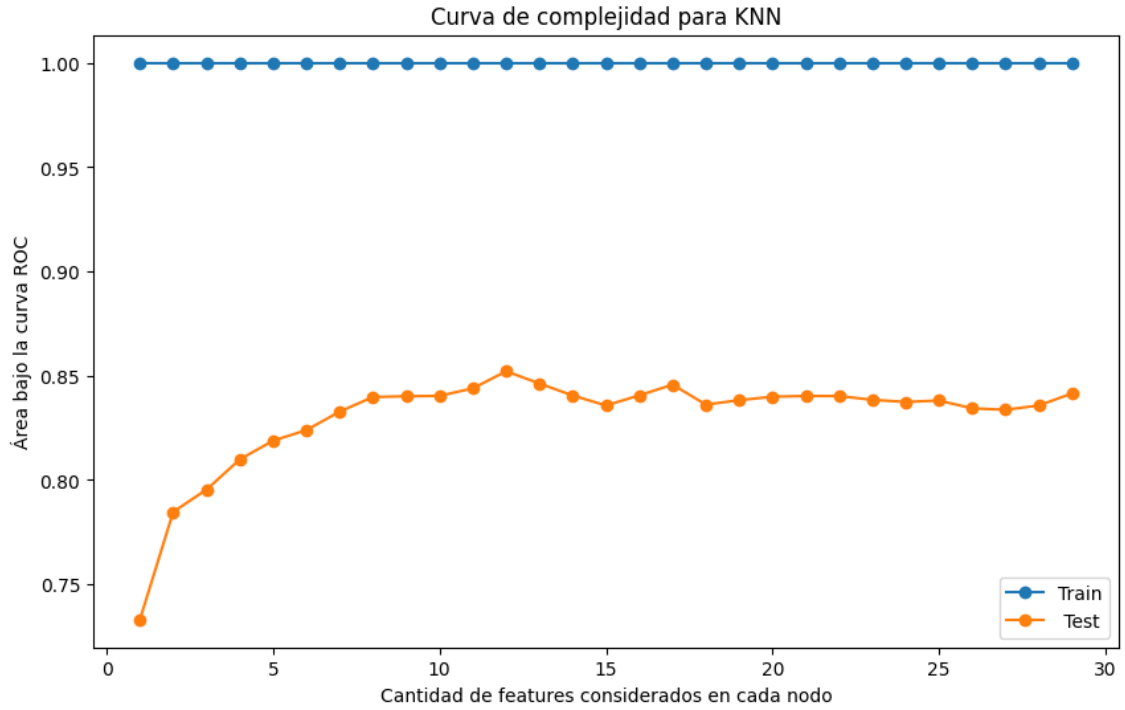
Veamos finalmente cuál es el mejor modelo.

Primero que nada, consideramos los mejores modelos encontrados usando Random Search. Estos fueron los siguientes:

Modelo	Promedio de AUROC	Hiperparámetros
Árbol de decisión	0.646762	{'criterion': éntropy', 'max depth': 3}
SVM	0.891996	{'C': 4.343, 'degree': 4, 'kernel': rbf}
KNN	0.842864	{'*n_neighbors*': 6, 'p': 1, 'weights': '*distance*'}
Naive Bayes	0.822782	{'var_s': 0.000918, 'priors': [0.5, 0.5]}

Cuadro 9: Los hiperparámetros con los valores de *AUROC* para los mejores modelos

Podemos observar que el promedio del área bajo la curva ROC dió en todos los casos más pequeño que para el modelo SVM. Además, en el caso de árboles de decisión, al hacer las curvas de complejidad, vimos que incluso probando con más profundidad máxima del árbol no mejoraba la performance. Al notar esto, nos pareció conveniente hacer la curva de complejidad para KNN variando la cantidad de vecinos. Esto nos dió el siguiente gráfico.



Sin embargo, al ver el gráfico vemos que en ningún caso el área bajo la curva ROC supera el valor alcanzado por SVM.

Por otro lado, como analizamos en la sección anterior, al considerar Random Forest seguimos sin notar mejoras.

Además, en el caso del modelo SVM, si consideramos la varianza y el sesgo, es el que mejor resultados dió e incluso podría llegar a dar mejor si agregamos más datos al conjunto de entrenamiento. Por lo que consideramos que es el modelo que mejor generaliza.

Finalmente, llegamos a la conclusión de que el mejor modelo es el SVM con kernel radial y C = 4.434.

Una vez que ya teníamos elegido el mejor modelo, lo entrenamos con todos los datos de desarrollo y una vez entrenado, lo usamos para predecir los datos de evaluación. Esto nos dió un área bajo la curva ROC de 0.8137. Podemos decir que esto tiene sentido, ya que el valor que obtuvimos para el area bajo la curva ROC durante la seleccion del modelo fue un parametro que estuvimos tuneando, es decir, que fuimos probando distintos modelos con distintos parametros para quedarnos finalmente con el que nos otorgo el valor mas alto, esto generalmente no representa la realidad y es por esta misma que empezamos separando un porcion de los datos, para despues de haber elegido nuestro modelo poder realizar estimaciones reales"de la performance del mismo.

Por último, para cada dato del dataset X_held_out calculamos la probabilidad de pertenecer a cada clase utilizando el modelo seleccionado.

6. Conclusiones

Este fue un trabajo enfocado en la detección temprana y el pronóstico preciso de enfermedades como el cáncer a través de modelos de machine learning. Nuestro estudio se enfocó en encontrar el mejor modelo para predecir si una persona tiene un buen o mal pronóstico para este tipo de enfermedades. A lo largo del trabajo, usamos varios modelos, tales como árboles de decisión, KNN, Support Vector Machines, Linear Discriminant Analysis, Naive Bayes y Random Forest. Finalmente, encontramos que el modelo SVM con kernel radial y un valor de C de aproximadamente 4.343 es el que mejor generaliza y proporciona resultados más prometedores en términos de área bajo la curva ROC.

Para los distintos modelos, analizamos su comportamiento mediante curvas de aprendizaje y de complejidad. Esta exploración nos permitió comprender mejor cómo cada modelo se desempeña con distintos valores de hiperparámetros y tamaños del conjunto de datos de entrenamiento, como también cómo cada modelo maneja el sesgo y la varianza.

Al hacer esto, nos encontramos con una serie de problemas que tuvimos que afrontar.

Por un lado, al momento de graficar las curvas de aprendizaje, descubrimos que sklearn no aclaraba si al agregar los datos al entrenamiento lo hacia de forma acumulativa. Por lo tanto, haciendo un análisis exhaustivo de la documentación y el código de la función learning_curve pudimos encontrar una función llamada learning_curve_Display.from_estimator()[5]que tenía un parámetro que indicaba si lo hacía de forma acumulativa, y por default estaba en True. Por ende, pudimos utilizar esta función para graficar las curvas de aprendizaje.

Cuando examinamos las curvas de complejidad del árbol de decisión, notamos que la curva correspondiente a las predicciones en los datos de test no mostraba una tendencia a estabilizarse, a pesar de haber logrado clasificar correctamente todas las instancias en nuestros datos de entrenamiento. La única variación que estábamos introduciendo era aumentar la profundidad máxima del árbol. Al revisar con mayor detenimiento la documentación de Decision Tree Classifier[6], descubrimos que el algoritmo utilizado para entrenar nuestro árbol no era determinista. Esto explicaba por qué nuestra curva de test no era estable. Para abordar este problema, decidimos fijar una semilla para eliminar este componente de aleatoriedad del algoritmo. Como resultado, logramos obtener una curva de complejidad que se estabilizó satisfactoriamente.

Finalmente, al aplicar el modelo SVM seleccionado en los datos de evaluación, obtuvimos una puntuación de área bajo la curva ROC de 0.8137, lo que confirma la eficacia y la generalización del modelo seleccionado.

[5]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LearningCurveDisplay.html
[6]https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html