Machine Learning I

Taller N°2: Comparación entre Distintos Algoritmos de Aprendizaje de Machine Learning

Alejandro Gil Gallego (274361), Juan David Sánchez González (272906)
Universidad La Sabana, Chia, Cundinamarca
alejandrogiga@unisabana.edu.co, juansangon@unisabana.edu.co
Maestría en Análitica Aplicada
Profesor: Hugo Franco, Ph.D.
2 de octubre de 2025

Resumen

Este documento compara el desempeño de cuatro algoritmos de clasificación supervisada aplicados a tres diferentes datasets, usando un workflow completo de machine learning. Se analizaron diferentes métricas de comparación entre todos los modelos para evaluar su rendimiento, tiempos de entrenamiento y correcta implementación del preprocesamiento de datos.

Index Terms

Knn, Random Forest, SVM, Decision Tree, dataset, modelos de clasificación

I. Introducción

El taller fue propuesto como parte del curso de machine learning I, este tenía como objetivo la comparación de diferentes algoritmos, de la precisión y desempeño de cada uno para datasets con comportamientos distintos entre si.

Las instrucciones para la realización del taller fueron las siguientes:

1. Completar los casos entregados usando en mayor medida un workflow de entrenamiento supervisado antes del proceso de selección del modelo completo.
2. Corregir, en caso de ser necesario, el workflow de acuerdo a los conceptos presentados en clase.
3. Utilizar todos los modelos vistos en cada problema y comparar sus desempeños. Incluyendo Random Forest
4. Preguntas a responder:
   - ¿Qué le sucede al desempeño si se salta el paso de feature scaling en kNN y en SVM?
   - ¿Cuál modelo fue el más rápido de entrenar?
   - ¿Qué tan bueno fue el desempeño del Random Forest comparado con los otros modelos?
   - ¿Qué tanto afecto al desempeño el cambiar los hiperparamétros?
   - Mirando al reporte de clasificación. ¿Las clases quedaron bien clasificadas por el mejor modelo?

II. Metodología

Para solucionar el taller, se compararon cuatro modelos de clasificación usando tres datasets diferentes, que se mencionan a continuación:

Los datos utilizados para el desarrollo del taller provienen de tres fuentes principales:

1. `Iris Dataset`: Este dataset contiene medidas de las características florales como la longitud y el ancho de sépalos y pétalos de tres especies de iris. Usado para clasificación multiclase.
2. `Diabetes Dataset`: Incluye variables médicas y características de los pacientes para predecir la presencia de diabetes en clasificación binaria.
3. `Breast Cancer Dataset`: Este dataset contiene características de tumores de cáncer de mama obtenidas a partir de imágenes digitalizadas para clasificar entre tumores benignos y malignos.

La metodología se dividió en cuatro etapas principales que permitieron desarrollar el trabajo de una manera correcta y bien estructurada.

- **Preprocesamiento de datos:** Los datos se dividieron en 70 % para entrenamiento y 30 % para prueba. Se seleccionaron las características más relevantes de cada conjunto de datos y se aplicó normalización con *StandardScaler* para los modelos que lo requieren.

- **Entrenamiento de los modelos:** Se entrenaron los cuatro algoritmos de clasificación supervisada, los cuales fueron, decision Tree, Random Forest, k-Nearest Neighbors y Support Vector Machine.
- **Métricas de evaluación:** Se midió el accuracy, se generaron matrices de confusión, se implementaron reportes de clasificación y se registraron los tiempos de entrenamiento de cada modelo.
- **Implementación de hiperparámetros a los modelos de Knn y SVM:** Se llevó a cabo una implementación de cambio de hiperparámetros para estos modelos, con el fín de observar como esto afecta a su rendimiento.

II-A. Preprocesamiento de datos

El primer paso del taller fue realizar la carga de los datasets mencionados anteriormente, en los cuales para el dataset de Iris se utilizó un archivo CSV local, mientras que para Diabetes se descargó desde un repositorio en línea, y por último para el dataset de Breast Cancer se cargó con ayuda de scikit-learn.

```
Algoritmo 1. Obtención del dataset Iris

Función iris_dataset_retribution(): DataFrame

   Leer archivo 'iris.csv' como DataFrame
   retornar df
```

La implementación detallada del algoritmo 1 se encuentra en el archivo adjunto Comparison.ipynb, disponible en el repositorio de GitHub del proyecto [4].

```
Algoritmo 2. Obtención del dataset de Diabetes

Función diabetes_dataset_retribution(): DataFrame

   Definir lista nombre_columnas

   Hacer petición HTTP GET a la url del dataset de diabetes
   Guardar el contenido en la variable csv_data
   Leer csv_data como Dataframe

   retornar dataset
```

La implementación detallada del algoritmo 2 se encuentra en el archivo adjunto Comparison.ipynb, disponible en el repositorio de GitHub del proyecto [4].

```
Algoritmo 3. Obtención del dataset de Cáncer de mama

Función cancer_dataset_retribution(): DataFrame

   Cargar dataset de cáncer de mama desde la librería sklearn.datasets
   Imprimir nombres de las características contenidas en dataset.feature_names

   retornar dataset
```

La implementación detallada del algoritmo 3 se encuentra en el archivo adjunto Comparison.ipynb, disponible en el repositorio de GitHub del proyecto [4].

Una vez cargados los datos, se procedió a realizar la separación entre las variables predictorias que para este caso corresponden a las características, de la variable objetivo. Para el dataset de Iris, se seleccionaron las cuatro medidas de las flores como características, y la especie como la variable objetivo. Para Diabetes, se eligieron siete características médicas relevantes como embarazos, insulina, indice de masa corporal, edad, glucosa, presión arterial y función de pedigree. En Breast Cancer se utilizaron todas las características del dataset.

El siguiente paso fue dividir los datos en conjuntos de entrenamiento y prueba. Aquí se hizo uso de la función `train_test_split` con una proporción de 70 % para entrenamiento y 30 % para prueba [1]. Se fijó un valor de `random_state` para asegurar la consistencia de los resultados en las futuras ejecuciones.

```
Algoritmo 4. División del dataset en entrenamiento y prueba

Función preprocessing_dataset(dataset): (X_train, X_test, y_train, y_test)
```

```
   Definir variables predictorias (X) a partir del dataset
   Definir variable objetivo (y) a partir del dataset

   Aplicar train_test_split(X, y, test_size = 0.3, random_state = valor_fijo)

   retornar X_train, X_test, y_train, y_test
```

La implementación detallada del algoritmo 4 se encuentra en el archivo adjunto Comparison.ipynb, disponible en el repositorio de GitHub del proyecto [4].

Es importante mencionar que en el preprocesamiento de datos no se aplicó la normalización, debido a que más adelante, esta se implementó con StandarScaler de manera específica para los modelos de Knn y SVM. Esta elección permitió evaluar y comparar cómo cambia el rendimiento de los modelos con y sin normalización de los datos.

```
Algoritmo 5. Estrategia de normalización de datos

Función normalización_preprocesamiento():

   si modelo pertenece a {Knn, SVM}:
      aplicar StandarScaler a X_train y X_test
   En otro caso:
      no aplicar normalización

   retornar datos procesados
```

La implementación detallada del algoritmo 5 se encuentra en el archivo adjunto Comparison.ipynb, disponible en el repositorio de GitHub del proyecto [4].

Finalmente, se verificó que la división de los datos fueran correctas, para así asegurar que los datos estuvieran listos para entrenas los modelos.

II-B. Entrenamiento de los modelos

Como se mencionó al iniciar el informe, se implementaron cuatro algoritmos de clasificación supervisada, los cuales fueron: Decision Tree, Random Forest, Knn y SVM. A continuación, se detalla cada uno de ellos.

II-B1. Decision Tree: Para este modelo se utilizó el clasificador DecisionTreeClassifier, el cual permitió crear un árbol de decisión que divide los datos según las características más relevantes. El entrenamiento se llevó a cabo ajustando el modelo a los datos con ayuda del método fit(), mientras que la etapa de predicción se realizó con predict(). Adicionalmente, se implementó la validación cruzada para evaluar su desempeño durante el entrenamiento.

```
Algoritmo 6. Clasificación con Decision Tree

Función decision_tree(X_train, X_test, y_train, y_test): modelo, predicciones

   Inicializar clasificador de decision tree
   Entrenar modelo con X_train y y_train
   Realizar predicciones sobre x_test
   Evaluar desempeño con validación cruzada y métricas de exactitud

   retornar modelo, predicciones
```

La implementación detallada del algoritmo 6 se encuentra en el archivo adjunto Comparison.ipynb, disponible en el repositorio de GitHub del proyecto [4].

II-B2. Random Forest: El Random Forest se implementó con RandomForestClassifier, configurado con n_estimators igual a 100 y una semilla aleatoria fija de 42 en este caso para garantizar la consistencia de los resultados. Este modelo combinó múltiples árboles de decisión para mejorar la precisión y reducir el sobreajuste. Además como lo indicaba el taller, este se probó .ºut-of-the-box", es decir, sin ajuste de hiperparámetros adicionales.

```
Algoritmo 7. Clasificación con Random Forest

Función random_forest(X_train, X_test, y_train, y_test): modelo
```

```
   Inicializar clasificador Random Forest
   Entrenar modelo con X_train y y_train
   Realizar predicciones sobre x_test
   Evaluar desempeño con métrica de exactitud

   retornar modelo
```

La implementación detallada del algoritmo 7 se encuentra en el archivo adjunto Comparison.ipynb, disponible en el repositorio de GitHub del proyecto [4].

II-B3. K-Nearest Neighbors: Para este modelo se hizo uso de KneighborsClassifier considerando un único vecino, lo que significa que clasifica cada punto según su instancia más cercana. Se implementaron dos versiones, una implementando el StandarScaler para normalizar los datos y la otra sin este. Esto permitió evaluar que tanto afecta el preprocesamiento en el rendimiento del modelo y de la misma manera se aplicó validación cruzada.

```
Algoritmo 8. Clasificación con Knn

Función knn(X_train, X_test, y_train, y_test): modelo

   Inicializar clasificador Knn con k definido
   Entrenar modelo con X_train y y_train
   Realizar predicciones sobre x_test
   Evaluar desempeño con validación cruzada y exactitud

   retornar modelo
```

La implementación detallada del algoritmo 8 se encuentra en el archivo adjunto Comparison.ipynb, disponible en el repositorio de GitHub del proyecto [4].

II-B4. Support Vector Machine: Se usó el clasificador SVC con un kernel de RBF para entrenar el modelo. Se probaron también dos versiones al igual que en el modelo de Knn, en la cual una fue sin estandarización y en la otra los datos se transformaron con StandarScaler. Esto permitió comparar como influye el preprocesamiento en el aprendizaje del modelo [2].

```
Algoritmo 9. Clasificación con SVM

Función svm(X_train, X_test, y_train, y_test): modelo

   Inicializar clasificador SVM con kernel definido
   Entrenar modelo con X_train y y_train
   Realizar predicciones sobre x_test
   Evaluar desempeño con validación cruzada y exactitud

   retornar modelo
```

La implementación detallada del algoritmo 9 se encuentra en el archivo adjunto Comparison.ipynb, disponible en el repositorio de GitHub del proyecto [4].

II-C. Métricas de evaluación

Para medir el desempeño de cada modelo se utilizaron diferentes métricas que permiten evaluar qué tan bien están funcionando los algoritmos de clasificación.

La métrica principal usada fue el accuracy la cual permite medir el porcentaje de predicciones correctas sobre el total de predicciones [3], este se calculó tanto en el conjunto de entretamiento como en el conjunto de prueba de cada modelo. También se generaron matrices de confusión para observar el desempeño de cada modelo, mostrando así cuantas predicciones fueron correctas y cuantas fueron erróneas.

Además, se implementó el reporte de clasificación el cual permitió analizar más métricas detalladas por clase, como precision, recall y f1-score. Y finalmente se calcularon las curvas ROC y el área bajo la curva AUC, que mide la capacidad del modelo para diferenciar entre las clase positiva y negativa.

II-D. Implementación de hiperparámetros a los modelos de Knn y SVM

Para evaluar el efecto de los hiperparámetros en el desempeño de los modelos de Knn y SVM, se implementaron varias modificaciones de estos. En el caso del Knn, se probaron cuatro valores del hiperparámetro sobre los datos sin escalar. Para

SVM, se evaluaron seis combinaciones diferentes también para los datos sin escalar. Este proceso, permitió observar el efecto de cada hiperparámetro en el desempeño del modelo.

```
Algoritmo 10. Hiperparámetros en Knn

Función evaluar_knn(X_train, X_test, y_train, y_test): modelo

   Para cada valor de k en {1,3,5,7}:
      Inicializar clasificador Knn con n_neighbors = k
      Entrenar modelo con X_train y y_train
      Calcular accuracy en X_test
      Mostrar resultados

   Comparar desempeños y seleccionar k con mejor resultadp
   retornar mejor modelo
```

La implementación detallada del algoritmo 8 se encuentra en el archivo adjunto Comparison.ipynb, disponible en el repositorio de GitHub del proyecto [4].

```
Algoritmo 11. Hiperparámetros en SVM

Función evaluar_SVM(X_train, X_test, y_train, y_test): modelo

   Para cada kernel en {linear, rbf}:
      Para cada valor de C en {0.1, 1, 10}
      Inicializar clasificador SVM
      Entrenar modelo con X_train y y_train
      Calcular accuracy en X_test
      Mostrar resultados

   Comparar desempeños y seleccionar con mejor resultadp
   retornar mejor modelo
```

La implementación detallada del algoritmo 8 se encuentra en el archivo adjunto Comparison.ipynb, disponible en el repositorio de GitHub del proyecto [4].

III. Resultados

Se logró la creación de un código funcional que logra responder a todos las instrucciones que se nos fueron mencionadas y que estan nombradas en la introducción.

Entre los resultados estan las matrices de confusión resultantes de cada uno de los modelos para cada uno de los dataset.

III-A. ***Iris.csv***

De la figura 1-4 se nos muestran las matrices de confusión hechas en iris.csv. Por la naturaleza del dataset todas las matrices son iguales, por lo que los resultados de los 4 modelos fueron iguales. Esto puede llegar a ser confundido con overfitting pero en este caso ocurre por el tamaño del dataset.

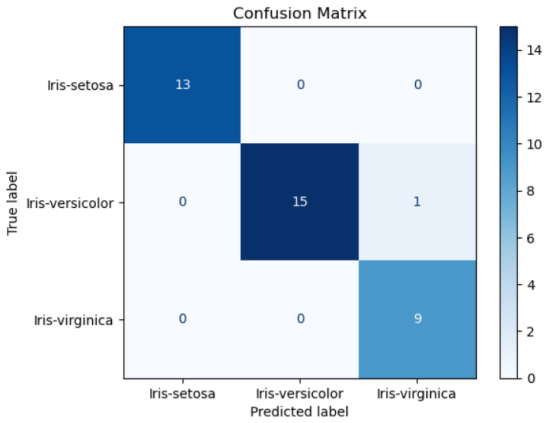

Figura 1. Decision Tree para iris.csv

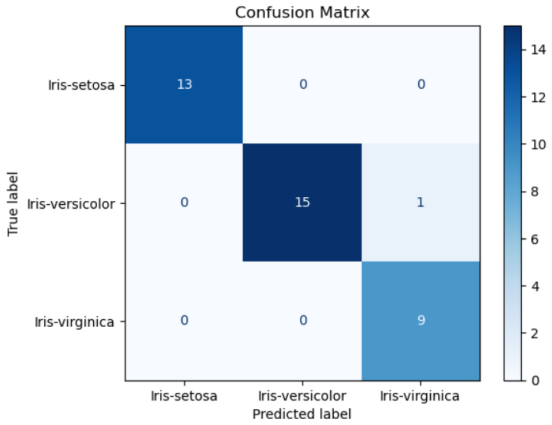

Figura 2. Random Forest para iris.csv

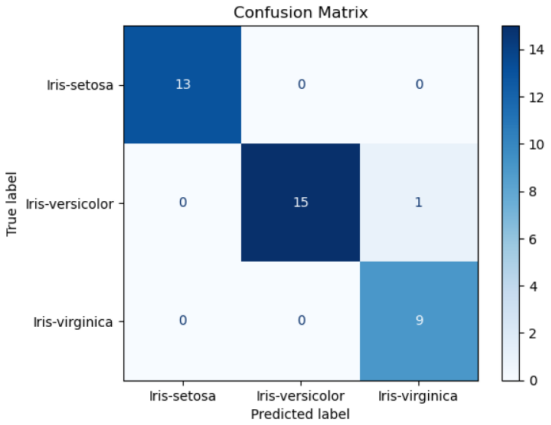

Figura 3. kNN para iris.csv

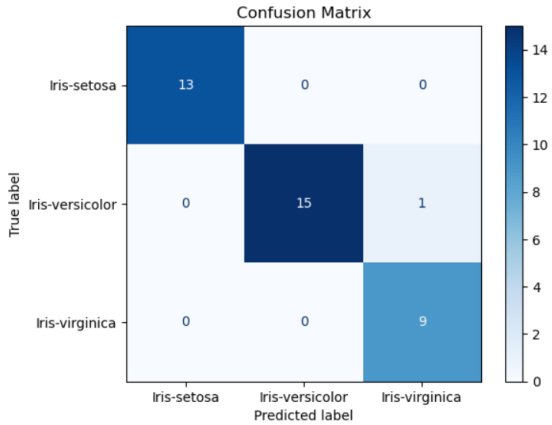

Figura 4. SVM para iris.csv

III-B. ***Cancer.csv***

De la figura 5-8 se nos muestran las matrices de confusión de los modelos para el dataset de cancer.csv, tomando en cuenta el mejor modelo a aquel con menor cantidad de datos mal designados, en ese caso el mejor modelo es el Random Forest como o muestra la figura 6 con solo 4 datos mal ubicados, seguido de kNN y Decision Tree, mientras que el peor modelo para este data set es SVM como lo muestra la figura 8.

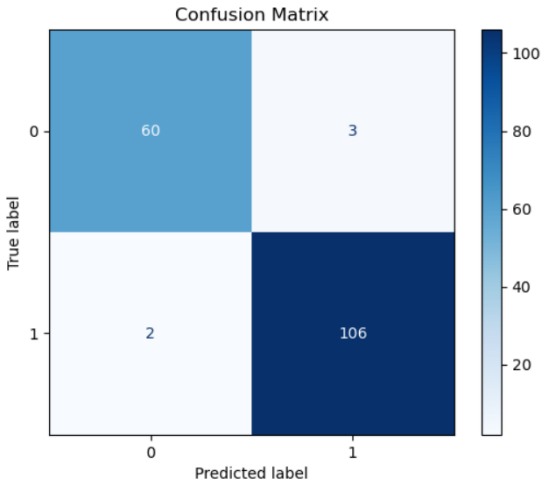

Figura 5. Decision Tree para cancer.csv

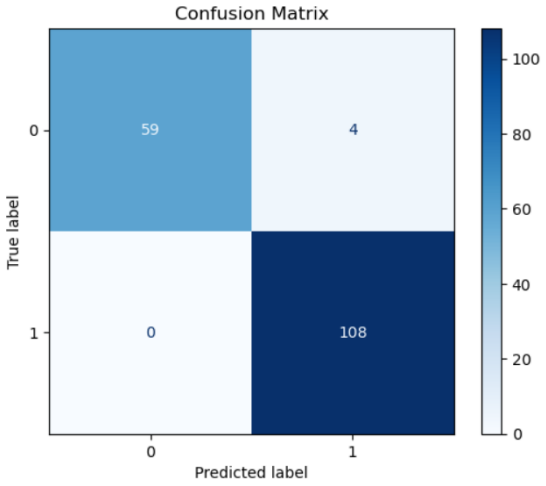

Figura 6. Random Forest para cancer.csv

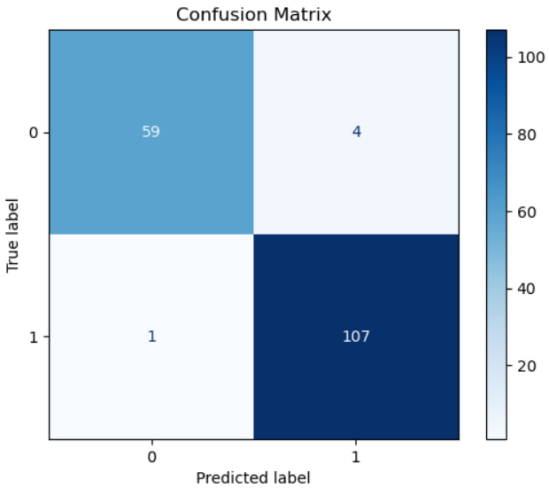

Figura 7. kNN para cancer.csv

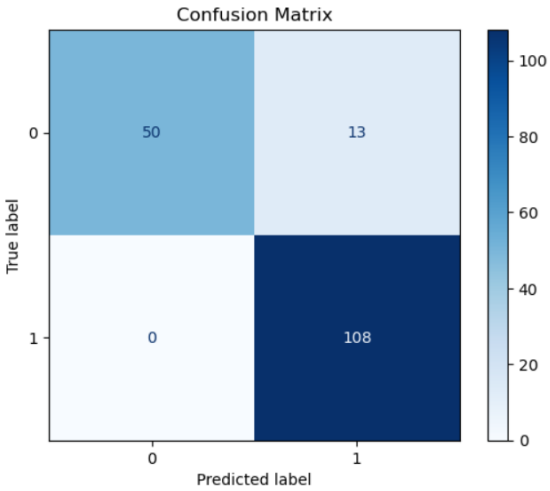

Figura 8. SVM para cancer.csv

III-C. ***Diabetes.csv***

Las figuras 9-12 nos muestran las matrices de confusión para cada algoritmo para el dataset de diabetes, rigiendonos por la misma vara que para cancer.csv el mejor modelo según la matriz de confusión es el SVM como lo muestra la figura 12. Seguido de kNN y Random Forest, siendo el peor el de la figura 9, el Decision Tree.

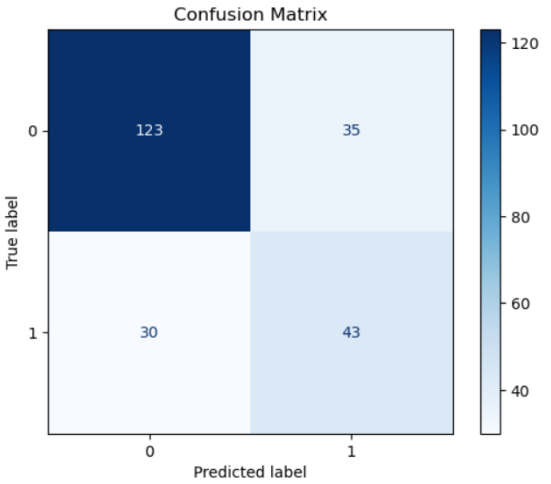

Figura 9. Decision Tree para diabetes.csv

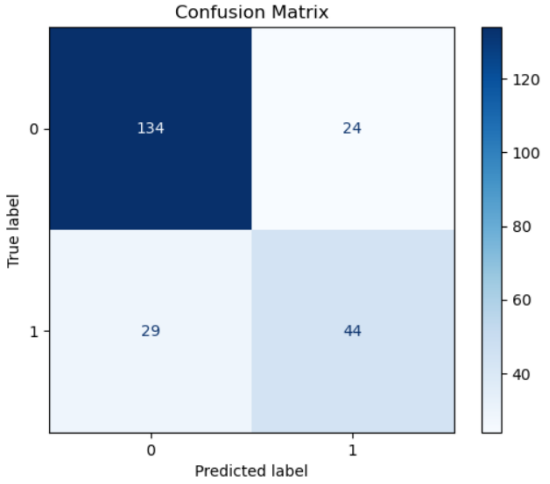

Figura 10. Random Forest para diabetes.csv

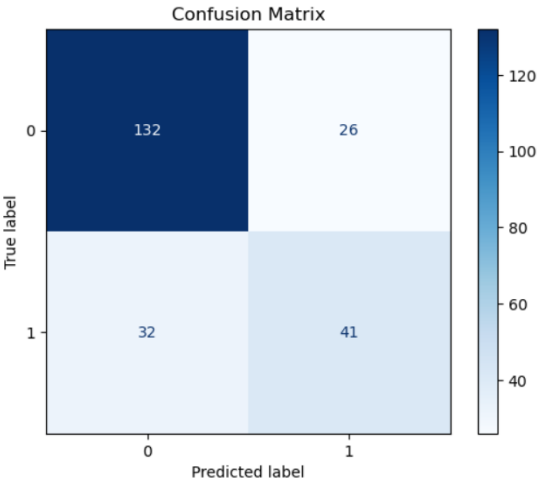

Figura 11. kNN para diabetes.csv

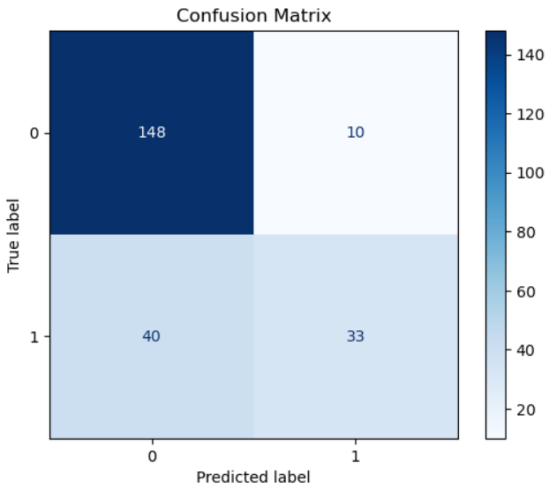

Figura 12. SVM para diabetes.csv

*III-D. **Preguntas***

Así mismo fue necesario la respuesta de las preguntas planteadas.

1. **¿Qué le sucede al desempeño si se salta el paso de feature scaling en kNN y en SVM?** Se logró evidenciar que con feature scaling el rendimiento mejora, en el caso de diabetes svm dio 0.74 y svm scaler 0.78.
2. **¿Cuál modelo fue el más rápido de entrenar?** Decision tree, para iris.csv demoró 0.05s, mientras que kNN demoró 0.06s, SVM 0.07s y Random Forest 0.22s. Para cancer.csv, Decision tree tardó 0.04s, kNN demoró 0.05s, SVM 0.05s y Random Forest 0.16s. Y por último para diabetes.csv, Decision tree demoró 0.04s, kNN 0.06s, SVM 0.10s y Random Forest 0.15s. En total el modelo más rápido fue Decision Tree y el más lento el de Random Forest.
3. **¿Qué tan bueno fue el desempeño del Random Forest comparado con los otros modelos?** Tuvo un rendimiento muy bueno, casi igual a los de los demás con un accuracy de 0.9737 para iris.csv, para cancer.csv 0.97 y para diabetes.csv 0.77, para este último tuvo una mejor en comparación con decision tree que dio un accuracy de 0.71.
4. **¿Qué tanto afecto al desempeño el cambiar los hiperparamétros?** Al realizar pruebas cambiando los hiperparametros manejados tanto en SVM como en kNN se llego a las siguientes conclusiones: El peor hiperparámetro para SVM para todos los datasets es kernel=rbf y C=0.1. Para kNN en iris.csv no se evidencia diferencia entre los hiperparámetros, mientras que para cancer.csv y diabetes.csv el mejor hiperparametro es k=7. En SVM en iris.csv casi todos los hiperparametros son buenos, para cancer.csv el mejor es kernel=linear y C=0.1, mientras que para diabetes.csv es mejor kernel=rbf y C=1.
5. **Mirando al reporte de clasificación. ¿Las clases quedaron bien clasificadas por el mejor modelo?** No todos las classes son predecidas de la misma forma, y en especial ocurre la pecularidad que el classification report presento muy

bajos rendimientos al trabajar con feature scaling. Los resultados tendieron a ser consistentes, en el caso de iris.csv por el tamaño del dataset daba un desempeño de casi siempre 1, para cancer.csv el mejor modelo fue Random Forest con 0.97 de desempeño, y para diabetes.csv el mejor modelo fue SVM con un desempeño del 0.86.

IV. Discusión

La utilidad del taller se expande más allá de los datasets trabajados, los resultados obtenidos y la aplicación de formas de organización de código como prefect nos permite tener una mejor forma de entender el código, como codificar soluciones más eficientemente y de una forma más ordenada. Así mismo se logró un aprendizaje sobre los comportamientos de los diferentes algoritmos, como algunos presentan ventajas y otros desventajas al trabajar con distintos tipos de datos. Esto en última instancia nos invita a en nuestras aplicaciones futuras de nuestros aprendizajes del modulo a probar este tipo de algoritmos, y otros que potencialmente aparezcan con el paso de los años para tener la certeza de poder elegir el mejor modelo a utilizar según la circunstancia y con decisiones basadas en respaldos matemáticos, reconociendo las fortalezas y falencias que tiene cada modelo.

Referencias

[1] scikit-learn, "train_test_split — scikit-learn 1.5.2 documentation," scikit-learn.org. [En línea]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[2] scikit-learn, "StandardScaler — scikit-learn documentation," scikit-learn.org. [En línea]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[3] D. M. Powers, .Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation,"Journal of Machine Learning Technologies, vol. 2, no. 1, pp. 37-63, 2011.

[4] Alejandro Gil, Juan David Sánchez, *Taller2_ML1*, GitHub, 2025. [En línea]. Disponible en: https://github.com/AGGE82/Taller-2-ML1