Trabajo Práctico 1 Aprendizaje Automatico

Aprendizaje Automático

Mayo 2025 - 1C

Autor	Correo electrónico
Falczuk, Noelia	noefalczuk@gmail.com
Fernández Zaragoza, Mariano	marianofzaragoza@gmail.com
Fiore, Juan Ignacio	juanifiore291@gmail.com
Larrea, Juan Iñaki	juaninaki@gmail.com
Ombrosi, Ulises	uombrosi@gmail.com

Índice

1. Separación de Datos

1.1. Pregunta 1

La consciente separación de datos es crucial para poder realziar análisis de los mismos y comprender los resultados. Por esto, comenzamos por estudiar las proporciones de la columna 'target'. Encontramos que de las 500 instancias, el 29.8 % está clasificado como buen pronóstico (1) y el restante como mal pronóstico (0).

Además, como buscamos estimar de forma precisa el desempeño del modelo, decidimos considerar la división en desarrollo y evaluación en la cual la proporción de etiquetas (0 y 1) que queda en cada set de validación respete a la proporción total. La motivación es asumir que la distribución de etiquetas recibida representa la distribución real en el mundo real. Por ende, queremos respetar esa proporción para no sub-estimar ni sobre-estimar el modelo. Esto esta aún más sustentado cuando consideramos que el problema que se analiza es un problema biológico. Parece correcto suponer que la distribución de las muestras va a respetar la distribución real de los datos.

Finalmente, decidimos tomar 15 % de los datos en el conjunto de evaluación, decisión tomada luego de analizar cómo quedaría el tamaño de los folds para k=5 al hacer cross validation con los datos luego de tomar el 20, 15 o 10 % delos datos para testear. Cabe mencionar que, en en este trabajo, priorizamos una buena estimación de evaluación a un mayor aprendizaje, pero también tomando un alto porcentaje que permita realizar predicciones coherentes y fundamentadas. Por eso, el porcentaje elegido fue 15.

2. Construcción de Modelos

2.1. Primer Arbol de decisión

Entrenamos un árbol de decisión con altura máxima 3 y el resto de los hiperparámetros en default.

2.2. K-Fold Cross Validation

Estimamos la performance del modelo utilizando K-fold cross validation con K=5, con las métricas Accuracy, Area Under the Precision-Recall Curve (AUPRC), y Area Under the Receiver Operating Characteristic Curve (AUCROC). Además, cuando corresponde calculamos el score global. Exhibimos los resultados en una tabla. Redondeamos para mejorar la legibilidad.

Cuadro 1: Resultados de evaluación del modelo

Fold	**Accuracy train**	**Accuracy val**	**AUPRC train**	**AUPRC val**	**AUCROC train**	**AUCROC val**
0	0.841	0.738	0.567	0.160	0.838	0.579
1	0.821	0.702	0.504	0.307	0.807	0.579
2	0.844	0.679	0.509	0.279	0.841	0.645
3	0.835	0.714	0.591	0.361	0.804	0.667
4	0.821	0.568	0.485	0.230	0.802	0.471
Promedio	0.833	0.680	0.531	0.267	0.818	0.595
Global	NaN	0.679	NaN	0.261	NaN	0.598

2.3. Comparación Hiperparámetros

Luego, exploramos las siguientes combinaciones de parámetros para árboles de decisión (siguiendo con k-fold con k=5) utilizando ParameterGrid de scikit learn.

Cuadro 2: Comparación de Accuracy con diferentes alturas máximas y criterios de corte

	Altura máxima	Criterio de corte	Accuracy (training)	Accuracy (validación)
0	3	gini	0.833	0.678
1	5	gini	0.923	0.648
2	None	gini	1.000	0.631
3	3	entropy	0.797	0.675
4	5	entropy	0.896	0.632
5	None	entropy	1.000	0.651

2.4. Conclusiones

Respecto de la primera tabla, las medidas sobre el conjunto de train muestran valores mayores que en los conjuntos de test como es de esperar. También vemos que para los diferentes scores (excepto el accuracy) calcular el promedio no coincide con el global. En general, los valores son tan bajos que nos hace pensar que el modelo no es bueno para el problema de aprendizaje planteado.

La diferencia entre el score global y el promedio se debe a cómo se calcula el score durante un proceso de validación. El score promedio se obtiene al calcular la métrica de evaluación de manera independiente en cada fold de la validación cruzada dejado para la validación. Luego se promedian estos resultados. Por el otro lado, el score global se calcula guardando todas las predicciones realizadas sobre los conjuntos de validación de cada fold. Luego, se calcula la métrica sobre las predicciones guardadas.

Sobre la segunda tabla se destaca que tanto para los criterios de Gini como el de Entropía, aumentar la altura máxima del árbol mejora el accuracy en train pero la empeora en test. Es decir, incluso para estas alturas bajas parece haber un sobre-ajuste. Esto refuerza que el modelo no es el adecuado para el problema.

Que ambos criterios hayan alcanzado un accuracy de 1 en el conjunto de train al no restringir la altura nos indica que no existen instancias contradictorias (que tengan los mismos valores en los atributos y una etiqueta diferente).

Comparando los hiperparámetros acorde al accuracy de validación, vemos que gini es un mejor hiperparámetro que entropy, aunque no es una diferencia significativa. Para las alturas que probamos, la mejor opción es entrenar un árbol de altura máxima 3 con gini como criterio de corte.

Cabe mencionar que, a pesar de que la altura máxima del árbol fue dejada irrestricta en algunos casos, tomó los valores [12, 13, 15, 13, 17] usando Gini y [8, 11, 11, 9, 11] usando entropy como criterios de corte. Esto podría deberse a que la cantidad de datos no permite dividir más veces en conjuntos que sean conjuntos que mejoren la ganancia de información o disminución de entropía. Parecería, preliminarmente, junto a los valores de accuracy para validación, que gini permitiría más sobre ajuste cuando libero la altura máxima.

3. Comparación de Modelos

Analizamos distintas combinaciones de hiper parámetros para cada algoritmo. En cada uno de ellos corrimos de forma Random Search para 50 iteraciones. Seleccionamos la mejor combinación de hiper parámetros evaluandolos mediante validación cruzada con la métrica ROC AUC. Explicamos brevemente cada hiper parámetro y hacemos algún comentario sobre los espacios de búsqueda y resultados obtenidos. La dificultad del Random Search es que aunque devuelve una combinación de hiperparametros óptima entre las probadas, no permite identificar cuáles de los parámetros son importantes y cuáles no. Para eso, incluimos gráficos que distinguen todas las iteraciones y corridas según sus hiperparametros. Nos permite rápidamente destacar los hiper parámetros que están muy correlacionados con la métrica que queremos optimizar. Lo desarrollamos para cada algoritmo pero aquí mostramos solamente el gráfico para SVM.

3.1. Hiperparamteros

Hiperparámetros en Árboles de Decisión:

'max_depth': restringe la profundidad máxima que alcanzará el árbol entrenado.

'criterion': define el criterio de corte para la reducción de la impureza (Gini o Entropy).

'splitter': define si en cada nodo se elige el mejor corte posible o si primero se elige un atributo al azar y luego se elige el mejor corte sobre ese atributo.

'min_samples_split': Establece la cantidad mínima de instancias para la cual evalúa si hacer un corte en un nodo del árbol.

'min_samples_leaf': Establece la cantidad mínima de instancias que pueden quedar en una hoja. Si un corte no cumple ese mínimo, no se hace.

Mejor combinación:

'splitter': best, 'min_samples_split': 16, 'min_samples_leaf': 10, 'max_depth': 10, 'criterion': 'gini'. Auc Roc: 0.689.

Para elegir los hiper-parametros sobre los que ibamos a hacer un random search y para fijar los rangos de busqueda, nos basamos en el ejercicio dos donde nos daba una idea de las mejores alturas, y buscamos impedir el sobre y sub-ajuste involucrando splitter, min samples split y min samples leaf.

Los hiper-parámetros que dieron los mejores resultados no contradicen los esperados luego de hacer el ejercicio 2. Sin embargo, al hacer random search, corremos el riesgo de no testear atributos importantes o no visualizar la importancia de algunos atributos. Pues, en el espacio de búsqueda, estos se eligen al azar. Para hacerle frente a este problema, decidimos hacer gráficos que mostraran para cada selección probada, qué tan bien performa el modelo. De esta manera, podemos disminuir las consecuencias de no elegir bien el espacio de búsqueda y agrandarlo o transladarlo a donde observamos que performa mejor.

Por otro lado, muchos de los hiper-parámetros elegidos están relacionados a la hora de construir el árbol. Por ejemplo, si min_samples_split=a y min_samples_leaf=a, ambas cotas no son independientes, ya que la segunda implica la primera: si no se permite que una hoja tenga menos de a instancias, entonces nunca se hará un corte en un nodo con menos de a instancias. Tratamos de tener en cuenta estas dependencias para definir un espacio de búsqueda inteligente.

Hiperparametros en KNN:

'n_neighbors': Número de vecinos que se usa para predecir la próxima entrada.

'weights': Define cómo se pesa la información de los vecinos para predecir (uniformemente o acorde a la distancia de la instancia a predecir).

'algorithm': Define el algoritmo que se usa para computar las distancias a vecinos.

Mejor combinación: 'weights': distance, 'n_neighbors': 6, 'algorithm': ball_tree'. Auc Roc: 0.840

Comentario: Revisando nos dimos cuenta que el hiper parámetro algorithm no modifica los valores de las distancias, solamente modifica la eficiencia. Por lo que para las métricas que estamos considerando, no tenía sentido variar este parámetro.

Hiperparametros en SVM:

'C': Define cuánto se permiten o penalizan los errores de clasificación, es necesario para los conjuntos que no pueden ser perfectamente separables. Cuánto más bajo el c, se permiten más instancias mal separadas.

'kernel': Define el Kernel que se usará. Esto establece en qué tipo de estructuras (o con que topologías) puede separar los datos el algoritmo.

Mejor combinación: mejor SVM: C : 3.021, 'kernel': 'rbf'. Auc Roc: 0.915

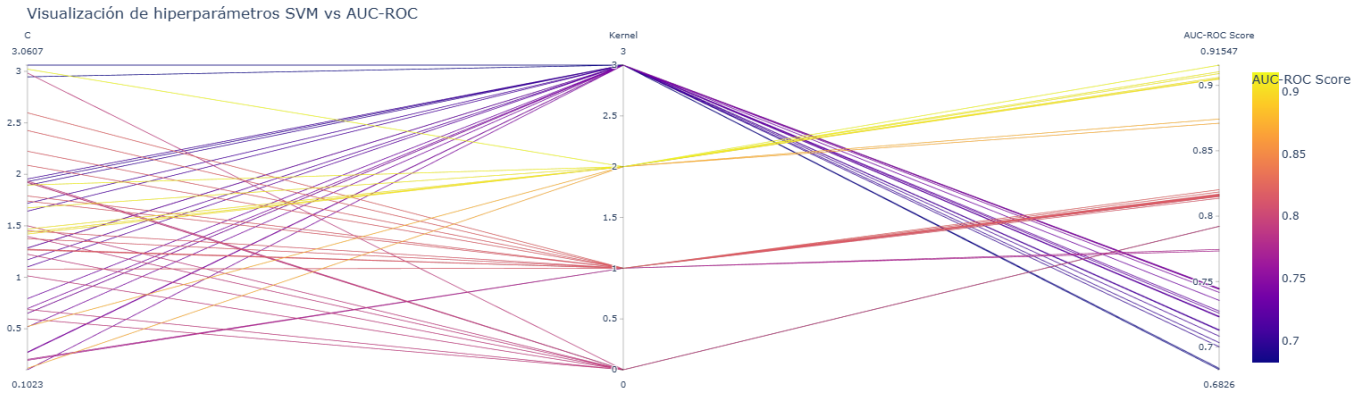


Figura 1: Kernels: 'linear': 0, 'poly': 1, 'rbf': 2, 'sigmoid': 3

Comentarios: Vemos que el kernel RBF es superior a los demás y parece más determinante la elección del Kernel a la del c. Sobre todo, comparándolo con la performance fijando el Kernel Lineal (Aux Roc = 0.792) y realizando nuevamente el randm search, podemos asumir que los datos están lejos de ser linealmente separables, tienen otra estructura que RBF si está captando.

3.2. Comparación con otros modelos

En esta sección evaluamos la performance de los modelos LDA y Naive Bayes. Ambos modelos tienen un AUC cercano, pero inferior al obtenido usando SVM. LDA logra precisión de 0.728 y Naive Bayes, 0.730. Parecería que logran captar información pero no suficiente. Observar que su precisión es casi la misma que para SVM con kernel linear. Esto es especialmente importante para LDA pues este modelo es bueno cuando las clases son linealmente separables. Respecto a Naive Bayes, este modelo asume que las características son independientes pero considerando la poca cantidad de datos y la gran cantidad de atributos, es difícil esperar esto.

En coclusión, se deberían probar parámetros que enfrenten estos problemas. Por ejemplo, para LDA es posible usar 'solver' y shrinkage que permitiría disminuir la correlación o tenerla en cuenta. priors también sería útil pues, como dijimos anteriormente, nuestros datos están desbalanceados; usar este parámetro permitiría darle mayor peso a esa clase minoritaria. Para Bayes, podríamos usar también el parámetro 'priors' o 'fitprior' por la misma razón. Además, podríamos usar el parametro 'alpha' ya que al tener tantas features y pocos datos es probable que la frecuencia de aparición de algunos valores sea muy cercana a cero y esto este afectando el modelo.

3.3. Mejor Modelo

El mejor modelo fue el SVM con los hiper-parámetros 'C' = 3.0212665565243775 y 'kernel': 'rbf'. No solo fue el mejor en terminos de score, sino también en terminos de varianza entre folds y diferencia entre score en entrenamiento y test durante la validación cruzada (cuadro 3).

Si bien lo comentamos antes, en base al mejor modelo obtenido creemos que en los datos pueden haber relaciones no lineales que no son bien captadas por otros kernels, e incluso otros algoritmos, pero el kernel radial (RBF) sí capta estas estructuras. Además, el mejor valor del hiper parámetro C fue el mayor entre los que probamos. Como un C alto penaliza las instancias mal clasificadas, que haya tenido una buena performance indica que el algoritmo está pudiendo separar los datos con pocos errores.

Modelo	Varianza entre folds	Train Score	Test Score (CV)	Diferencia
Árbol	0.0071	0.9497	0.6591	0.2907
KNN	0.0050	1.0000	0.8403	0.1597
SVM	0.0013	1.0000	0.9155	0.0845

Cuadro 3: Comparación de modelos: varianza entre folds y scores en validación cruzada.

4. Sesgo y Varianza

4.1. Curvas de complejidad

Durante el desarrollo de este ejercicio decidimos realizar dos implementaciones: una implementación usando la librería y otra, realizandola a mano de forma de no tener que promediar los valores de AUC-ROC de cada fold en el cross validation.

En primer lugar, es necesario notar que para el caso del SVM las curvas dan muy similares pero para el caso de árboles no. Por lo tanto, se hará un análisis general de cómo afectan los parametros y luego se realizará en la sección 6 un análisis de posibles complicaciones en el uso de sklearn y sus librerías, sin conocer cómo calculan los scores.

La curva de complejidad para el hiperparámetro C del modelo SVM muestra una cierta estabilidad del score sobre test para valores de C menores a 0.5, manteniéndose por debajo de 0.86. A partir de C = 0.5 hasta C = 2, el score mejora notablemente, alcanzando un valor cercano a 0.89, y luego permanece estable hasta C = 100. Sobre el set de entrenamiento presenta un comportamiento similar, pero con scores consistentemente mayores (superiores a 0.98), convergiendo a 1.0 para valores más altos de C. Esto indica un posible sobreajuste progresivo, ya que al aumentar C, el modelo tiende a penalizar menos los errores en el entrenamiento, reduciendo el sesgo a costa de un aumento en la varianza.

Respecto al trade-off sesgo-varianza, a valores pequeños de C existe una regularización más fuerte. El modelo es más simple, con menor varianza pero mayor sesgo. Por esto, el desempeño sobre test es menor. A medida que C crece se reduce el sesgo. Sin embargo, parecería que sigue generalizando bien sobre el set de testeo.

Cabe destacar que C = 3, valor elegido para el mejor modelo de SVM, se encuentra dentro del rango estable identificado como reducción de sesgo y varianza.

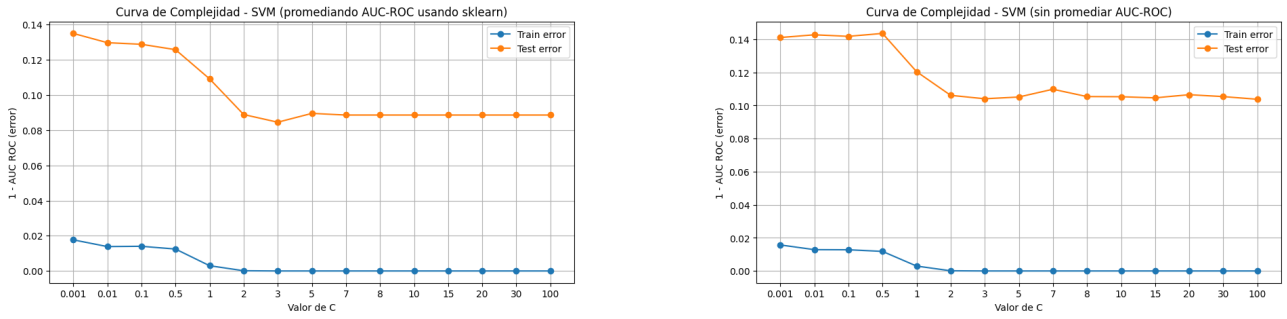


Figura 2: Comparación de curvas de complejidad para SVM.

En cuanto a árboles, es distinto lo que ocurre en el gráfico realizado a partir de la librería y el hecho 'a mano'. Realizaremos un pequeño análisis de ambos.

En el primer caso, se observa que a profundidades bajas existe alto sesgo y que varianza debido a que el modelo es más simple (max depth menor) pero no logra capturar bien la estructura de los datos fallando más en la detección. El error el train y test es mayor. A medida que se aumenta la complejidad, disminuye el error, disminuye el sesgo, pero aumenta la complejidad.

Se observa, en el segundo caso, como a mayor complejidad el modelo tiene menor bias pero mayor varianza, pues aumenta el error en testeo mientras baja el de entrenamiento. Sucede que para valores bajos (max depth = 2 0 3) se consigue mejor performance en el set de testeo a costa de menor performance en el de entrenamiento.

Aquí parecería lograrse un mejor equilibrio entre sesgo y varianza. Finalmente, es este segundo gráfico que logra mostrar como aumentar la complejidad del modelo hacer que se llegue a un over-fitting y disminuirla hace que haya under-fitting.

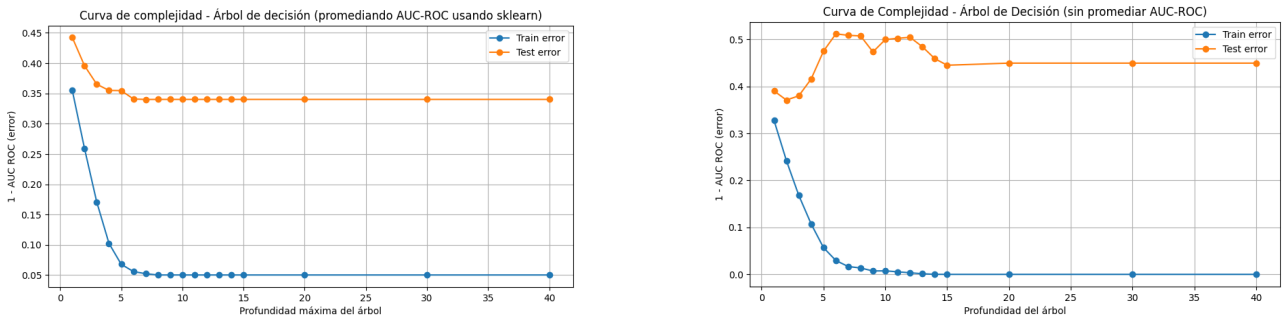
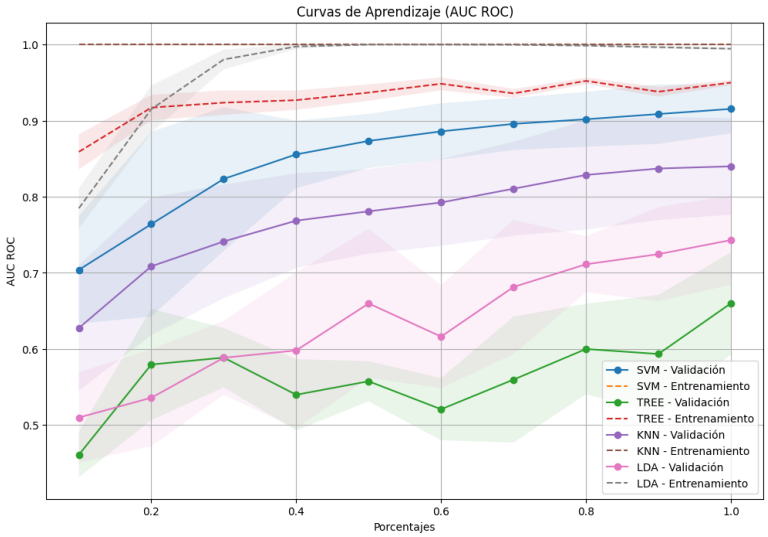


Figura 3: Comparación de curvas de complejidad para el Árbol de Decisión.

4.2. Curvas de aprendizaje

Observamos que los conjuntos de entrenamiento de los modelos SVM, KNN y LDA presentan valores de AUC ROC cercanos a 1, lo cual puede indicar que para determinados tamaños de muestra estos modelos tienden a sobreajustar los datos de entrenamiento. Viendo a los conjuntos de validación de dichos modelos, vemos como tienden a aumentar su valor en el AUC ROC a medida que tomamos más datos. Esto nos puede sugerir que ayuda aumentar la cantidad de datos.

En el caso del modelo TREE, se observa que el AUC ROC en el conjunto de entrenamiento se estabiliza en un valor inferior a 0,9, mientras que en el conjunto de validación nunca supera el umbral de 0,7. Pensamos que esta brecha entre ambos conjuntos sugiere una capacidad limitada del modelo para generalizar a nuevos datos. Además, el hecho de que el AUC ROC obtenido al entrenar con el 90 % de los datos sea superior al obtenido utilizando la totalidad del conjunto sugiere que un aumento en la cantidad de datos no necesariamente contribuye a una mejora en el rendimiento del modelo y creemos que en este caso alcanzó su límite.

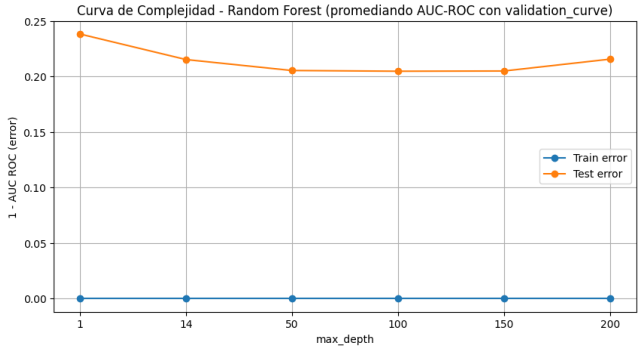


4.3. Random Forest

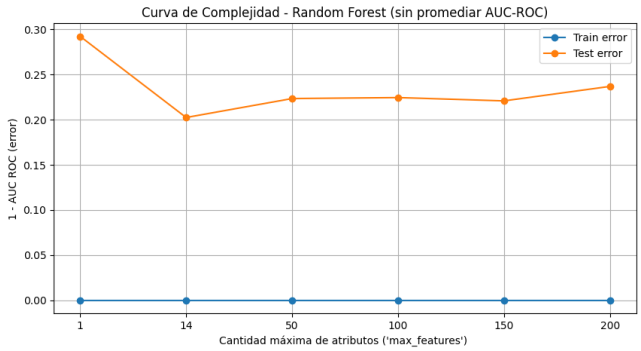
Una vez entrenado el Random Forest con 200 árboles, graficamos una curva de complejidad variando el hiperparámetro “max features”, el cual refiere a la “m” vista en clase y en la bibliografía, especificando la cantidad de atributos que se eligen de manera aleatoria en cada nodo al realizar los cortes en las ramificaciones de los árboles entrenados.

Para Ms muy pequeños, la performance no es tan buena dado que son pocos los atributos con los que puede tomar una decisión de corte en los nodos, por eso vemos que a mayor valor el error decrece, pero a partir de cierto puntos (aproximadamente 100) el error vuelve a subir, dado que con tanta cantidad de atributos es posible que el modelo empiece a sobreajustar, debido a la correlación que puede existir entre los atributos (en m = 200 es lo mismo que hacer bagging).

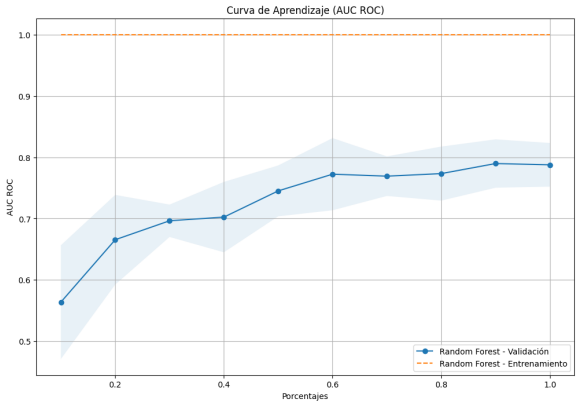
Una vez realizada la curva de aprendizaje (figura 4.c) para este algoritmo, podemos apreciar una tendencia creciente (aunque no tan pronunciada al final) que nos hace creer que quizás más datos podrían favorecer a un mejor aprendizaje, por lo que resultaría útil su obtención.



(a) Curva de Complejidad



(b) Curva de Complejidad (sin promediar)



(c) Curva de Aprendizaje

Figura 4: Comparación de curvas: dos de complejidad a la izquierda, la de aprendizaje a la derecha.

5. Evaluación de performance

Realizamos la evaluación utilizando nuestro mejor modelo, un SVM donde los mejores parametros obtenidos son: C = 3.0212665565243775 y kernel = 'rbf'. Este modelo fue entrenado con los datos de entrenamiento (x train, y train) y evaluado sobre el conjunto de validación (x validacion, y validacion), reservados previamente con este fin. A partir de las probabilidades predichas por el modelo, se calculó el rendimiento mediante la métrica AUC-ROC. Estimamos un valor de performance de 0.835.

Finalmente, se utilizaron los datos provistos por la cátedra, contenidos en la variable HELD OUT, los cuales no poseen etiquetas. Con el objetivo de generar predicciones sobre este conjunto, se reentrenó el modelo SVM previamente seleccionado utilizando todos los datos disponibles (data). Una vez finalizado este entrenamiento, se generaron las probabilidades predichas sobre HELD OUT, guardadas en Y HELD OUT.

6. Conclusiones

Este trabajo práctico nos fue muy útil para realizar por completo las distintas etapas en un trabajo de la literatura de aprendizaje automático. Comenzamos discutiendo, a partir de los datos que teníamos, cual era la mejor decisión para la separación de los datos en conjuntos de entrenamiento y validación. Esto es crucial para asegurar un aprendizaje realista y acertado sobre los datos con los que trabajamos, y que luego generalice bien para distintos datos nuevos. En la segunda etapa, construimos y entrenamos distintos modelos con el set de entrenamiento, sin utilizar los datos de evaluación para luego no sobreestimar la performance.

Después de realizar los modelos, graficamos curvas de aprendizaje y de complejidad para entender mejor a cada uno de ellos, y así ver cómo los distintos hiper parámetros pueden afectar en el desempeño de sus estimaciones, y a su vez ver cómo performan en función del porcentaje de datos con los que son entrenados.

Una vez seleccionado nuestro mejor modelo, evaluamos sobre los datos de validación que separamos en un principio, para así obtener un score general sobre la performance de nuestro modelo. Luego, surgieron argumentos a favor y en contra de utilizar todos los datos de entrenamiento para reentrenar este mejor modelo y, con él, realizar las predicciones sobre el conjunto de datos de la competencia brindados por la cátedra. Observando las curvas de aprendizaje y sus pendientes positivas, decidimos que a mayor cantidad de datos mejor poder de generalización tiene el modelo. Así, entrenar con todos los datos sería lo adecuado.

Finalmente, para entender los resultados, se hizo una proyección PCA de los datos held out y los usados durante el entrenamiento (entrenamiento y validación) y un análisis de gaussianas y likelihood donde obtuvimos los siguientes valores: LL(HELD OUT|Val): 245851109 / LL(HELD OUT|Train): 15662113 / LL(Val|Train): 6013809. A pesar de que las escalas no los hacen comparables, podemos ver qué tan parecidos son los datos de Validación con los Held Out y los datos de train con los Held out. Además, realizar un PCA (para visualizar el gráfico interactivo dirigirse al collab) también permite observar que los datos de entrenamiento y los HELD OUT serían de distribuciones similares, lo que se condice con el marco contextual del problema.

Como se nombró en la Sección 4.1, calcular el AUC-ROC promediando los valores obtenidos en cada fold durante el k-fold cross-validation no necesariamente equivale al score global; el promedio por folds tiende a sobreestimar el rendimiento del modelo. Esta sobreestimación puede deberse a la alta varianza generada por tener pocos datos en los conjuntos de test de cada fold, haciendo que algunos folds tengan scores 'falsamente' altos. Por esta razon, siempre se optó por graficar ambas curvas (complejidad global y promedio) y realizar una comparación. Este debe ser considerado al comparar modelos. Por ejemplo, al usar RandomizedSearchCV de scikit-learn, el best estimator es aquel que maximiza la métrica promedio durante la validación cruzada. Sin embargo, esto no significa que ese modelo sea el mejor. Podría ser que para ciertos folds sea el que mejor da pero no en general.

Un ejemplo de esto es lo que sucede con árboles. El modelo con max depth=10 fue seleccionado como el mejor según el promedio de roc auc, pero al calcular el score global con este modelo sobre un conjunto independiente, el resultado fue 0.5, equivalente a un clasificador aleatorio.