Informe de Aprendizaje Automático

Chavez, Mauro
a@gmail.com

Lewkowicz, Iván
a@gmail.com

Drelewicz, Santiago
a@gmail.com

Torrez, Matías
matiastorrez157@gmail.com

Culaciati, Dante
a@gmail.com

29 de abril de 2025

1. Ejercicio 1

En este inciso se pide separar los datos en conjuntos de entrenamiento y evaluación, donde no se debe utilizar la libreria `train_test_split` de `sklearn`.

Primero se realizo una exploracion de los datos, donde se observa que el dataset posee 200 features, todas numericas, y 500 filas. Se observa que el dataset no tiene valores nulos y que ademas se trata de un problema desbalanceado, donde el %70 de los datos pertenecen a la clase 1 y el % restante pertenece a la clase 0, por lo que no es necesario realizar un preprocesamiento de los datos. Se decide entonces utilizar el 80 % de los datos para entrenamiento y el 20 % restante para evaluacion.

Como la proporción de los datos es desbalanceada, realizamos un `stratified split` en la separación de los datos, procurando mantener la proporción del dataset original para los datos de entrenamiento y evaluación.

2. Ejercicio 1.1

3. Ejercicio 2

Para la primera parte de este ejercicio, entrenamos un arból de decisión con altura máxima 3 y estimamos la performance del modelo con K fold cross validation para distintas métricas. Las metricas utilizadas son *Accuracy*, *AUPRC* y *AUC ROC* y se realizo un *K-fold* con $K = 5$.

En la tabla 1 se muestran los resultados obtenidos para cada una de las 5 permutaciones de los datos, asi como el promedio de cada métrica para todas las permutaciones y el resultado global, el cual se obtiene al calcular las métricas utilizando el conjunto de predicciones formado a partir de concatenar las predicciones de cada fold.

Permutación	**Accuracy** (training)	**Accuracy** (validación)	**AUPRC** (training)	**AUPRC** (validación)	**AUC ROC** (training)	**AUC ROC** (validación)
1	0.8125	0.6375	0.6710	0.3226	0.8058	0.5298
2	0.840625	0.5875	0.7337	0.3337	0.8458	0.5246
3	0.825	0.6875	0.6431	0.3437	0.7513	0.5811
4	0.81875	0.7	0.6573	0.3626	0.7877	0.5938
5	0.84375	0.65	0.6958	0.4144	0.8085	0.5967
Promedios	0.828125	0.6525	0.6802	0.3554	0.7998	0.5651
Global	(NO)		(NO)		(NO)	

Cuadro 1: Resultados por permutación y métricas

Se observa que este modelo presenta un buen desempeño en el conjunto de entrenamiento, pero su desempeño en el conjunto de validación es bastante bajo, lo que podría indicar que el modelo está sobreajustado a los datos de entrenamiento.

Para la segunda parte del ejercicio, se exploraron diferentes combinaciones de hiperparámetros para el modelo de árbol de decisión, utilizando `GridSearchCV` de `sklearn`. Se probaron diferentes valores para la profundidad máxima del árbol y el cirterio de corte. Se utilizó `StratifiedKFold` con $K = 5$ para la validación cruzada. En la tabla 2 se muestran los resultados obtenidos para cada combinación de hiperparámetros, así como el promedio de *Accuracy* para cada combinación.

Altura máxima	**Criterio de corte**	**Accuracy** (training)	**Accuracy** (validación)
3	Gini	0.6375	0.6710
5	Gini	0.5875	0.7337
Infinito	Gini	0.6875	0.6431
3	Entropía	0.7	0.6573
5	Entropía	0.65	0.6958
Infinito	Entropía	0.828125	0.828125

Cuadro 2: Resultados por permutación y métricas

4. Ejercicio 3

En esta seccion se exploraron diferentes combinaciones de hiperparametros para los modelos de arboles de decision, *KNN* y *SVM* y se los comparo contra los algoritmos *LDA* y *GaussianNB* sin realizar

una busqueda de hiperparametros . Se buscó identificar el mejor modelo de cada familia de algoritmos buscando maximizar el *AUC ROC*.

4.1. Metodologia

Para realizar la busqueda de hiperparametros y la estimacion del rendimieinto de los modelos se utilizo la tecnica de *Nested Cross Validation*, utilizando *RondomizedSearchCV* para la busqueda de hiperparametros y, al igual que en la seccion anterior, se utilizo *StratifiedKFold* para la creacion de los *Kfolds*. *Nested Cross Validation* es una técnica que permite evaluar el rendimiento de un modelo de aprendizaje automático mientras se optimizan sus hiperparámetros. En este enfoque, se utilizan dos bucles de validación cruzada: uno externo para evaluar el rendimiento del modelo y otro interno para ajustar los hiperparámetros. Esto ayuda a evitar el sobreajuste y proporciona una estimación más precisa del rendimiento del modelo en datos no vistos, en la figura 1 se muestra un esquema de como se realiza este proceso.

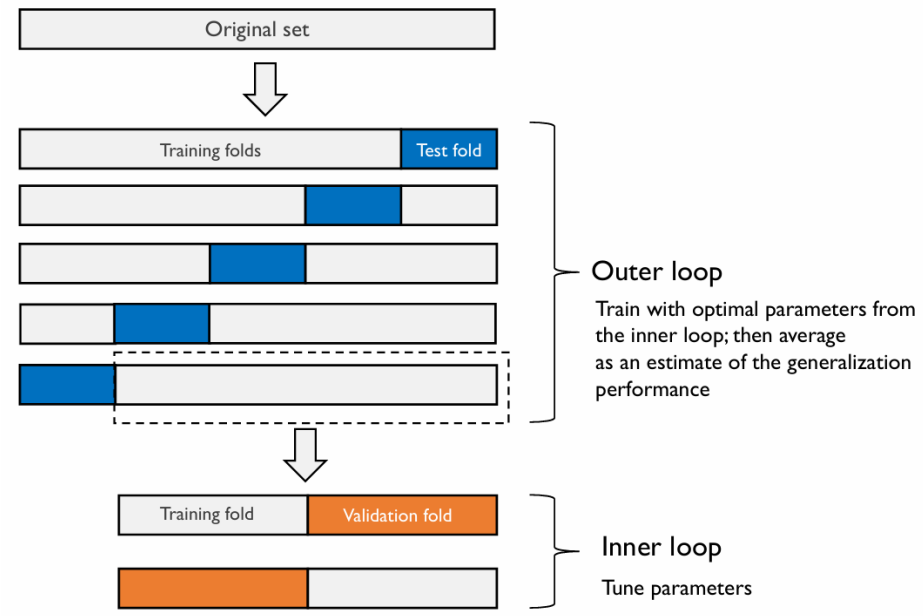

Figura 1: Esquema de *Nested Cross Validation*, donde el bucle interno se utiliza para la búsqueda de hiperparámetros y el bucle externo para la evaluación del rendimiento.

Para poder asegurar que los resultados sean reproducibles, se utilizó un valor fijo para la semilla aleatoria en la creación de los *Kfolds* y en la búsqueda de hiperparámetros. Para asegurar una buena exploración del espacio de hiperparámetros se realizaron 150 iteraciones de *RandomizedSearchCV* con semilla fija.

Para el caso de los algortimos *SVM* y *KNN* se aplico una normalizacion de los datos, utilizando `StandardScaler` de `sklearn`, para asegurar que todas las features tengan la misma escala y evitar que algunas características dominen el proceso de entrenamiento. Ya que a diferencia de los arboles de decision, estos algoritmos son sensibles a la escala de los datos.

4.2. Arbol de decisión

A continuación se detallan los hiperparámetros utilizados junto con sus respectivos valores de prueba:

- `max_depth` (int o Ninguno): Altura máxima del árbol, mientras mas profundo sea el árbol, más complejo será el modelo y habra mas posibilidades de sobreajusta.
- `criterion` (gini, entropia): Criterio de corte,
- `min_samples_leaf`: Cantidad mínima de muestras por hoja, se probaron valores entre 0 y 1,0 con una distribucion uniforme. Este rango represnte la fraccion de datos a considerar sobre el total

- `min_samples_split`: Cantidad mínima de muestras para dividir un nodo, se probaron valores entre 0 y 1,0 con una distribucion uniforme. Este rango represnte la fraccion de datos a considerar sobre el total
- `class_weight`: Peso de las clases, se probaron los valores `balanced` y `None`.
- `max_features`: Cantidad máxima de features a considerar para cada división, se probaron los valores `sqrt`, `log2` y `None`.

Para el caso del hiperparametro `max_depth` el rango elegido busca tratar de explorar tanto arboles cortos y profundos. Se probaron los criterios de Gini y entropia ya que, si bien ambos son usados como criterios de corte para el arbol, representan cosas diferente. Mientras Gini se puede ineterpretar como una medida de impureza al clasificar,la entropia se puede interpretar como una medida de incertidumbre. Los rangos de `min_samples_leaf` y `min_samples_split` tratan de cubrir todo el rango de proporciones de los datos. El hiperparametro *class_weight* maneja el peso asignado a cada clase,las elecciones posibles buscan estudiar si balancear los pesos a partir de la distribucion de clases mejora el modelo o, si a pesar de ser un problema imbalanceado, el asignarles igual peso a amb as clases mejora el desempeño del arbol.

4.3. *KNN*

A continuación se detallan los hiperparámetros utilizados junto con sus respectivos valores de prueba:

- `n_neighbors`: Cantidad de vecinos a considerar, se probaron valores aleatorios en el rango de 3 a 50.
- `weights`: Estrategia de ponderación, se probaron los valores `uniform` y `distance`.
- `metric`: Métrica de distancia, se probaron las métricas `euclidean`, `manhattan`, `minkowski` y `cosine`.
- `p`: Potencia de la métrica de Minkowski, se probaron valores aleatorios entre 1 y 4.
- `algorithm`: Algoritmo de búsqueda de vecinos, se probaron los valores `auto`, `ball_tree`, `kd_tree` y `brute`.
-

Para el caso de la seleccion de rango de `n_neighbors`, sabiendo de antemano el tamaño del dataset de desarollo , el cual era de 350 filas, se decidio probar un rango suficientemente grande de vecinos, donde el mayor numero posible es 50 lo cual equivale a un 14 % de los datos. En el caso de los pesos, se decidio probar tanto la ponderacion uniforme como la ponderacion por distancia, ya que en el caso de que los datos esten desbalanceados, la ponderacion por distancia puede ayudar a mejorar el desempeño del modelo. Se probaron diferentes metricas de distancia, . La eleccion de valores para el hiperparametro del algoritmo busca explicitar los algoritmos a utilizar, ya que por defecto *KNN* utiliza el algoritmo `auto` que selecciona el mejor algortimos y se busco probar todas las opciones posibles.

4.4. *SVM*

A continuación se detallan los hiperparámetros utilizados junto con sus respectivos valores de prueba:

- `C`: Parámetro de regularización, se probaron valores aleatorios entre 0,001 y 1000.
- `kernel`: Tipo de kernel a utilizar, se probaron los valores `linear`, `poly`, `rbf` y `sigmoid`.
- `degree`: Grado del polinomio, se probaron valores aleatorios entre 1 y 5.
- `gamma`: Coeficiente del kernel, se probaron valores aleatorios entre 0,01 y 1.

El valor de C se elijo de tal forma que se busque explorar tanto un modelo con alta regularizacion como uno con baja regularizacion. Debido a que la dimensionaldiad del dataset es alta, se decidio probar difernte tipos de kernels, ya que el kernel lineal puede no ser el mejor para este tipo de datos. El rango de valores para el hiperparametro `degree` se eligio para poder darle libertad de complejidad al modelo en el caso del kernel polinomial.

4.5. *LDA* y *GaussianNB*

A pesar de que para comparar se utilizaron los algoritmos de *LDA* y *GaussianNB* sin realizar una búsqueda de hiperparámetros, se mencionan los posibles hiperparámetros que podrían ser utilizados para estos algoritmos:

- *LDA*:
  - `solver`: Algoritmo de optimización, se pueden probar los valores `svd`, `lsqr` y `eigen`.
  - `shrinkage`: Método de regularización, se pueden probar los valores `auto` y `None`.
  - `tol`: Tolerancia para la convergencia, se pueden probar valores aleatorios entre 0,0001 y 0,1.
- *GaussianNB*:
  - `var_smoothing`: Parámetro de suavizado de varianza, se pueden probar valores aleatorios entre $1e-9$ y $1e-1$.

4.6. Resultados

En la tabla 3 se muestran los resultados obtenidos para cada modelo, junto con el promedio del *AUC ROC* y su desviacion estandar. Obtenidos a partir del metodo de *Repeated-KFold NestedCV* mencionado en la seccion 4.1.

Modelo	Mejor hiperparámetros
Árbol de decisión	`class_weight`= None, `criterion`=`entropy`, `max_depth`=6, `max_features`=0.3056, `min_samples_l`
KNN	'algorithm'='brute', 'metric'='cosine', 'n_neighbors'=8, 'weights'=
SVM	C=0,171, coef0=0.069,degree=4, gamma='scale', 'kernel'= 'p
LDA	Modelo base
GaussianNB	Modelo base

Cuadro 3: *AUC ROC* de los modelos probados junto con sus mejores hiperparámetros.

Se observa que el mejor modelo en terminos de *AUC ROC* es *SVM* con un kernel polinomico de grado 4, seguido por *KNN*. Esto era esperable, ya que tanto *SVM* como *KNN* son algortimos que pueden captar relaciones no lineales en los datos.

5. Ejercicio 4

6. Ejercicio 5

En base a todo lo anteriormente mencionado, se decide utilizar el modelo de *SVM* con kernel polinómico de grado 4 para realizar la estimacion del *AUC ROC* a reportar sobre el conjunto de evaluacion que se separó en el ejercicio 1

7. Conclusión

Resuma los hallazgos principales y las conclusiones del informe.

Referencias

Incluya las referencias bibliográficas utilizadas en el informe.