Aprendizaje Automático I

Trabajo Práctico 1 - Aprendizaje supervisado

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Clasificación de expresiones genómicas

Integrante	LU	Correo electrónico
Alamo, Malena	1620/21	malusalamo@gmail.com
Fontana Walser, Florencia	1530/21	florfontana02@gmail.com
Klimkowski, Victoria	1390/21	02vicky02@gmail.com
Gandolfo, Lorenzo	169/21	lolegandolfo@gmail.com
Corzini, Frabrizio	32/22	fcorzini@gmail.com

Ejercicio 1 - Separación de datos

Dado que se dispone de una cantidad limitada de muestras (500 instancias), fue fundamental definir una estrategia de partición que permitiera desarrollar y evaluar modelos sin introducir sesgos ni pérdida de información relevante.

Se comenzó analizando la distribución de la variable objetivo `target`, y se observó un desbalance entre las clases de buen pronóstico (1) y mal pronóstico (0). Ante este desbalance, se decidió realizar una división estratificada, de modo que las proporciones de cada clase se mantuvieran tanto en el conjunto de entrenamiento como en el de testeo.

Para implementar la partición sin utilizar la función `train_test_split` de `sklearn`, se separaron las muestras de cada clase, se mezclaron y luego se extrajo un 90 % de cada clase para conformar el conjunto de entrenamiento, reservando el 10 % restante para el conjunto de testeo. Finalmente, se combinaron las particiones de ambas clases y se mezclaron nuevamente para evitar cualquier ordenamiento implícito.

La proporción final de clases en cada conjunto fue del 70 % para el target 0 y 30 % para el 1.

Esta división de los datos constituye la base sobre la cual se entrenarán y evaluarán los modelos a lo largo del trabajo.

Ejercicio 2 - Construcción de modelos

2.1

Definimos nuestro árbol de decisión con los parámetros indicados en la consigna.

2.2

Para resolver el ejercicio implementamos una función que toma como parámetros `X` (datos), `y` (columna target), `model` (modelo a entrenar) y `k` (cantidad de folds para realizar la validación cruzada). La misma realiza una validación cruzada estratificada sobre los datos y calcula las métricas indicadas en la consigna: Accuracy, Area Under the Precision-Recall Curve (AUPRC) y Area Under the ROC Curve (AUCROC).

Calculamos los scores para cada fold de validación, sus promedios, y también el **score global**, es decir, el valor de las métricas aplicadas a todas las instancias de validación combinadas (como se vio en clase).

Resultados

Permutación	Accuracy (training)	Accuracy (validación)	AUPRC (training)	AUPRC (validación)	AUC ROC (training)	AUC ROC (validación)
1	0.805	0.722	0.740	0.406	0.805	0.645
2	0.777	0.622	0.754	0.362	0.838	0.631
3	0.811	0.711	0.704	0.439	0.821	0.643
4	0.808	0.589	0.670	0.346	0.836	0.524
5	0.836	0.708	0.709	0.414	0.790	0.652
Promedios	0.807	0.670	0.715	0.393	0.818	0.619
Global	NO	0.670	NO	0.385	NO	0.633

Cuadro 1: Resultados del árbol de decisión con `max_depth=3` evaluado mediante validación cruzada estratificada (K=5).

Los resultados muestran cierta variabilidad entre folds, especialmente en las métricas de validación, lo cual es esperable dada la limitada cantidad de datos. La diferencia entre las métricas promedio y global es relativamente baja, lo cual indica que el modelo tiene una performance razonablemente estable entre permutaciones.

2.3

A continuación se presentan los resultados obtenidos luego de realizar la búsqueda con `ParameterGrid` de `scikit-learn`:

Altura máxima	Criterio de corte	Accuracy (training)	Accuracy (validación)
3	Gini	0.807	0.670
5	Gini	0.929	0.670
Inf	Gini	1.000	0.673
3	Entropía	0.797	0.690
5	Entropía	0.910	0.635
Inf	Entropía	1.000	0.653

Cuadro 2: Promedios de accuracy para distintas combinaciones de hiperparámetros en árboles de decisión.

2.4

A partir de estas tablas se observa que al aumentar la altura máxima del árbol, la `accuracy` sobre los datos de entrenamiento mejora, llegando incluso al valor perfecto de 1 cuando no se impone límite de profundidad. Sin embargo, esta mejora no se traduce en una mejor performance sobre los datos de validación, donde las diferencias son mínimas o incluso decrecen levemente. Esto podría significar que los modelos más complejos están sobreajustando, capturando detalles específicos del conjunto de entrenamiento que no generalizan bien.

En cuanto al criterio de corte (`gini` vs. `entropy`), no se observaron diferencias significativas en la `accuracy`, por lo que ambos resultan comparables para este problema.

Se puede observar la necesidad de controlar la complejidad del modelo, especialmente en datasets con poca cantidad de datos.

Ejercicio 3 - Comparación de algoritmos

Para encontrar la mejor configuración de cada uno de los modelos pedidos se utilizó `RandomizedSearchCV` con 1000 iteraciones y validación cruzada de 5 folds, utilizando el AUC ROC como métrica de evaluación para cada uno.

3.1.1 - Árboles de decisión

A continuación se describen los hiperparámetros seleccionados y la justificación de la elección para cada uno. También. para la elección de los mismos, sumamos recomendaciones brindadas por ChatGPT:

- `max_depth`: controla la profundidad máxima del árbol. Se exploraron valores desde 1 hasta 50 en pasos de 1, más la opción `None` (sin restricción), para cubrir desde modelos muy simples hasta modelos complejos con riesgo de sobreajuste.
- `criterion`: define la función de impureza utilizada para medir la calidad de los splits. Se incluyeron `gini`, `entropy` y `log_loss`, para evaluar diferentes enfoques teóricos de separación.
- `splitter`: determina la estrategia de división en cada nodo. Se compararon `best` (divide con el mejor split posible) y `random` (elige aleatoriamente entre los mejores), lo cual afecta la aleatoriedad del modelo y su variabilidad.
- `class_weight`: permite balancear el peso de cada clase. Se probaron `None` y `balanced`, siendo relevante debido al desbalance en la variable objetivo.
- `min_samples_split`: número mínimo de muestras requeridas para dividir un nodo. Se exploraron valores entre 2 y 100 en pasos de 5, permitiendo controlar la granularidad de las divisiones.
- `min_samples_leaf`: número mínimo de muestras en una hoja. Se probaron valores entre 1 y 20 en pasos de 5 para evitar hojas con pocas muestras que tienden al sobreajuste.
- `max_features`: cantidad máxima de features a considerar al dividir un nodo. Se incluyeron opciones clásicas como `None`, `sqrt`, `log2`, y fracciones como 0.2 y 0.5, lo que afecta directamente la variabilidad y generalización del modelo.
- `max_leaf_nodes`: límite superior de nodos hoja. Se probaron valores entre 2 y 100 en pasos de 5, más la opción `None`, para controlar la complejidad del árbol.
- `min_weight_fraction_leaf`: fracción mínima del total de muestras requerida en una hoja. Se probaron los valores 0.0, 0.01 y 0.05 como mecanismos adicionales de regularización.

```
Mejores parámetros:
{'splitter': 'random', 'min_weight_fraction_leaf': 0.05, 'min_samples_split': 52,
  'min_samples_leaf': 11, 'max_leaf_nodes': 62, 'max_features': None,
  'max_depth': 34, 'criterion': 'entropy', 'class_weight': 'balanced'}

Mejor score (AUCROC en CV): 0.6822
```

La mejor combinación encontrada corresponde a un árbol con profundidad alta pero regularizado por medio de restricciones como `min_samples_leaf`, `min_samples_split`, `min_weight_fraction_leaf` y un número limitado de nodos hoja. Además, el uso del `splitter` aleatorio y el balanceo de clases introduce aleatoriedad y controla el desbalance.

3.1.2 - Support Vector Machine (SVM)

Se exploraron los siguientes hiperparámetros:

- `C`: Parámetro de regularización. Se evaluaron 20 valores espaciados logarítmicamente entre 10^{-4} y 10^{4}, permitiendo abarcar desde modelos fuertemente regularizados (mayor sesgo) hasta modelos más flexibles (mayor varianza).
- `kernel`: Se probaron los kernels `rbf`, `poly` y `sigmoid`, para evaluar diferentes transformaciones del espacio de entrada y determinar cuál se adapta mejor a la separación de las clases.
- `gamma`: Controla la influencia de cada muestra en la definición del hiperplano separador. Se incluyeron los valores `scale` y `auto`, junto con 20 valores logarítmicos entre 10^{-6} y 10^{1}, lo cual es fundamental para ajustar la complejidad del kernel.
- `degree`: Grado del polinomio, aplicado cuando se utiliza el kernel `poly`. Se evaluaron los valores 2, 3, 4, 5 y 6 para determinar cómo afecta la complejidad del modelo en funciones polinómicas.
- `coef0`: Término independiente que aparece en los kernels `poly` y `sigmoid`. Se consideraron los valores 0.0, 0.1, 0.5 y 1.0.
- `class_weight`: Permite balancear la penalización de las clases. Se incluyeron las opciones `None` y `balanced`, siendo relevante en caso de presentarse desbalance en la variable objetivo.

Los resultados obtenidos fueron los siguientes:

```
Mejores parámetros:
{'kernel': 'rbf', 'gamma': 'scale', 'degree': 2, 'coef0': 1.0, 'class_weight': None, 'C': 206.9138}

Mejor score (AUCROC en CV): 0.9195
```

El mejor modelo corresponde a un SVM con kernel `rbf`, con `gamma scale` y un valor de `C` alrededor de 207. Debido a la alta dimensionalidad del problema, es esperable que el kernel gaussiano performe bien. Finalmente, `class_weight None` indica que a pesar del desbalance los datos se ajustaron correctamente sin necesidad de ponderación adicional.

SVM se posiciona como la mejor configuración entre los modelos evaluados, alcanzando un AUC ROC cercano a 0.92 en validación cruzada.

3.1.3 - K-Nearest Neighbors (KNN)

Se definió el siguiente espacio de búsqueda:

- `n_neighbors`: Define la cantidad de vecinos considerados para la votación de la clase. Variar este parámetro ayuda a balancear la varianza y el sesgo del modelo: más vecinos reducen la varianza, pero pueden aumentar el sesgo.
- `weights`: Establece la forma de ponderar el voto de cada vecino. Usar ponderaciones, como üniforme.º "distancia", puede mejorar el ajuste del modelo especialmente en espacios con muchas dimensiones.

- `metric`: Determina la forma de medir la distancia entre observaciones. Evaluar diferentes métricas permite identificar cuál se ajusta mejor a la distribución y alta dimensionalidad del problema.
- `p`: se usa solo con la métrica `minkowski` y determina qué tan parecida es la distancia a la manhattan (p=1) o a la euclidiana (p=2).

El modelo fue entrenado con 1000 combinaciones aleatorias del espacio anterior. Los mejores resultados obtenidos fueron:

```
Mejores parámetros:
{'weights': 'distance', 'p': 9, 'n_neighbors': 17, 'metric': 'manhattan'}

Mejor score (AUCROC en CV): 0.8864
```

El mejor modelo considera 17 vecinos, utiliza la métrica de distancia `manhattan` y pondera por distancia (`weights = distance`). El uso de `manhattan` en lugar de `euclidean` también podría deberse a la alta dimensionalidad del problema, donde las distancias L1 tienden a ser más discriminantes.

Este modelo logró un AUC ROC de aproximadamente 0.886, una performance buena, aunque ligeramente por debajo del SVM, que alcanzó un score de 0.919.

3.2 - Comparación con otros modelos

Los modelos LDA y Naive Bayes, evaluados con hiperparámetros por defecto, arrojaron los siguientes resultados de AUC ROC en validación cruzada:

Modelo	AUCROC (CV)
SVM (optimizado)	0.9195
KNN (optimizado)	0.8864
Naive Bayes (default)	0.7994
LDA (default)	0.7666
Árbol de Decisión (optimizado)	0.6822

Cuadro 3: Comparación de AUCROC en distintos modelos, donde optimizado refiere al modelo entrenado con la mejor configuración obtenida en Randomized Search

Ambos modelos superaron al árbol de decisión optimizado, pero quedaron por debajo de KNN y SVM. En el caso de LDA, sería relevante explorar los hiperparámetros `solver` y `shrinkage`, que controlan el cálculo y regularización de la matriz de covarianza. Para Naive Bayes, podría considerarse el uso de otras variantes como `MultinomialNB` o `BernoulliNB` si se discretizaran las variables. Pese a su simplicidad, ambos modelos demostraron ser competitivos.

3.3 - Mejor modelo

El mejor modelo fue SVM con kernel `rbf`, `C` $\approx$ `207`, `gamma = scale`, y sin balanceo de clases. Alcanzó un AUC ROC de 0.9195 en validación cruzada.

Este desempeño se explica porque SVM con kernel `rbf` es particularmente efectivo en problemas de alta dimensionalidad como el presente (200 variables), permitiendo separar clases en un espacio transformado sin asumir relaciones lineales. La regularización controlada por `C` y la adaptación de `gamma` al dataset contribuyen a un buen equilibrio entre sesgo y varianza, maximizando su capacidad de generalización.

Ejercicio 4 - Diagnóstico Sesgo-Varianza

4.1 - Curva de complejidad

Por un lado, se analizó la complejidad del modelo SVM a partir del hiperparámetro `C`, que controla el grado de penalización por errores de clasificación, manteniendo el resto de los parámetros fijos según la mejor combinación

obtenida en el ejercicio 3.1.

Se construyó una curva de complejidad variando `C` entre 10^{-3} y 10^{3} en escala logarítmicamente espaciada, y se midieron los valores de AUC ROC en entrenamiento y validación a partir de una validación cruzada de 5 folds.

Por otro lado, se evaluó el impacto del hiperparámetro `max_depth` sobre la complejidad del modelo de árbol de decisión, manteniendo constantes los demás hiperparámetros según la mejor combinación obtenida previamente.

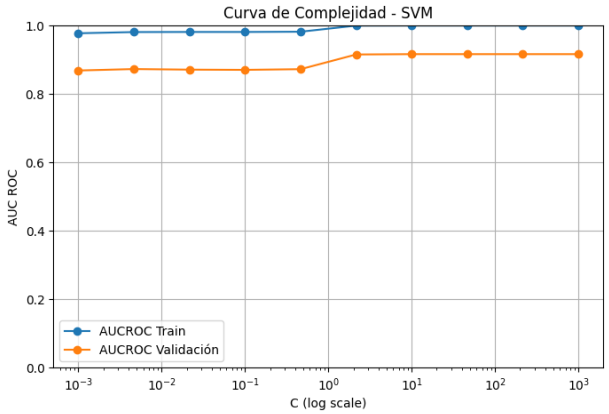

Figura 1: Curva de complejidad para SVM: variación de AUCROC con el hiperparámetro `C`.

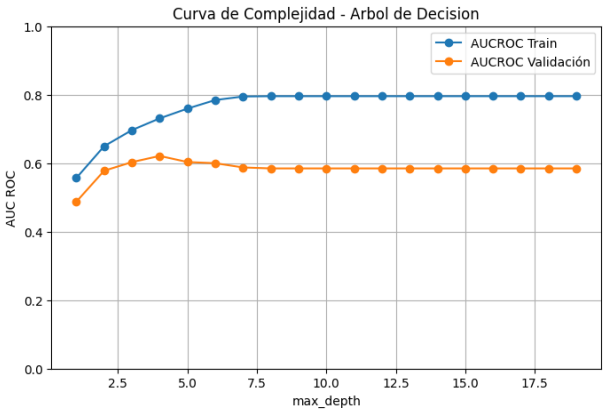

Figura 2: Curva de complejidad para árbol de decisión: variación de AUCROC con el hiperparámetro `max_depth`.

En SVM, el hiperparámetro C define la penalización de errores en la clasificación. A medida que el C aumenta, se penalizan más los errores en el entrenamiento. C chico: se aceptan más errores, el modelo es propenso a underfitting (sesgo alto) C grande: se aceptan menos errores, el modelo es propenso a overfitting (varianza alta) Sin embargo, en la visualización no se ve un umbral de overfitting donde haya mejora en la performance del conjunto de entrenamiento pero empeore en el conjunto de testeo.

Cabe destacar que en la búsqueda realizada mediante `RandomizedSearchCV`, el mejor modelo seleccionado tenía $\texttt{C} \approx 206$. Sin embargo, en la curva de complejidad se observá que configuraciones con $\texttt{C} \approx 2$ se logra un desempeño comparable. Al revisar los resultados de la búsqueda, se comprobó que esta combinación de `C` no había sido evaluada junto con los mejores hiperparámetros restantes. Esto sugiere que una configuración con menor complejidad podría haber sido suficiente para obtener el mejor resultado.

El gráfico de Árbol de Decisión muestra que a medida que se incrementa la profundidad del árbol, el AUC ROC en el conjunto de entrenamiento crece rápidamente hasta alcanzar un techo cercano a 0.83, lo que indica una mejora sostenida en la capacidad del modelo para ajustar los datos. Sin embargo, el AUC en validación se mantiene prácticamente constante alrededor de 0.58, con pequeñas fluctuaciones. Esto muestra que el modelo se vuelve cada vez más complejo y logra memorizar el conjunto de entrenamiento, pero no logra generalizar mejor en datos no vistos. En consecuencia, resulta en overfitting.

La curva sugiere que aumentar la profundidad más allá de 3 o 4 no aporta beneficios en términos de performance de validación, y que un árbol más simple podría performar mejor.

4.2 - Curvas de aprendizaje

Se graficaron curvas de aprendizaje para los tres modelos principales: SVM, Árbol de decisión y LDA.

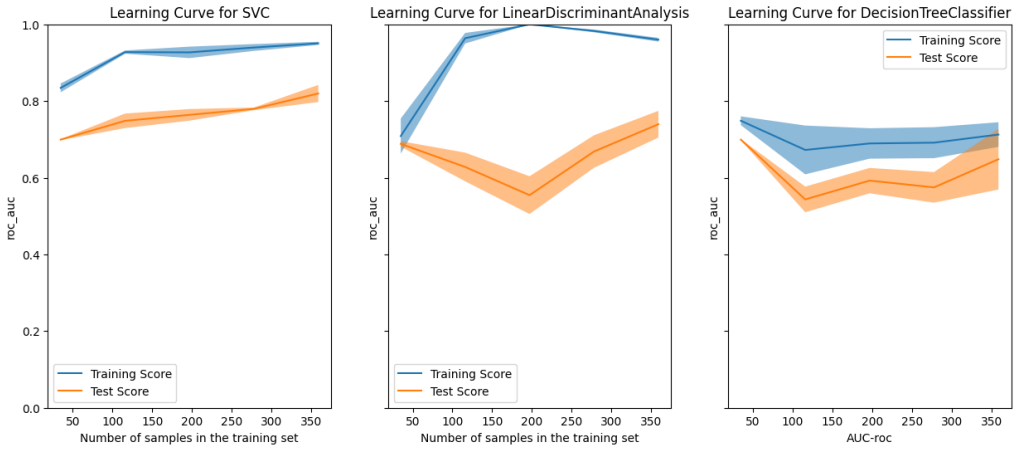

Figura 3: Curvas de aprendizaje para árbol de decisión, SVM y LDA.

En el caso de **SVM**, se observa una tendencia clara y estable de mejora tanto en entrenamiento como en validación a medida que crece el tamaño del conjunto de entrenamiento. Esto indica que el modelo podría beneficiarse de contar con más datos.

Para **LDA**, la curva de entrenamiento alcanza rápidamente un score cercano a 1, mientras que la curva de validación desciende hasta tener un conjunto de entrenamiento de 200 datos y después vuelve a subir y comienza a generalizar al aumentar la muestra.

En contraste, el **árbol de decisión** muestra una tendencia decreciente en entrenamiento y apenas un leve crecimiento en validación, lo que indica que el modelo no está aprendiendo patrones más generales con más datos y probablemente esté limitado por su propia estructura o por una configuración con alta varianza.

En conclusión, **SVM y LDA se beneficiarían de incorporar más datos**, mientras que el árbol de decisión parecería haber alcanzado su límite de performance en este contexto.

4.3 - Random Forest: max_features y curva de aprendizaje

Se entrenó un modelo `RandomForestClassifier` con 200 árboles. El AUC ROC en entrenamiento fue de 1.0, mientras que en validación fue de 0.66, lo cual evidencia un sobreajuste significativo.

Para estudiar el efecto del hiperparámetro `max_features`, se construyó una curva de complejidad evaluando distintas cantidades de features consideradas en cada split de los árboles. Este hiperparámetro define cuántas variables se seleccionan aleatoriamente para evaluar en cada nodo: valores bajos generan mayor diversidad entre los árboles (lo cual favorece la generalización), mientras que valores altos hacen que los árboles se parezcan más entre sí.

Como prueba, realizamos una búsqueda de hiperparámetros con Randomized Search y graficamos la variación de `max_features` con el resto de hiperparámetros pertenecientes a la mejor configuración resultante de la búsqueda. Nuestra intención era comparar el sobreajuste variando `max_features` usando los hiperparámetros default vs. usando los hiperparámetros de la mejor configuración según Randomized Search. Para nuestra sorpresa, los resultados son similares.

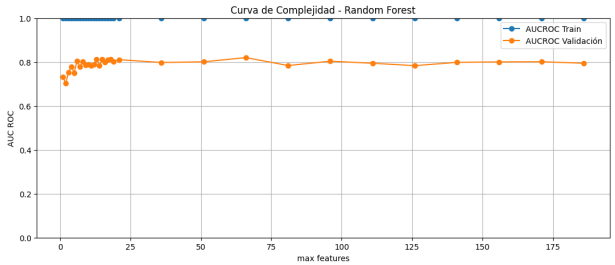

Figura 4: Curva de complejidad del Random Forest de 200 árboles con `max_features=8`.

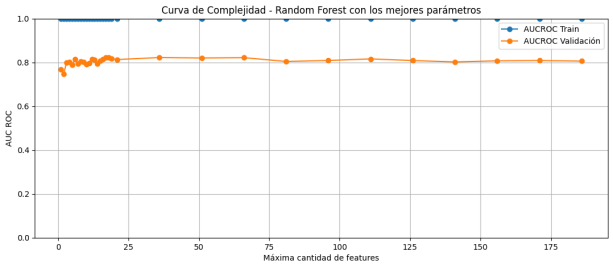

Figura 5: Curva de complejidad del Random Forest con parámetros óptimos del randomized search.

Vemos que, en estas curvas de complejidad, el entrenamiento prácticamente tiene una performance constante en 1, lo cual indica sobreajuste independientemente de la cantidad máxima de features, siendo también relativamente estable la curva de validación alrededor de 0.8.

Análogamente, construimos las curvas de aprendizaje de 200 árboles con los hiperparámetros por default vs los óptimos.

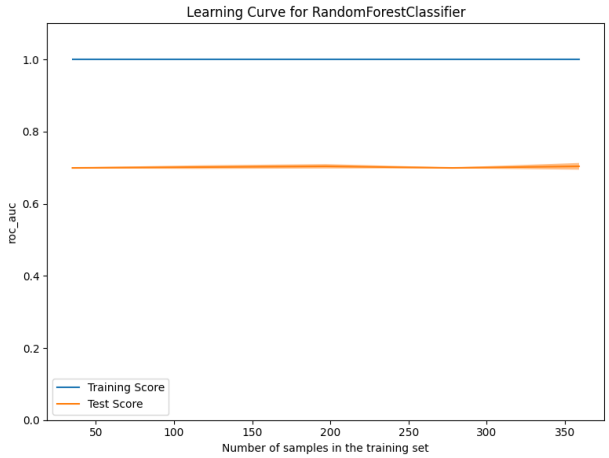

Figura 6: Curva de aprendizaje del Random Forest de 200 árboles con `max_features=8`.

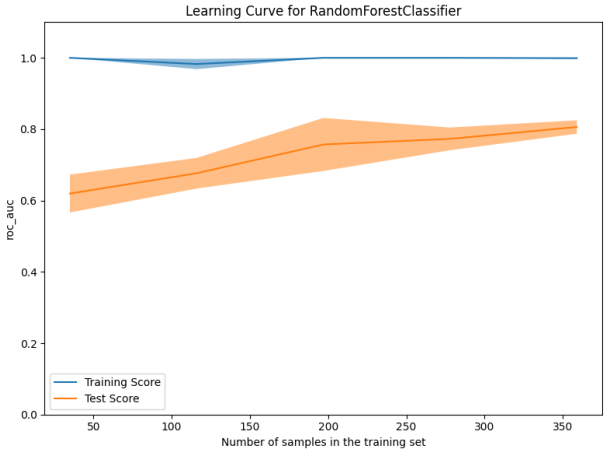

Figura 7: Curva de aprendizaje del Random Forest con parámetros óptimos del randomized search.

Con respecto a las Curvas de Aprendizaje, podemos ver que si utilizamos los hiperparámetros por default de RandomForest hay una performance constante a medida que se aumenta el número de datos (fijando `n_estimators`=200. Si, en cambio, utilizamos los hiperparámetros de la mejor configuración de Randomized Search, se ve cómo a medida que aumenta el número de datos la curva de validación aumenta, lo que da indicio de que está generalizando mejor.

Ejercicio 5

Para realizar la mejor estimación de la performance de nuestro modelo, inicialmente separamos un 10 % de los datos y lo llamamos Base Test. Luego de hacer todo el proceso de selección de modelos, nos quedamos con el mejor y calculamos la performance sobre la Base Test con el modelo que tenía mejor performance, es decir el SVM con los parámetros del Randomized Search. Obtuvimos un AUC de 0.9268.

Conclusiones

En el trabajo abordamos de manera principal la selección de modelos en un contexto de datos ilimitados y desbalanceados. Para manejar estas problemáticas, incluímos estrategias cómo la división de datos estratificada y la inclusión de hiperparámetros que regularicen el desbalance.

También se analizaron con mucho énfasis el peso de distintos hiperparámetros en el overfitting de los modelos. Este punto nos pareció importante ya que, al tener poca cantidad de datos, observamos cómo los modelos entrenados eran muy vulnerables al overfitting.

Además incluímos el análisis de la correlación entre las variables y la búsqueda de hiperparámetros para la comparación de las curvas de aprendizaje y complejidad en Random Forest.

De manera paralela, quisimos realizar un test de permutación (con la ayuda de Chat GPT) de nuestro mejor modelo. El test de permutación consiste en romper la relación entre los features y el target permutando aleatoriamente las etiquetas y entrenando/evaluando el modelo repetidamente. En 39 de 100 iteraciones obtuvimos un AUC mayor a nuestro mejor AUC, lo cual puede indicar que el modelo puede performar bien sin una relación real entre X e y.

Para futuras mejoras en el trabajo, consideramos que puede servir incluir técnicas de Oversampling para manejar

el desbalance de los datos y analizar cómo afecta en los resultados. También nos parece importante incluir un trabajo de análisis e ingeneria de atributos previo al entrenamiento del modelo.

Como conclusión, a pesar de las técnicas que podrían sumarse al desarrollo del modelo, consideramos que los resultados del mejor modelo entrenado son satisfactorios.