

# Práctica 1:

## Análisis Predictivo Mediante Clasificación



# UNIVERSIDAD DE GRANADA

Laura Lázaro Soraluce  
[lazarosoraluce@correo.ugr.es](mailto:lazarosoraluce@correo.ugr.es)  
Grupo 1  
Inteligencia de Negocio

## **Índice:**

1. [Introducción](#)
2. [Procesado de datos](#)
3. [Resultados obtenidos](#)
4. [Configuración de algoritmos](#)
5. [Ánálisis de resultados](#)
6. [Interpretación de los datos](#)
7. [Contenido adicional](#)
8. [Bibliografía](#)

# Introducción:

En esta práctica utilizamos la plataforma KNIME para abordar tres problemas distintos de clasificación:

- **Predicción de aprobación de créditos:** este conjunto de datos consta de 32.581 instancias divididas en 12 columnas con información para predecir la aprobación de créditos en función de características personales y financieras de los que lo solicitan. Las variables de este problema incluyen atributos demográficos y financieros, como el historial de pagos de créditos (como variable binaria: aprobado/rechazado). Algunas de las categorías que nos dan en este problema tienen valores faltantes: person\_emp\_length tiene 895 y loan\_int\_rate 3.116.
- **Predicción de una segunda cita:** este conjunto de datos proviene de un experimento de citas rápidas en el cual los participantes, luego de una breve interacción, deciden si les gustaría tener una segunda cita con su pareja actual. Entre los datos tenemos varios atributos, como la percepción de atractivo, sinceridad, inteligencia y compatibilidad de intereses, información demográfica y de autopercepción. El problema es de clasificación binaria (¿Se quiere una segunda cita? Sí/no) en el que las decisiones se ven influenciadas por varios factores psicológicos y de comportamiento, por lo que modelos no lineales podrían funcionar bien con este problema. En este caso contamos con: 8378 instancias; 121 características de las cuales 59 son numéricas y 62 simbólicas; 2 posibles clases; y 18,372 valores faltantes en total, distribuidos en 7330 instancias. Como vemos, hay un alto número de valores faltantes por lo que será crucial el preprocesamiento de datos y selección adecuada de las características para abordar este problema.
- **Predicción del tipo de enfermedad eritemato-escamosa:** en el ámbito de la dermatología, nos encontramos con este complejo problema para el que tenemos que considerar características clínicas e históricas patológicas muy similares. A diferencia de los dos problemas anteriores, este es un problema de clasificación de clases múltiples, no de clasificación binaria. El conjunto de datos contiene 366 instancias y 34 atributos (12 clínicos y 22 histopatológicos) de los cuales sólo 1 tiene valores nominales (family history que vale 1 si alguna de las enfermedades estudiadas se ha observado antes en la familia y 0 si no) mientras que el resto son lineales (todos fueron dado un número entre 0 y 3, donde 0 indica que el atributo no estaba presente, 3 indica que lo estaba en la mayor medida posible, y 1, 2 indican valores intermedios). La única categoría que tiene valores faltantes es age. Estos atributos fueron usados para identificar el tipo de enfermedad. Este problema es de clasificación multiclase, pues tenemos seis clases diferentes: psoriasis, dermatitis seborreica, liquen plano, pitiriasis rosada, dermatitis crónica y pitiriasis rubra pilaris. Estas enfermedades tienen algunos patrones únicos que las diferencian, pero también tienen muchos en común, lo que supone la dificultad de este ejercicio.  
He decidido usar Random Forest, SVM, KNN, Gradient Boosting y Redes Neuronales para este problema. Cada uno de estos modelos puede adaptarse para tareas de clasificación con múltiples clases, esenciales para diferenciar entre las seis

enfermedades. Las características de cada enfermedad son similares y pueden estar correlacionadas, lo que requiere algoritmos que puedan captar relaciones no lineales y diferenciarlas de manera efectiva. Aunque algunos modelos, como las redes neuronales, pueden ser menos interpretables, otros como Random Forest y XGBoost permiten evaluar la importancia de las características, lo cual es valioso en aplicaciones médicas. Cada uno de estos algoritmos aporta una ventaja específica al problema (que explicaré en el apartado correspondiente), y probarlos en conjunto podría ofrecer una visión amplia y comparativa de cuál es el más adecuado para esta tarea.

Trabajamos con distintos tipos de datos, clasificación y variables. Hay casos de clasificación binaria y multiclase, datos balanceados y desbalanceados, variables categóricas y numéricas... Van a ser necesarias técnicas de preprocesamiento, por ejemplo, en los 3 problemas nos encontramos con valores faltantes, por lo que puede ser necesaria la imputación de valores faltantes. También podemos necesitar transformar variables categóricas en binarias para que los algoritmos tengan un mayor rendimiento. Haremos análisis exploratorios para identificar las distribuciones y relaciones de los atributos y su influencia en los modelos.

Para esta práctica usaremos KNIME, como venimos haciendo en los seminarios. Esto nos permite comparar múltiples algoritmos mediante sus nodos de clasificación y preprocesamiento, y además proporciona herramientas para la visualización y validación de resultados.

El objetivo principal de esta práctica es evaluar el rendimiento de al menos cinco algoritmos diferentes de clasificación sobre cada uno de los tres problemas, utilizando preprocesamiento para optimizar el desempeño de dichos algoritmos.

# Procesado de datos:

En esta sección, usando el preprocesamiento de los datos, nos aseguraremos de que estén en condiciones óptimas para el análisis y aplicación de los algoritmos de clasificación.

Trataremos valores faltantes, variables categóricas y equilibraremos las clases en los casos en los que sea necesario.

## 1. Predicción de Aprobación de Créditos

En este problema, tenemos tanto datos con atributos numéricos como categóricos, y tengo 895 valores faltantes en person\_emp\_length, y 3116 en loan\_int\_rate que tendré que tratar.

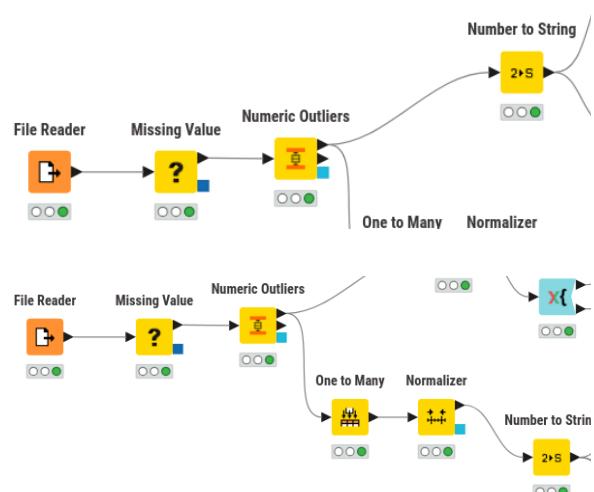
El primer nodo que utilizo es **File Reader** para cargar el archivo de datos .csv.

Conecto este a **Missing Value**, donde relleno los valores faltantes de las columnas correspondientes: person\_emp\_length lo relleno con la mediana (pues no se ve afectada por los valores atípicos que están presentes) mientras que loan\_int\_rate lo relleno con la media (ya que es un atributo que se distribuye más homogéneamente).

Luego utilizo **Numeric Outliers**, donde elimino las filas que tengan valores atípicos, para no tenerlos en cuenta en los algoritmos. Lo hago sólo sobre las columnas person\_age y person\_income, pues en la primera encontré algunos valores sin sentido y la segunda tiene un rango muy grande, por lo que podía haber algún valor atípico. En concreto he escogido k=3 para considerar atípicos los valores que estén más allá de tres veces la desviación típica [1], un método muy usado en distintos campos de la informática (también usamos este valor en la práctica de Visión por Computador). Si lo reducimos, se podrían detectar como outliers puntos que no lo son, ya que se perdería sensibilidad. Si lo aumentamos, se podrían dejar pasar algunos valores atípicos importantes.

Ahora continúo el preprocesamiento de datos de manera específica para cada algoritmo. En todos los algoritmos en los que normalizo, utilizo [0,1], pues las variables tienen rangos definidos y las relaciones entre ellas se mantendrán mejor con este tipo de escalado.

- **Decision Tree y Naives Bayes:** Uso **Number to String** para convertir la variable loan\_status, que originalmente era int, a string y que pueda ser utilizada en el algoritmo.
- **K Nearest Neighbor, MLP y Logistic Regression:** Uso **One to Many** para convertir los distintos valores que aparecen en las columnas person\_home\_ownership, loan\_intent,

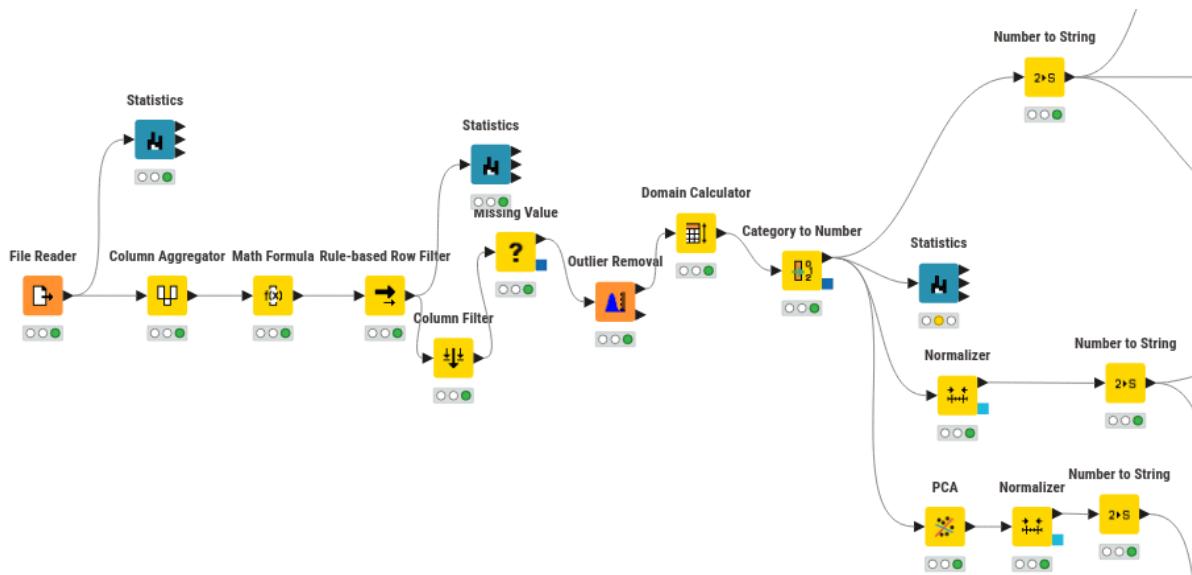


loan\_grade y cb\_person\_default\_on\_file. Elimino estas columnas de la tabla output que me da este nodo, ya que aparecerá una columna nueva de valores binarios por cada valor distinto que había en estas columnas. Así los modelos funcionan mejor, ya que evitamos cualquier interpretación numérica incorrecta y se procesan mejor las variables categóricas sin sugerir relaciones inexistentes entre categorías. El siguiente nodo es **Normalizer** donde normalizo los datos de person\_age, person\_income, person\_emp\_length, loan\_amnt, loan\_percent\_income y cb\_person\_cred\_hist\_length, pues estos tres algoritmos son sensibles a la escala de las variables y así mejoramos su desempeño. En KNN, por ejemplo, los atributos con valores más grandes dominarían la medida de distancia si no normalizamos. Por último uso **Number to String** al igual que en los dos algoritmos anteriores.

Para ver el resultado del preprocesado de datos gráficamente, utilizo el nodo **Statistics**, que muestra un pequeño gráfico por cada categoría, representando el reparto de valores de cada una. Esto me ayuda a entender los datos con los que trabajamos.

## 2. Predicción de una Segunda Cita

Este conjunto de datos es complejo debido a la gran cantidad de valores faltantes, la mezcla de variables numéricas y simbólicas, y las distintas naturalezas de los datos. Hay 18372 valores faltantes en muchas de las columnas y además, hay filas que tienen muchos valores faltantes en sus distintas columnas. En concreto, hay valores faltantes en: age, age\_o, race, race\_o, importance\_same\_race, importance\_same\_religion, field, pref\_o\_attractive, pref\_o\_sincere, pref\_o\_intelligence, pref\_o\_funny, pref\_o\_ambitious, pref\_o\_shared\_interests, attractive\_o, sincere\_o, intellegnce\_o, funny\_o, ambitious\_o, shared\_interests\_o, attractive\_important, sincere\_important, intelligence\_important, funny\_important, ambition\_important, shared\_interests\_important, attractive, intelligence, funny, ambition.



El primer nodo que utilice es **File Reader** para cargar el archivo de datos .csv.

Como hay filas que tienen valores faltantes en muchas de sus columnas, he decidido contar cuántos valores faltantes hay en cada fila y, si el porcentaje de columnas que tienen valores faltantes es mayor que el 30%, elimino la fila entera para no incluirla en los algoritmos. Para lograr esto, uso tres nodos [2]:

- **Column Aggregator:** crea una nueva columna llamada “Missing Value Count” que, como su nombre indica, contiene el número de missing values que tiene cada fila.
- **Math Formula:** calcula el porcentaje de valores faltantes en cada fila basado en la columna “Missing Value Count”. Esto es para ver más claro los valores que queremos eliminar.
- **Rule-based Row Filter:** filtra las filas manteniendo sólo las que tengan la columna “Missing Value Count” a un valor menor que 30 (por ciento).

Finalmente, uso **Column Filter** para eliminar la columna “Missing Value Count”, ya que sólo la creé para los cálculos de filtrado y no quiero que influya en los modelos.

Con el nodo **Missing Value**, pongo todos los valores faltantes de los atributos numéricos de preferencias, importancia y personalidad al valor de la mediana, para evitar sesgos extremos. El resto de atributos con valores faltantes, como lo son la edad, `expected_happy`, `expected_num`, `interests`, `like`, los relleno con la media.. Por último, elimino las filas que tengan valores faltantes en la columna `match`, ya que considero que no es útil incluirlas en los modelos si falta el valor de la variable objetivo.

Uso **Outlier Removal** de la misma manera que en el problema 1 de esta práctica uso **Numeric Outliers**, para eliminar todas las filas que tengan valores atípicos, es decir, que se pasen de tres veces la desviación típica (He usado un nodo diferente en este problema y el problema 1 simplemente por tener variedad y probar nodos distintos en mis flujos). Los campos de edad no los incluyo en este nodo, ya que considero que la edad sí que puede fluctuar bastante y haber gente bastante mayor o bastante menor que la mayoría, y no quería eliminarlos. Sin embargo, si valores como `intelligence_importance` o `attractive_important` resultaran anormalmente altos o bajos, no tendría sentido y deberían ser eliminados.

**Domain Calculator** y **Category to Number** los uso juntos para determinar los posibles valores de cada columna y convertir todas las columnas categóricas en valores numéricos.

Esto lo hago porque los algoritmos funcionan mejor con números, que conocen cómo tratar, que con valores que no saben interpretar lo que significan o la relación que hay entre ellos.

Todo este preprocessamiento es común a todos los algoritmos. Ahora continúo el preprocessamiento de datos de manera específica para cada algoritmo. En todos los algoritmos en los que normalizo los datos, utilizo el método de Z-score en este caso, pues ayuda a lidiar mejor con las variables de comportamiento y variables subjetivas, que tienen distribuciones muy diferentes. Además, sabemos que tanto **MLP** como **KNN** se benefician de trabajar sobre datos centrados y con desviación unitaria. Además, como en este conjunto de datos los datos pueden tener muchos valores diferentes, sabemos que este tipo de normalización es menos sensible a los outliers.

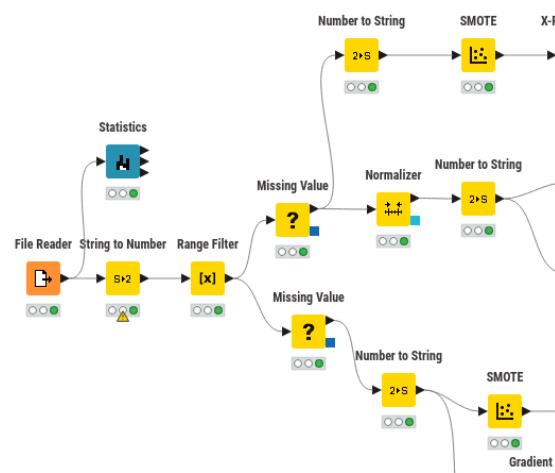
- **Decision Tree y Random Forest:** Uso **Number to String** para convertir la variable match, que originalmente era int, a string y que pueda ser utilizada en los algoritmos.
- **KNN y XGBoost Linear Model:** Uso **Normalizer** donde normalizo todos los datos para que estén en la misma escala y no afecten a estos algoritmos (sensibles a escala), pues había preferencias que iban de 0 a 10 y otras de 0 a 13, por ejemplo. Así conseguimos evitar que el algoritmo interprete los valores en diferentes rangos como inherentemente más importantes. No normalizo age, age\_o, d\_age, expected\_num\_interested\_in\_me ni expected\_num\_matches, ya que la variación de estos valores es significativa. Por último uso **Number to String** igual que en los dos algoritmos anteriores.
- **RProp MLP:** como se trata de un modelo que puede ser costoso en tiempo de ejecución y recursos de cómputo, con **PCA** reduzco la dimensionalidad de los datos, de manera que se conserve el 99'5% de la varianza. Así procesamos la mayor parte de la información sin un costo computacional innecesario, buscando un balance entre resultados del modelo y recursos de computación. Con **Normalizer** y **Number to String** hago exactamente lo mismo que en los algoritmos anteriores.

### 3. Predicción del Tipo de Enfermedad Eritemato-Escamosa

Este problema de clasificación multiclasa presenta un conjunto de datos más específico, con variables clínicas e histopatológicas que son mayormente numéricas (excepto age que es String).

El primer nodo que utilzo es **File Reader** para cargar el archivo de datos .csv.

En este caso, he decidido seguir el orden de preprocessamiento recomendado en las clases de teoría: Reducción de ruido, Valores faltantes y Selección de columnas.



Uso **String to Number** para convertir la columna *age* de String a Int, como el resto de nuestras columnas, pues sabemos que en realidad la edad es un número, aunque inicialmente se haya representado como cadena.

Uso **Range Filter** para filtrar los datos outliers, es decir, limito el rango de las variables correspondientes al rango [0,3], como se indica en la web de dataset. En este caso, vemos que aplicar este nodo no ha quitado ninguna fila, ya que todas estaban dentro del rango correcto. Aún así me parecía prudente aplicarlo por si no hubiese sido este el caso.

Lo siguiente que uso es el nodo de **Missing Value**. Para los algoritmos de **Random Forest**, **SVM** y **KNN**, imputo los valores faltantes de la columna *age* como la mediana. Esto lo hago porque estos algoritmos no manejan valores faltantes de forma nativa. Si dejásemos los valores faltantes, estos algoritmos los considerarían un valor real y esto distorsionaría el análisis. Si usamos la mediana, no introducimos sesgos, pues es menos sensible a valores atípicos que la media. Para los algoritmos **XGBoost** y **Gradient Boosted Trees**, pongo los valores faltantes de *age* al valor -1. Esto es porque los algoritmos de boosting basados en árboles de decisión pueden manejar valores faltantes de forma interna y flexible. En lugar de imputar con una medida estadística, podemos utilizar un valor específico para que el modelo lo interprete como un valor faltante. Usar -1 es una estrategia común para marcar valores como faltantes sin introducir un valor real.

Pasamos ahora al preprocesamiento específico de cada algoritmo:

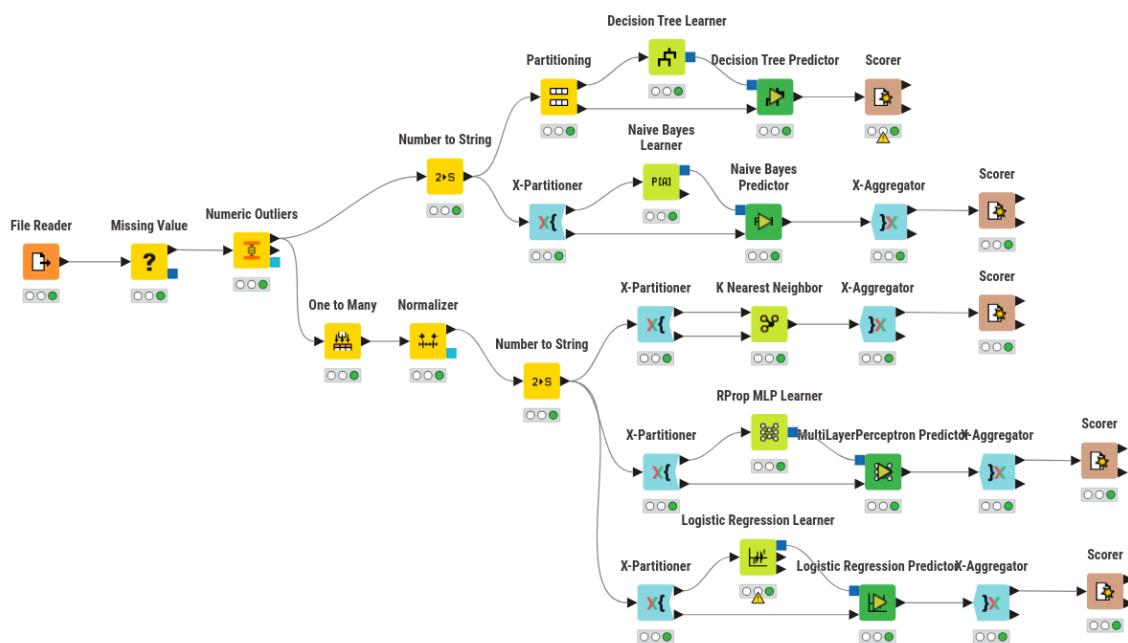
- **Random Forest:** uso **Number to String**, pues como sabemos, necesita que la variable objetivo esté en formato String.
- **SVM y KNN:** uso **Normalizer** con el método Z-Score. Estos dos algoritmos son sensibles a la escala de datos. SVM calcula distancias en el espacio de características mientras que KNN usa distancias entre instancias. Si no normalizamos, los atributos en diferentes escalas pueden distorsionar estos cálculos. Uso también **Number to String** de la misma manera que en los demás algoritmos.
  - Para **KNN** uso además **SMOTE** para abordar el problema de desequilibrio de clases (por ejemplo psoriasis tiene muchas más instancias que pityriasis rubra pilaris), pues es sensible al desequilibrio. Este nodo crea ejemplos sintéticos en las clases minoritarias para equilibrar la distribución de clases y mejorar la capacidad del modelo para generalizar en las clases menos representadas.
  - Para **XGBoost** y **Gradient Boosted Trees**, uso **Number to String**.
    - Para **Gradient Boosted Trees** también uso **SMOTE** por la misma razón que en **KNN**.

# Resultados obtenidos:

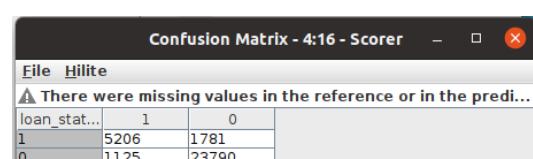
En los tres problemas de esta práctica, utilice el nodo **Scorer** para obtener los valores de las distintas métricas que me permiten saber cómo de bien ha funcionado el modelo para predecir la variable correspondiente. **Scorer** me permite ver la precisión, el recall, el f1-score, la accuracy, los falsos/verdaderos positivos y los falsos/verdaderos negativos.

## 1. Predicción de Aprobación de Créditos

Uso **X-aggregator** y **X-partitioner** (5 folds). Esto es validación cruzada que se utiliza para evaluar el desempeño del modelo usando varias particiones de los datos (en nuestro caso en 5 particiones) en lugar de una sola partición. En cada iteración se utiliza uno de los folds como conjunto de test y los otros 4 como conjunto de entrenamiento. La validación cruzada ayuda a que el modelo sea robusto y generalice bien, ya que evita el sobreajuste a nuestro conjunto de datos particular.



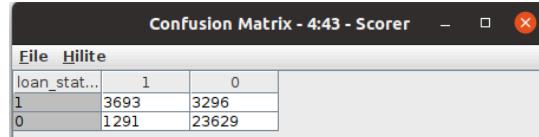
### a. Decision Tree



Métrica	Valor
Accuracy	90.891%
Precision	$0.822*(6987)+0.93*(24915)/(31902)$ = 90.635%
Recall	$0.745*(6987)+0.955*(24915)/(31920)$

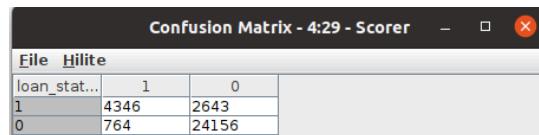
	= 90.849%
F1-Score	$0.782*(6987)+0.942*(24915)/(31920)$ = 90.645%

### b. Naives Bayes



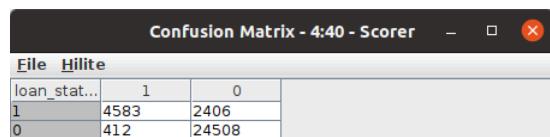
Métrica	Valor
Accuracy	85.625%
Precision	$0.741*(6989)+0.878*(24920)/(31909)$ = 84.799%
Recall	$0.528*(6989)+0.948*(24920)/(31909)$ = 85.601%
F1-Score	$0.617*(6989)+0.912*(24920)/(31909)$ = 84.739%

### c. KNN



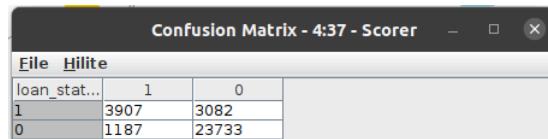
Métrica	Valor
Accuracy	89.285%
Precision	$0.849*(6989)+0.901*(24920)/(31909)$ = 88.961%
Recall	$0.622*(6989)+0.969*(24920)/(31909)$ = 89.3%
F1-Score	$0.718*(6989)+0.934*(24920)/(31909)$ = 88.669%

### d. MLP



Métrica	Valor
Accuracy	91.169%
Precision	$0.918*(6989)+0.911*(24920)/(31909) = 91.253\%$
Recall	$0.656*(6989)+0.983*(24920)/(31909) = 91.138\%$
F1-Score	$0.765*(6989)+0.946*(24920)/(31909) = 90.636\%$

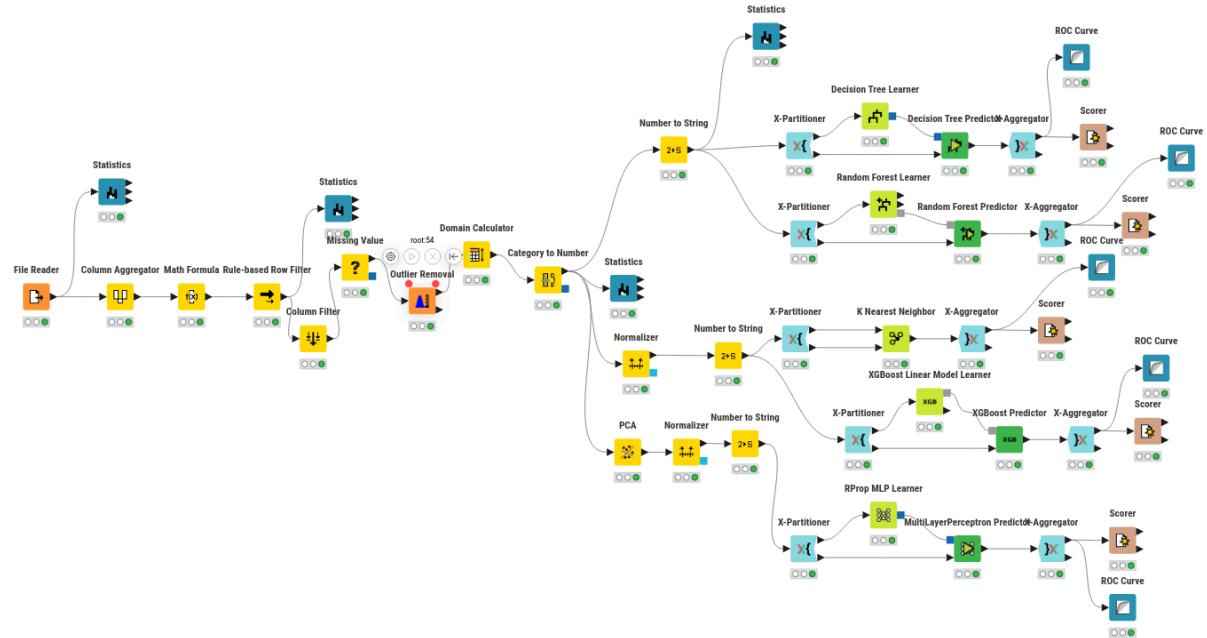
#### e. Logistic Regression



Métrica	Valor
Accuracy	86.621%
Precision	$0.767*(6989)+0.885*(24920)/(31909) = 85.915\%$
Recall	$0.559*(6989)+0.952*(24920)/(31909) = 86.592\%$
F1-Score	$0.647*(6989)+0.917*(24920)/(31909) = 85.786\%$

## 2. Predicción de una segunda cita

Como en todos los algoritmos de esta práctica, utilizo **X-partitioner** (5 folds) y **X-aggregator** para hacer validación cruzada con los datos.



### a. Decision Tree

Confusion Matrix - 3:13 - Scorer		
	File	Hilite
match \ P...	0	1
0	4777	477
1	555	424

Métrica	Valor
Accuracy	83.443%
Precision	$0.471*(979)+0.896*(5254)/(6233) = 82.925\%$
Recall	$0.433*(979)+0.909*(5254)/(6233) = 83.424\%$
F1-Score	$0.451*(979)+0.903*(5254)/(6233) = 83.201\%$

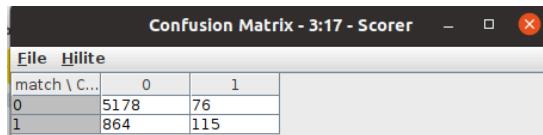
### b. Random Forest

Confusion Matrix - 3:37 - Scorer		
	File	Hilite
match \ P...	0	1
0	5196	58
1	700	279

Métrica	Valor
Accuracy	87.839%

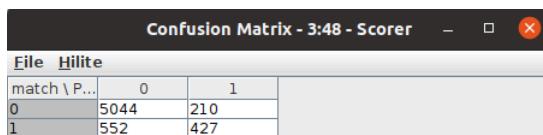
Precision	$0.828*(979)+0.881*(5254)/(6233) = 87.268\%$
Recall	$0.285*(979)+0.989*(5254)/(6233) = 87.842\%$
F1-Score	$0.424*(979)+0.932*(5254)/(6233) = 85.221\%$

### c. KNN



Métrica	Valor
Accuracy	84.919%
Precision	$0.602*(979)+0.857*(5254)/(6233) = 81.695\%$
Recall	$0.117*(979)+0.986*(5254)/(6233) = 84.951\%$
F1-Score	$0.197*(979)+0.917*(5254)/(6233) = 80.391\%$

### d. XGBoost



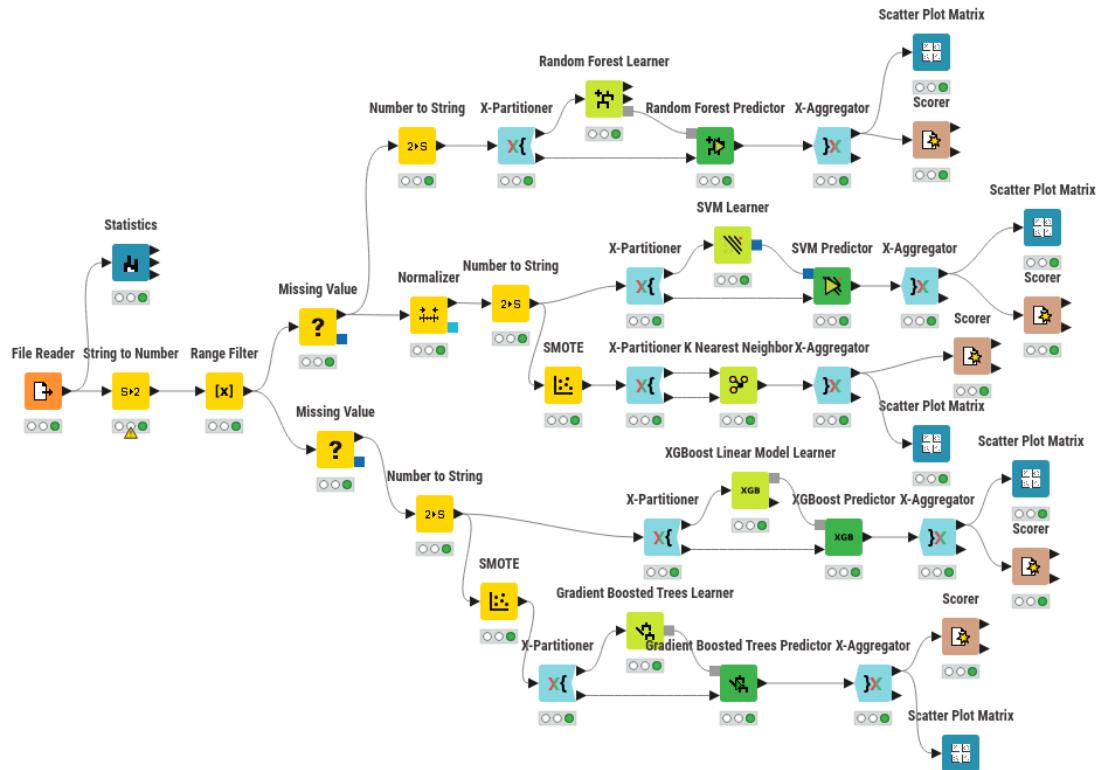
Métrica	Valor
Accuracy	87.775%
Precision	$0.67*(979)+0.901*(5254)/(6233) = 86.472\%$
Recall	$0.436*(979)+0.96*(5254)/(6233) = 87.77\%$
F1-Score	$0.528*(979)+0.93*(5254)/(6233) = 86.686\%$

### e. MLP

Confusion Matrix - 3:24 - Scorer		
File	Hilite	
match \ P...	0	1
0	4871	383
1	542	437

Métrica	Valor
Accuracy	85.16%
Precision	$0.533*(979)+0.9*(5254)/(6233) = 84.236\%$
Recall	$0.446*(979)+0.927*(5254)/(6233) = 85.459\%$
F1-Score	$0.486*(979)+0.913*(5254)/(6233) = 84.593\%$

### 3. Predicción del tipo de enfermedad eritemato-escamosa



Como en todos los algoritmos de esta práctica, utilizo **X-partitioner** (5 folds) y **X-aggregator** para hacer validación cruzada con los datos.

### a. Random Forest

Confusion Matrix - 4:30 - Scorer							
File	Hilite	2	1	3	5	4	6
class \ Pr...		57	1	0	0	2	1
2		0	112	0	0	0	0
1		0	0	72	0	0	0
3		0	0	0	52	0	0
5		0	0	0	0	45	0
4		4	0	0	0	0	0
6		0	0	0	0	0	20

Métrica	Valor
Accuracy	97.814%
Precision	$0.934*(61)+0.991*(112)+72+52+0.957*(49)+0.952*(20)/366 = 97.787\%$
Recall	$0.934*(61)+112+72+52+0.918*(49)+20/366 = 97.802\%$
F1-Score	$0.934*(61)+0.996*(112)+72+52+0.938*(49)+0.976*(20)/366 = 97.816\%$

### b. SVM

Confusion Matrix - 4:31 - Scorer							
File	Hilite	2	1	3	5	4	6
class \ Pr...		54	1	0	0	6	0
2		1	111	0	0	0	0
1		0	0	72	0	0	0
3		0	0	0	51	1	0
5		3	0	0	0	46	0
4		1	0	0	0	0	19

Métrica	Valor
Accuracy	96.448%
Precision	$0.915*(61)+0.991*(112)+72+52+0.978*(49)+20/366 = 98.013\%$
Recall	$0.885*(61)+0.991*(112)+72+0.981*(52)+0.939*(49)+0.95*(20)/366 = 96.448\%$
F1-Score	$0.9*(61)+0.991*(112)+72+0.99*(52)+0.902*(49)+0.74*(20)/366 = 95.183\%$

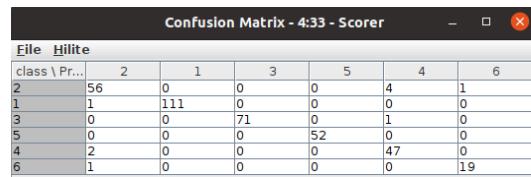
### c. KNN

Confusion Matrix - 4:32 - Scorer							
File	Hilite	2	1	3	5	4	6
class \ Cl...		93	0	0	3	16	0
2		8	93	0	2	7	2
1		0	0	112	0	0	0
3		4	0	0	106	2	0
5		5	0	0	2	105	0
4		0	0	0	0	0	112
6		0	0	0	0	0	0

Métrica	Valor
Accuracy	92.411%
Precision	$0.845*(112)+112+112+0.938*112+0.808*(112)+112/672 = 98.013\%$

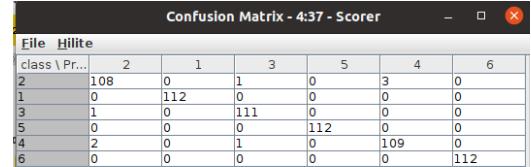
Recall	$0.83*(112)+0.83*112+112+0.946*112+0.938*(112)+112 / 672 = 92.4\%$
F1-Score	$0.97*(112)+112+112+0.988*112+0.955*(112)+0.966*112 / 672 = 97.983\%$

#### d. XGBoost



Métrica	Valor
Accuracy	97.268%
Precision	$0.933*61+112+72+52+0.904*49+0.95*20 / 366 = 97.325\%$
Recall	$0.918*61+0.991*112+0.986*72+52+0.959*49+0.95*20 / 366 = 96.906\%$
F1-Score	$0.926*61+0.996*112+0.993*72+52+0.931*49+0.95*20 / 366 = 97.31\%$

#### e. Gradient Boosted Trees



Métrica	Valor
Accuracy	92.411%
Precision	$0.973*112+112+0.982*112+112+0.973*112+112 / 672 = 98.8\%$
Recall	$0.964*112+112+0.991*112+112+0.973*112+112 / 672 = 98.8\%$
F1-Score	$0.969*112+112+0.987*112+112+0.973*112+112 / 672 = 98.817\%$

# Configuración de algoritmos:

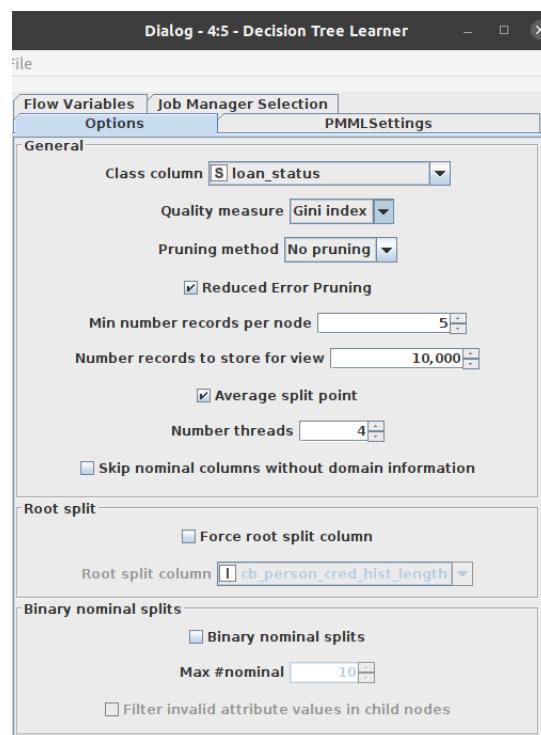
## 1. Predicción de Aprobación de Créditos

### a. Decision Tree

Los árboles de decisión funcionan bien en conjuntos de datos que combinan atributos categóricos y numéricos, como es el caso aquí, con variables como la edad del cliente y el tipo de propiedad de vivienda. En problemas de crédito, donde es importante entender qué factores llevan a una decisión, los árboles de decisión destacan por su interpretabilidad [3]: un banco puede observar directamente qué características (ingresos, empleo, porcentaje de ingresos destinado al crédito) están influyendo en la aprobación o rechazo, lo que es valioso para la toma de decisiones.

Para la configuración específica de este algoritmo, me he basado en la de los seminarios, y he observado que se obtenían buenos resultados. Min number records per node establece la cantidad mínima de instancias que debe tener un nodo del árbol para que pueda continuar dividiéndose en subnodos. Un valor más bajo supondría un árbol más ramificado, que se ajusta más a los datos de entrenamiento, pudiendo producir un sobreajuste. Un valor más alto limita demasiado la profundidad del árbol, pudiendo perder algunos detalles importantes. Estableciéndolo a 5, consigo un buen equilibrio. A **Decision Tree Learner**

le paso el 80% de los datos de entrenamiento (los datos con los que aprende el modelo) mientras que a **Decision Tree Predictor** le paso el 20% de test, que es sobre los que tiene que predecir qué va a pasar. Además, destaco que en **Decision Tree Learner** he desactivado la opción de *Skip nominal columns without domain information*, ya que si no lo hacía, acababa con valores de predicción nulos.



### b. Naives Bayes

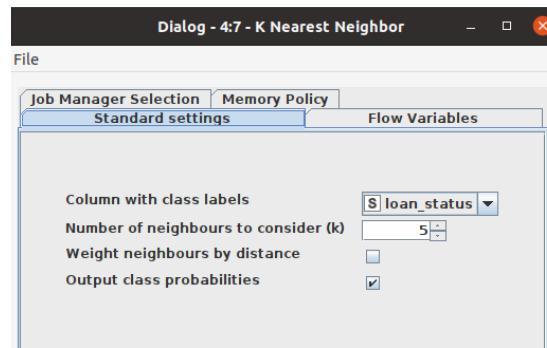
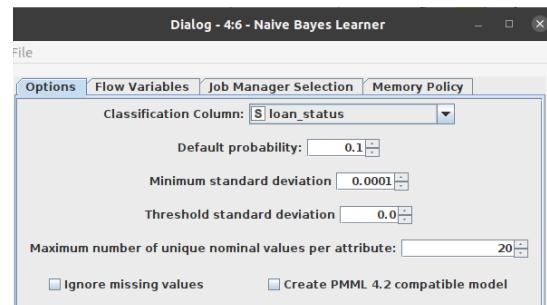
Es un modelo rápido y efectivo para grandes volúmenes de datos, como este problema que tiene 32 mil instancias. Este algoritmo asume independencia entre los atributos, lo cual no es cierto en este problema, pero aún así obtenemos buenos resultados cuando las clases de predicción están bien

separadas. **Naive Bayes** permite capturar patrones básicos de aprobación y rechazo sin complicarse con interdependencias detalladas entre variables [4].

Para la configuración del algoritmo, he usado una probabilidad por defecto de 0.1. Este valor se usa porque podría darse el caso de que haya combinaciones específicas que no se dan en los datos de entrenamiento, lo cual supondría una probabilidad de 0 para esas combinaciones. Al poner una probabilidad mínima 0.1, se suavizan los valores para que el modelo no estime una probabilidad de 0. Esto es útil ya que en este problema hay muchas combinaciones posibles distintas (ya que hay atributos dobles que pueden tomar cualquier valor en un gran rango), y así manejamos mejor casos raros o datos escasos. He decidido subirlo a 0.1 para que el modelo disminuya la influencia de datos específicos, por la razón que ya he explicado, para que sea más robusto cuando hay muchas combinaciones de atributos posibles y evitar que se vuelva demasiado específico. Uso **Naives Bayes Learner** para aprender de los datos de entrenamiento y **Naives Bayes Predictor** para predecir qué pasará con los datos de test.

### c. KNN

Es un modelo basado en similitud [5] que puede ser útil en problemas de crédito, donde los clientes con características similares tienden a recibir decisiones de aprobación similares. Este enfoque es intuitivo en este contexto, ya que los patrones de similitud entre clientes permiten estimar la probabilidad de aprobación de un crédito según la experiencia con clientes de perfiles similares. Aunque **KNN** puede ser sensible a valores faltantes, estos han sido gestionados en el preprocesamiento de los datos.

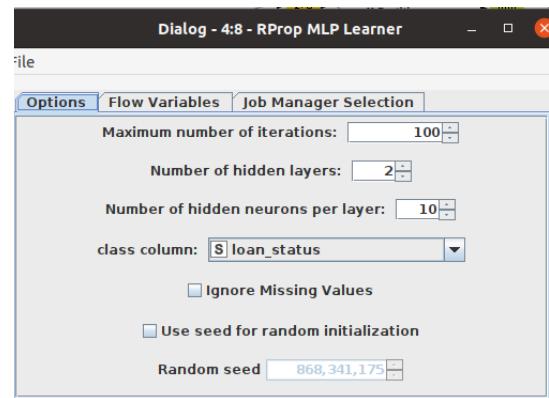


He configurado k=5, de manera que el modelo considera los 5 vecinos más cercanos para clasificar cada instancia. Esta elección de k permite captar patrones de similitud entre clientes sin ser demasiado específico, lo cual reduce el riesgo de sobreajuste. Como no se utiliza ponderación de vecinos por distancia, todos los vecinos contribuyen por igual en la decisión final. Esto es adecuado en problemas de crédito, donde buscamos patrones generales de similitud en vez de favorecer a los vecinos más cercanos.

#### d. MLP

Las redes neuronales pueden capturar relaciones complejas y no lineales entre los atributos del cliente y la aprobación de crédito [6]. En este conjunto de datos, hay dependencias complejas entre variables como el ingreso y tiempo de años trabajados, o el porcentaje del sueldo destinado a pagar el crédito. **MLP** es una elección adecuada para explorar patrones no lineales que otros algoritmos podrían no capturar.

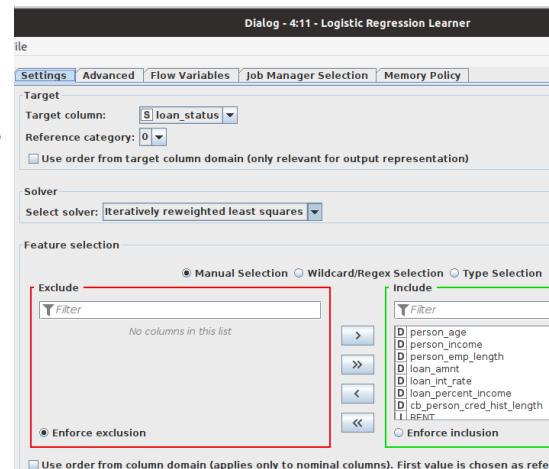
Para la configuración del algoritmo, he usado 100 iteraciones. Esto es la cantidad de veces que el modelo pasa por todo el conjunto de entrenamiento para ajustar el peso. Para elegir este valor, además de usar lo que sé de teoría, lo fui alterando para ver cómo afectaba al rendimiento del modelo, hasta que encontré el rango que mejor funcionaba. Si lo bajase mucho, tendríamos un modelo que no aprende suficientemente bien de los datos, porque no lo hemos entrenado lo suficiente. Si lo subiese demasiado, podríamos tener el efecto contrario, un sobreajuste, y perderíamos generalización, algo que precisamente no queremos para este problema. Combino **RProp MLP Leaner** para los datos de entrenamiento con **MultiLayerPerceptron Predictor** para los de test.



#### e. Logistic Regression

Es un modelo estadístico interpretativo y eficiente para problemas de clasificación binaria, como la aprobación o rechazo de un crédito. Este algoritmo estima probabilidades, lo cual es útil para este problema ya que permite obtener una "probabilidad de riesgo" asociada a cada cliente. También es muy interpretable, ayudando a visualizar cómo influyen cada uno de los atributos en la decisión de crédito. Además, su simplicidad y rapidez la hacen ideal para conjuntos de datos grandes como este.

Para la configuración del algoritmo [7], incluyo todas las columnas en la selección de atributos, no hago ninguna configuración especial más que la que viene por defecto. Combino **Logistic Regression Learner** para los datos de



entrenamiento con **Logistic Regression Predictor** para los de test. Utilizo el método Iteratively Reweighted Least Squares, ya que converge de forma más precisa que Stochastic Average Gradient en conjunto de datos no muy grandes como es el nuestro. Además, ajusta mejor los pesos en cada iteración, cuando los datos contienen múltiples factores influyentes, como es nuestro caso. Así encontramos una solución más precisa en menos iteraciones.

## 2. Predicción de una segunda cita

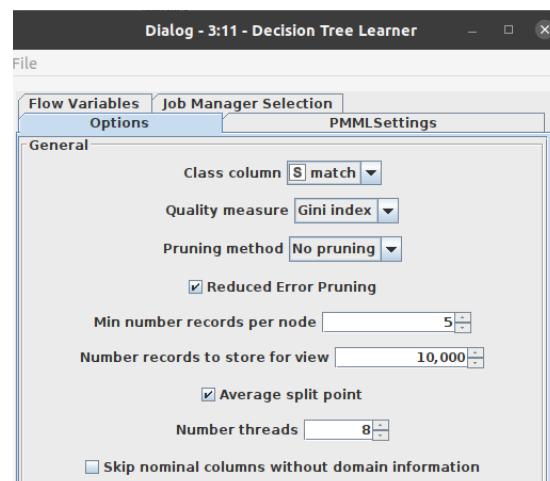
### a. Decision Tree

Los árboles de decisión son intuitivos y permiten capturar relaciones no lineales entre las variables. En nuestro problema, un árbol de decisión puede identificar umbrales clave, como la edad o la importancia de ciertas características en una pareja, para hacer predicciones. Además, funcionan bien con datos tanto categóricos como numéricos, lo cual es útil para este conjunto de datos que incluye atributos diversos (como "edad" o "atractivo").

Lo negativo de este algoritmo es que es propenso al sobreajuste y sensible a pequeñas variaciones de los datos, pero he intentado escoger los parámetros lo mejor posible para evitar esto.

Elijo el índice Gini, que es útil ya que las variables en el conjunto de datos incluyen atributos que miden preferencias subjetivas y diferencias de interés (como atractivo, sinceridad, inteligencia, etc.) podrían influir de manera binaria en la decisión de tener una segunda cita. Gini crea divisiones claras entre las clases en base a estos atributos. Además, a diferencia del criterio de entropía, el índice Gini es

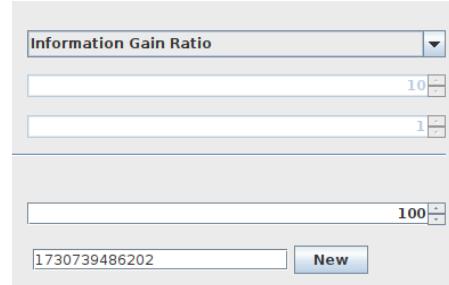
computacionalmente más eficiente, lo que es beneficioso para este conjunto de datos, que cuenta con muchas variables. Uso Min Number Records per Node = 5, para limitar la división del árbol para evitar un ajuste excesivo a instancias específicas, especialmente en este conjunto de datos, que puede incluir datos ruidosos y preferencias muy variadas. Si el valor fuera menor se crearía un árbol mucho más profundo y detallado, lo que podríamos tener divisiones muy específicas para ciertas combinaciones de atributos que pueden no ser representativas. Si el valor fuera mayor se reduciría la profundidad del árbol, lo que podría llevar a perder detalles importantes en la clasificación, sobre todo en atributos clave que varían mucho entre individuos.



### b. Random Forest

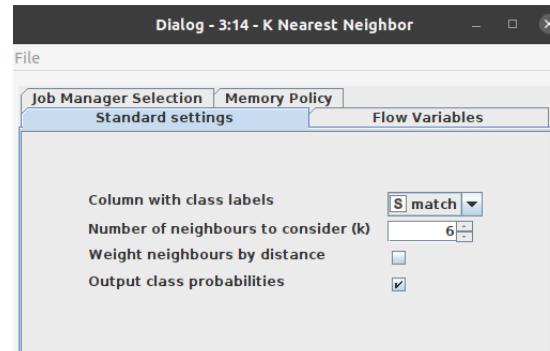
Son una extensión del algoritmo anterior, los árboles de decisión. Se combinan varios árboles para mejorar la precisión y reducir el sobreajuste. En nuestro problema, **Random Forest** puede capturar tambien los patrones complejos al considerar variaciones en los subárboles y reducir el riesgo de sobreajuste que teníamos con **Decision Tree**. Es adecuado para problemas en los que hay muchas variables que pueden ser importantes y pueden interactuar de manera compleja, como es nuestro caso [8].

Utilizo Information Gain Ratio, que es útil cuando tenemos datos con desequilibrios de clase, ya que algunas características podrían tener un impacto desigual en la probabilidad de tener una segunda cita. Utilizo 100 árboles para encontrar el balance entre precisión y tiempo de ejecución. Si pusiese más árboles, el modelo podría ser más robusto, pero tomaría mucho más tiempo para entrenarse.



### c. KNN

Es un modelo basado en similitud que puede ser útil en nuestro problema, ya que permite identificar personas con preferencias o características similares que probablemente tomarían decisiones similares sobre una segunda cita. **KNN** es fácil de interpretar en este contexto, donde los patrones de similitud entre individuos pueden influir en los resultados.



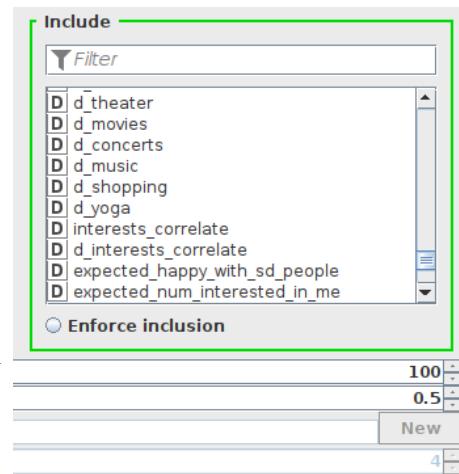
He configurado k=6, lo que permite al modelo considerar los 6 vecinos más cercanos al hacer la predicción. Esta elección de k ayuda a captar un equilibrio entre captar patrones generales sin ser excesivamente sensible a valores individuales o ruidosos en los datos. Al no aplicar ponderación de vecinos por distancia, todos los vecinos se consideran por igual, lo cual es adecuado en este caso donde el tamaño de muestra es moderado y no se espera que los vecinos más cercanos dominen en exceso la predicción.

### d. XGBoost Linear Model

Es un algoritmo potente de boosting basado en árboles. Optimiza la precisión al combinar modelos débiles en una serie de rondas de aprendizaje. Es ideal

para problemas complejos como la predicción de citas, donde diferentes características podrían influir de manera acumulativa. Maneja bien tanto datos categóricos como numéricos y es especialmente bueno cuando hay relaciones no lineales y múltiples interacciones.

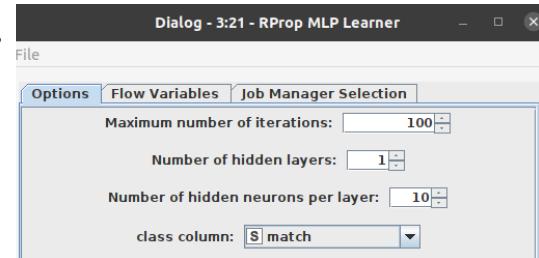
He configurado Boosting Rounds a 100. Este valor determina el número de rondas de boosting, donde cada ronda ajusta el modelo para reducir los errores del anterior. He buscado tener buen balance entre precisión y sobreajuste, pues si aumentase demasiado el número de rondas, podríamos tener sobreajuste, es decir, que el modelo sería muy bueno con los datos de entrenamiento, pero le costaría generalizar a datos de test.



#### e. RProp MLP

Las redes neuronales son buenas para capturar patrones no lineales complejos en los datos. En nuestro problema, donde las interacciones entre variables son complejas, la **MLP** es ideal. **RProp** ajusta los pesos en función de la dirección del gradiente, en lugar de la magnitud, lo que ayuda a estabilizar el aprendizaje.

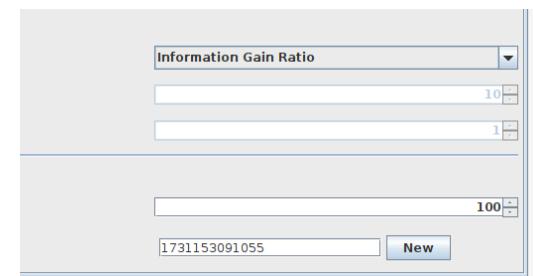
He configurado max iterations a 100, para evitar que la red sobreajuste y para mantener un tiempo de entrenamiento razonable, pues si aumentaba mucho este valor, el entrenamiento del modelo pasaba a ser demasiado lento.



### 3. Predicción de enfermedades eritemato-escamosa

#### a. Random Forest

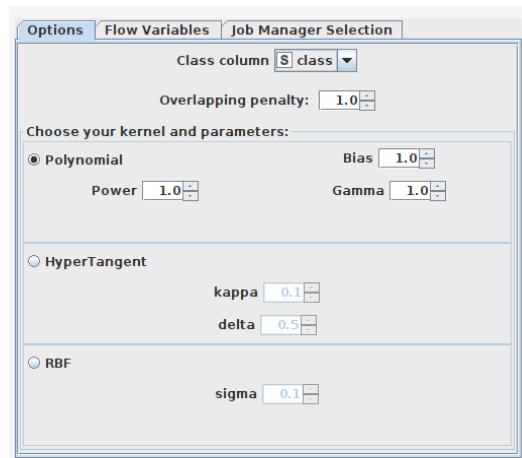
Es adecuado para problemas donde existen múltiples clases y posibles relaciones no lineales entre características. Además, son capaces de manejar datos con ruido y características correlacionadas, lo cual es útil aquí, dado que las enfermedades tienen patrones superpuestos. Además, este modelo puede realizar selección de características



al mostrar qué atributos son más importantes para la clasificación, lo cual es valioso en este contexto donde hay 34 características.<sup>[OBJ]</sup>

### b. SVM

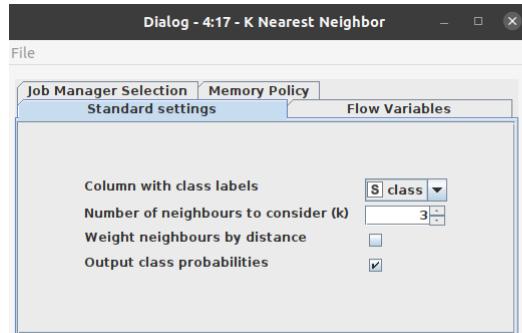
Es eficaz en problemas de clasificación multiclas cuando las clases son difíciles de separar y puede capturar relaciones complejas en los datos [9]. Esto es crucial en el contexto de dermatología, donde las características clínicas y patológicas pueden ser similares entre clases, pero separadas por relaciones complejas. **SVM** es capaz de encontrar un límite de decisión que maximiza la distancia entre clases, lo cual puede ayudar en la clasificación precisa de estas enfermedades con patrones similares. He configurado **SVM** con un kernel polinomial y gamma=1. Este kernel permite captar relaciones no lineales entre los síntomas dermatológicos, permitiendo al modelo crear límites de decisión más flexibles y precisos para separar enfermedades que comparten muchas características clínicas.



La configuración la he dejado como la que venía por defecto, pues vemos que funciona bien interpretando y prediciendo los datos de este problema.

### c. KNN

Es un algoritmo basado en la distancia que puede ser útil para problemas donde los datos presentan agrupaciones naturales. Este método es intuitivo para problemas médicos porque clasifica en función de las instancias más cercanas, lo que es comparable a un diagnóstico médico basado en la similitud con casos anteriores. **KNN** es especialmente útil aquí porque, al trabajar con una escala limitada (0-3), es probable que los valores de las características generen agrupaciones distintivas para cada enfermedad.

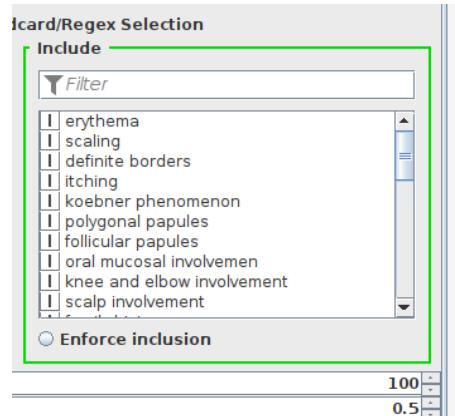


He configurado **KNN** con k=3 y sin ponderación de vecinos por distancia. Esta configuración permite al modelo considerar cada vecino con el mismo peso, lo cual es útil para captar patrones generales entre enfermedades sin dar prioridad a valores específicos en el espacio de características.

#### d. XGBoost

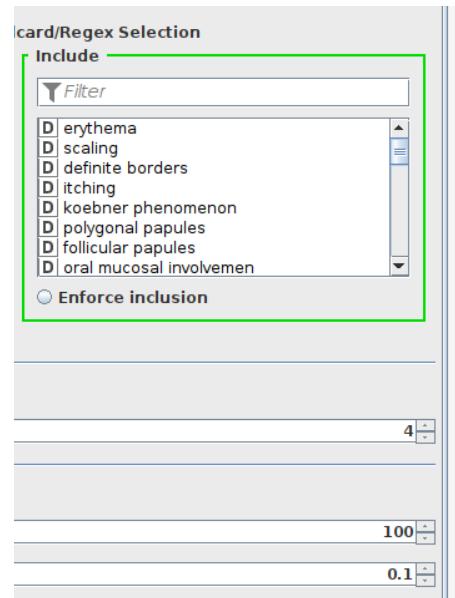
Es poderoso en problemas de clasificación multiclase complejos y suelen superar a otros modelos en precisión. En este problema, donde existen muchas características, los modelos de boosting pueden ser útiles porque construyen secuencialmente árboles de decisión que corrigen los errores de los anteriores.

**XGBoost** también permite manejar valores faltantes, como en la columna de edad en este caso, de manera eficaz al generar automáticamente particiones óptimas para estos valores. Esto, combinado con su capacidad para manejar relaciones no lineales y seleccionar características relevantes, hace que sea adecuado para el problema.



#### e. Gradient Boosted Trees

Es bueno para este problema debido a su capacidad para manejar tareas de clasificación complejas y su habilidad para combinar el poder de múltiples árboles débiles y crear un modelo final robusto y preciso. En el caso del diagnóstico de enfermedades eritemato-escamosas, las características clínicas e histopatológicas son diversas y pueden interactuar de maneras complejas. Esto permite que GBT se ajuste a patrones complejos sin perder generalización, algo crucial cuando las enfermedades comparten tantas características similares pero tienen pequeñas variaciones distintivas. Otra ventaja clave es que GBT maneja bien tanto características numéricas como categóricas y tiene una alta resistencia al ruido, lo que resulta útil considerando que los datos incluyen múltiples variables clínicas e histopatológicas que podrían contener algunas variabilidades no informativas.



## Análisis de resultados:

En los tres problemas de esta práctica, utilice el nodo **ROC Curve** para obtener una representación visual de cómo bien el modelo predice la variable correspondiente, que lo interpreto a partir del Área Bajo la Curva del nodo **ROC Curve**.

### 1. Predicción de Aprobación de Créditos

Aquí tenemos la tabla que muestra los valores de diferentes métricas para los cinco algoritmos que hemos utilizado para este problema de aprobación de créditos, prediciendo `loan_status`.

Para este problema, considero que tanto la precisión (minimizar falsos positivos) como el recall (minimizar falsos negativos) son igual de importantes, ya que es negativo para el banco darle un crédito a alguien a quien no planeaban dárselo según sus datos (false positive), pero es igual de negativo no darle un crédito a un cliente que cumplía los requisitos (false negative), pues puede suponer su cambio a otra entidad bancaria y por lo tanto traducirse en una pérdida de clientes.

Algoritmo	Accuracy	Precision	Recall	F1-Score
Decision Tree	90.891%	90.635%	90.849%	90.645%
Naives Bayes	85.625%	84.799%	85.601%	84.739%
KNN	89.285%	88.961%	89.3%	88.669%
RProp MLP	91.169%	91.253%	91.138%	90.636%
Logistic Regression	86.621%	85.915%	86.592%	85.786%

Vemos que **Decision Tree** obtiene un valor alto en todas las métricas, lo cual indica que el modelo es capaz de capturar bien tanto los casos positivos como los negativos, logrando un buen balance entre precisión y recall, como buscábamos. Su habilidad para dividir los datos en función de los atributos relevantes, como pueden serlo `person_home_ownership` o `loan_percent_income`, le permite realizar clasificaciones efectivas en este problema, donde algunos atributos tienen un peso más alto en la decisión de aprobación del crédito.

**Naive Bayes** presenta una accuracy y F1-Score algo más bajas en comparación con otros algoritmos, lo que refleja una menor capacidad para distinguir entre las clases. Este algoritmo se ve limitado debido a su suposición de independencia entre los atributos, lo cual no se adapta muy bien a nuestro problema, donde hay interacciones entre las variables como el ingreso y el porcentaje de ingresos destinado al crédito. Aún así, sigue siendo útil en escenarios donde el modelo debe ser simple y rápido.

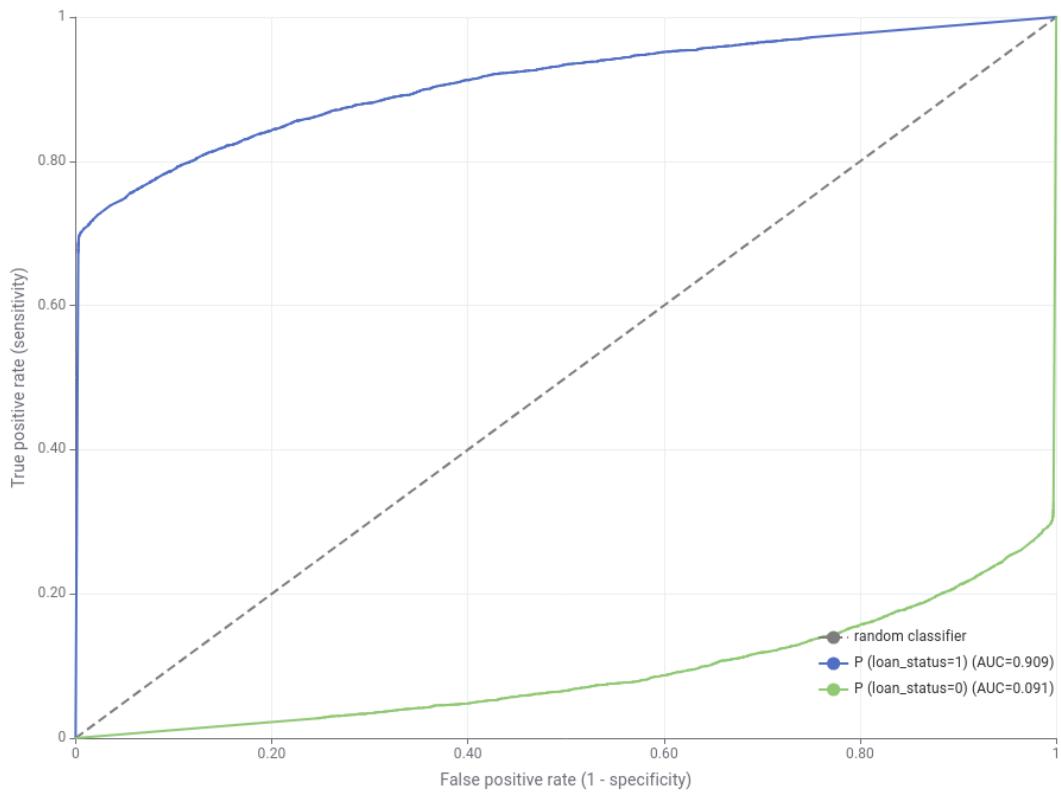
**K-Nearest Neighbors (KNN)** obtiene buenos valores de precisión, recall y F1-Score, quedando por encima de Naive Bayes y de Logistic Regression, aunque un poco por debajo de **Decision Tree** y **RProp MLP**. KNN puede verse afectado por el ruido y la presencia de datos similares en distintas clases, sobre todo en atributos categóricos. Además, en este problema, variables como loan\_amnt, loan\_percent\_income y loan\_int\_rate pueden presentar correlaciones que no se modelan del todo bien mediante las distancias euclidianas que KNN utiliza para calcular la similitud.

**RProp MLP** es el modelo con el mejor rendimiento en todas las métricas, destacando especialmente en precisión y recall. Como red neuronal es capaz de capturar patrones complejos y relaciones no lineales en los datos, lo cual es ventajoso en este problema donde los atributos como income, loan\_percent\_income, y otras características podrían estar interrelacionados de forma no lineal y ser difícil de detectar con otros modelos.

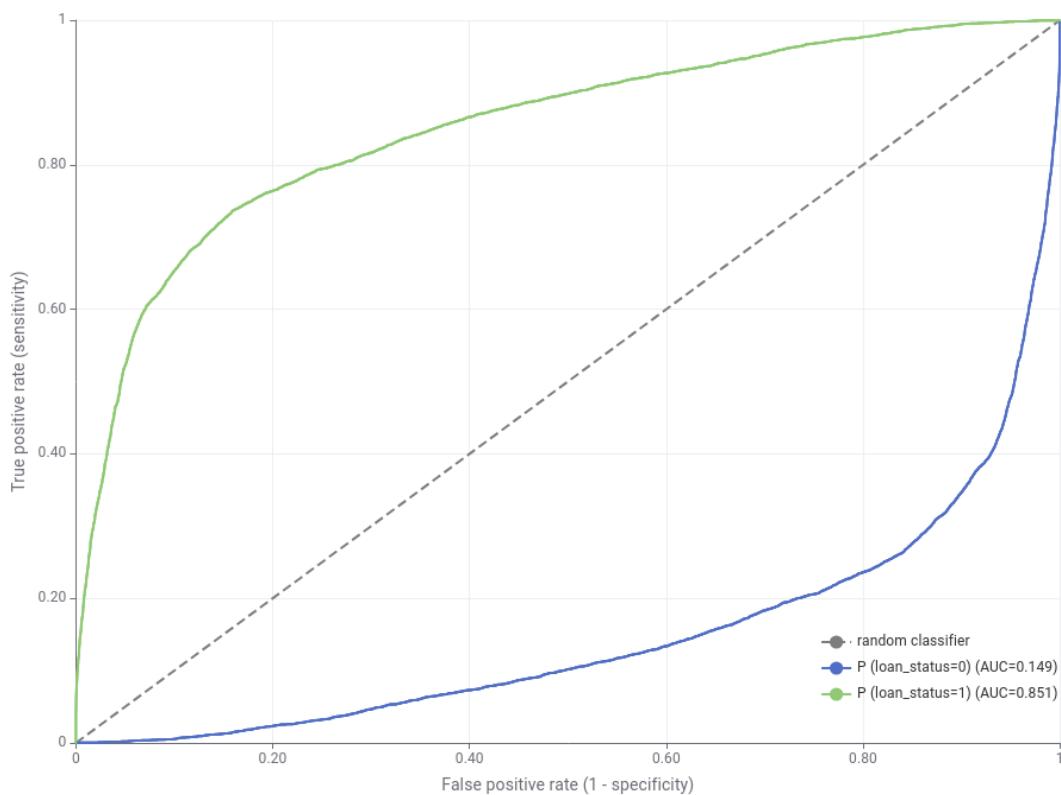
**Logistic Regression** muestra métricas de accuracy y F1-Score menores en comparación con los modelos de árbol de decisión y **MLP**. Aunque Logistic Regression es rápida y fácil de interpretar, su rendimiento inferior puede deberse a que no es tan eficaz para capturar relaciones no lineales en los datos. En este problema, donde se observa una interacción compleja entre varios atributos, **Logistic Regression** se ve limitada en su capacidad para modelar patrones complejos, lo cual explica su desempeño inferior.

Por otro lado, utilice curvas ROC para estudiar los resultados y sacar conclusiones. En las curvas ROC, cuanto más alto sea el AUC de la clase positiva (loan\_status=1) y más bajo sea el de loan\_status=0, nos indica que nuestro modelo está siendo más efectivo al detectar los verdaderos positivos y descartar los negativos, que es justo lo que queremos en un problema de clasificación binaria como este.

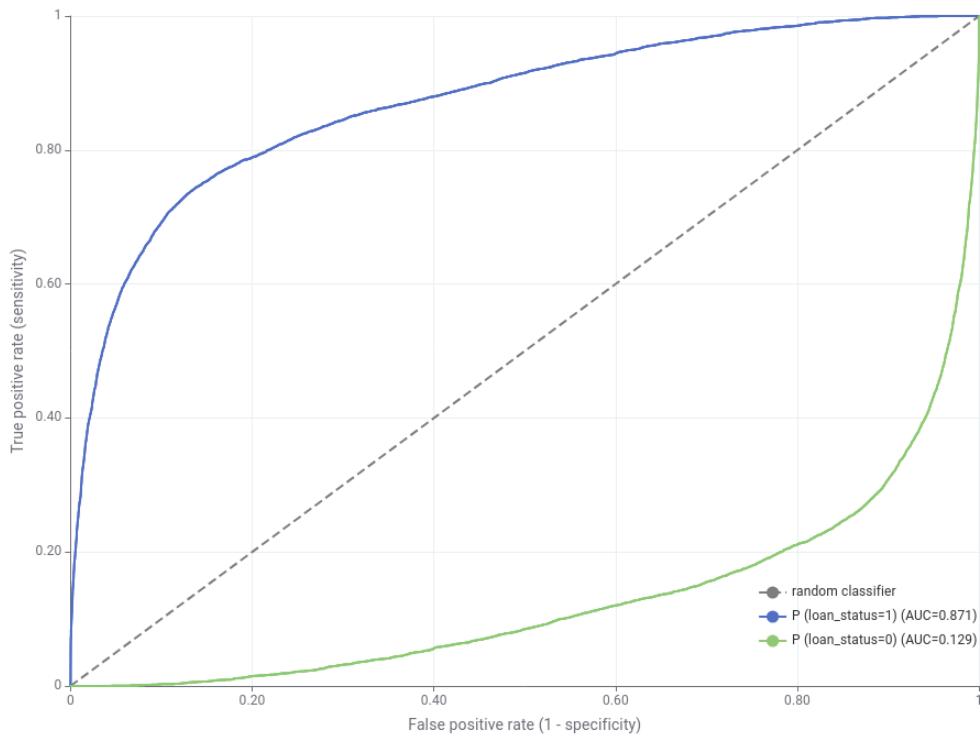
### CURVA ROC de DECISION TREE:



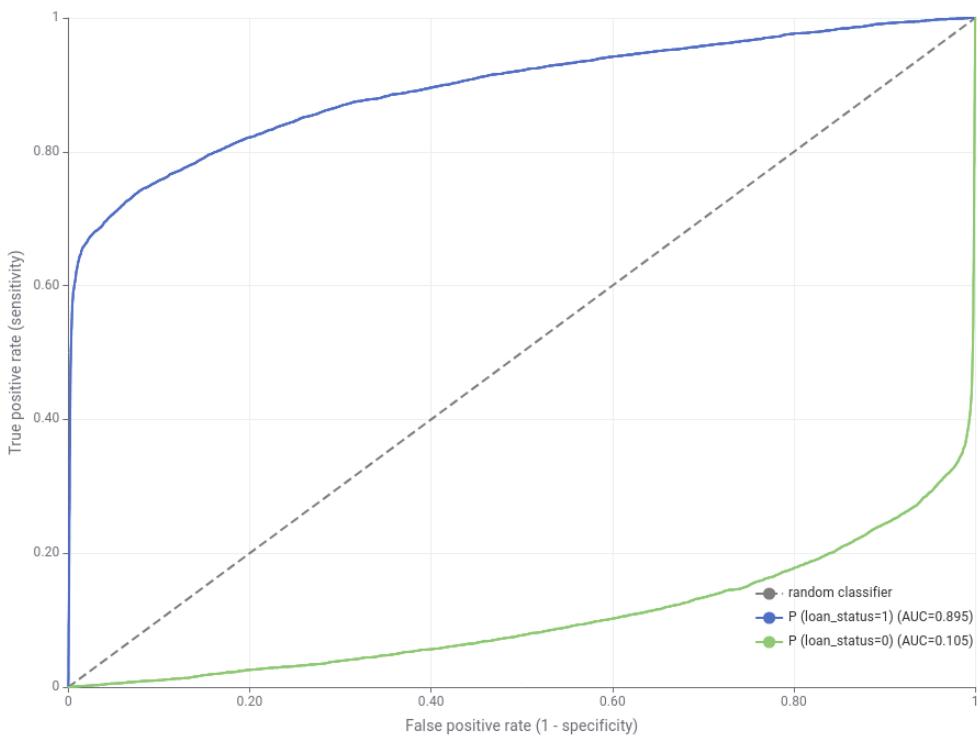
### CURVA ROC de NAIVES BAYES:



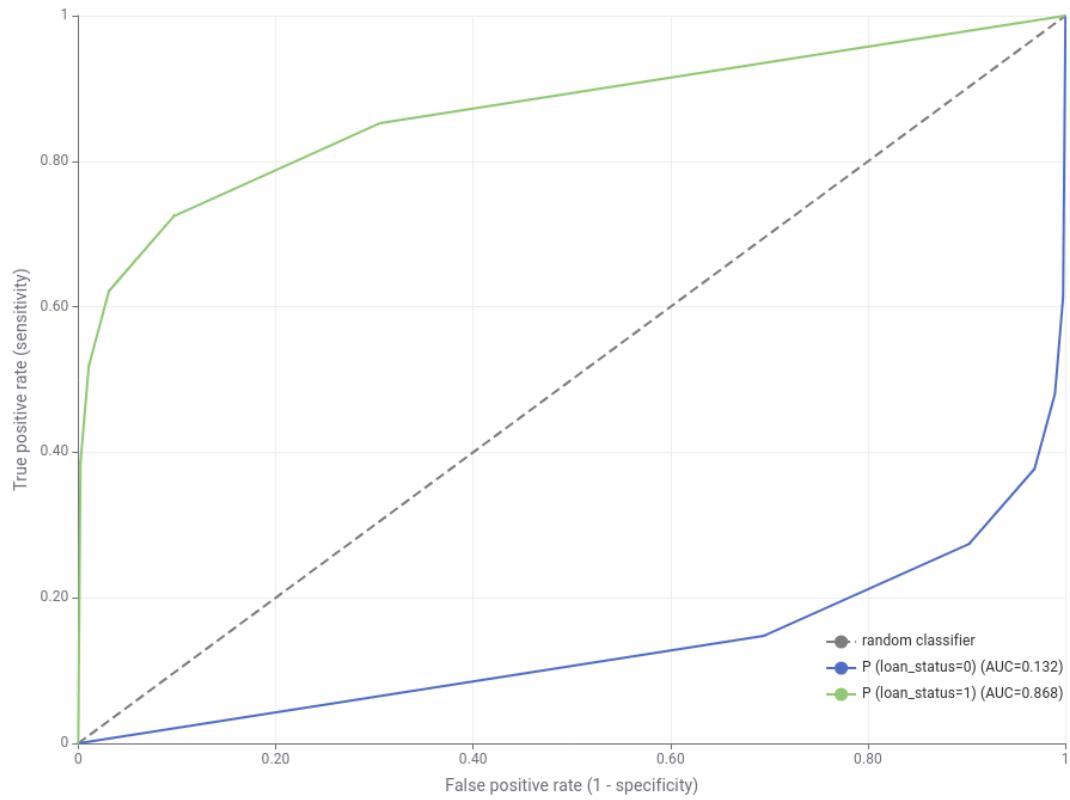
### CURVA ROC de LOGISTIC REGRESSION:



### CURVA ROC de MLP:



## CURVA ROC de KNN:



## 2. Predicción de una segunda cita

Algoritmo	Accuracy	Precision	Recall	F1-Score
Decision Tree	83.443%	82.925%	83.424%	83.201%
Random Forest	87.839%	87.268%	87.842%	85.221%
KNN	84.919%	81.695%	84.951%	80.391%
XGBoost Linear Model	87.775%	86.472%	87.77%	86.686%
RProp MLP	85.16%	84.236%	85.459%	84.593%

Para el problema de predicción de una segunda cita, los resultados de cada algoritmo en términos de precisión, recall, F1-Score y accuracy nos muestran cómo cada uno se adapta a las características de este problema.

**Decision Tree** tiene un desempeño sólido, con métricas balanceadas, lo que indica que el modelo es capaz de capturar de manera efectiva tanto los casos en los que una pareja estaría interesada en una segunda cita como los que no. Esto sugiere que el modelo ha logrado identificar patrones importantes en los datos, basados en criterios de división que distinguen bien las preferencias de los participantes. La ventaja de los

árboles de decisión es que trabajan bien con datos categóricos y numéricos y detectan patrones basados en valores específicos de los atributos. Aún así, el árbol de decisión muestra una accuracy algo menor que otros modelos más complejos lo cual puede deberse a su naturaleza individual y su tendencia a sobreajustarse a los datos de entrenamiento.

**Random Forest** obtiene la mejor accuracy entre todos los modelos (87.84%), con un balance destacado entre precisión y recall. Esto indica que el modelo es muy efectivo para predecir con exactitud tanto los casos positivos como los negativos. Al combinar varios árboles, **Random Forest** captura patrones complejos en los datos y reduce el riesgo de sobreajuste que podría haber afectado al **Decision Tree**. Al utilizar un criterio de selección como el Information Gain Ratio y al construir varios árboles, se han podido detectar interacciones sutiles entre los atributos de los participantes en la cita, que juntos pueden aumentar la probabilidad de una segunda cita. La alta precisión y recall sugieren que este modelo es capaz de identificar con precisión las combinaciones de atributos que predicen el éxito de una segunda cita, siendo el modelo más efectivo en general en este problema.

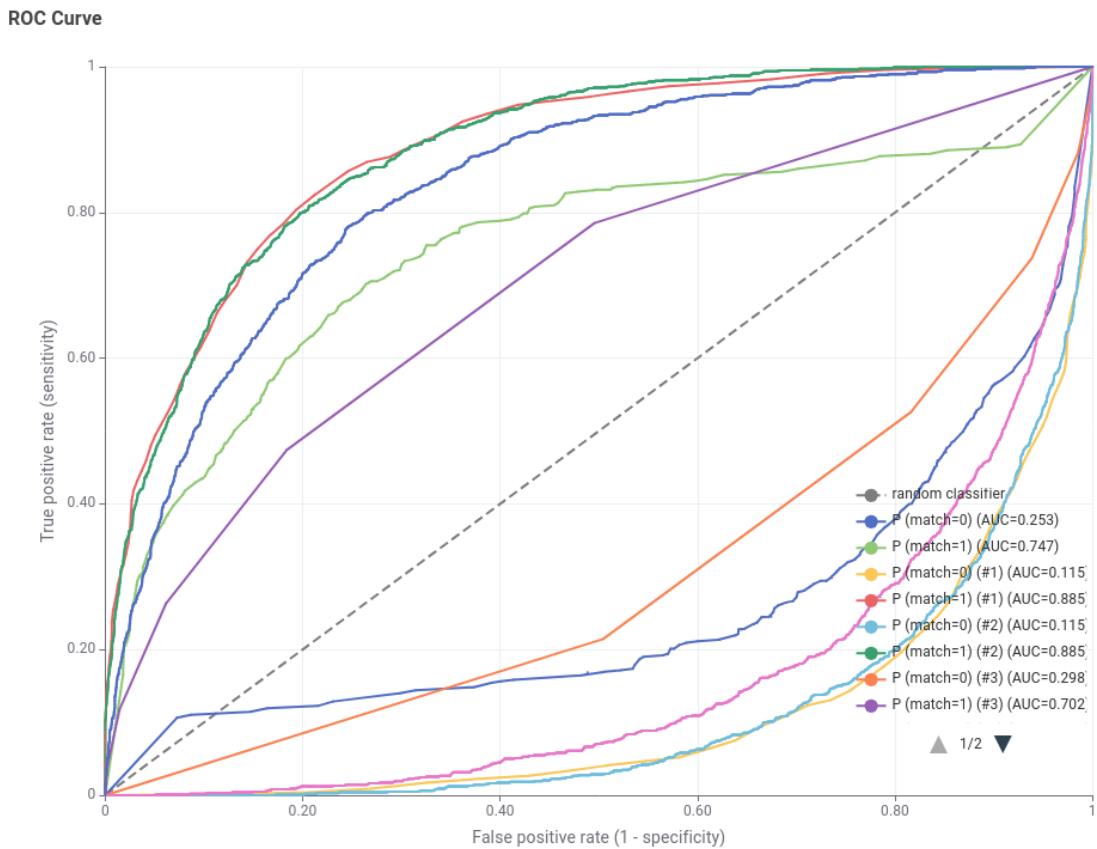
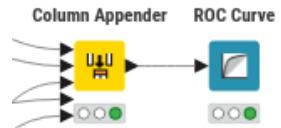
**KNN** presenta una accuracy de 84.92%, un poco menor que la de los modelos **Random Forest** y **XGBoost**. Aunque **KNN** alcanza buenos valores de precisión y recall, su F1-Score es el más bajo de todos los modelos, lo cual sugiere cierta dificultad para capturar con consistencia tanto los verdaderos positivos como los negativos. Esto puede deberse a que, al ser un modelo basado en la similitud de los vecinos, puede verse afectado por la variabilidad en los datos. En el contexto de citas, las preferencias y el atractivo son subjetivos y pueden variar significativamente entre individuos, lo que puede llevar a que **KNN** confunda casos similares en diferentes clases. Además, es sensible al ruido, y la elección de la métrica de distancia puede no ser la ideal para este problema, donde las relaciones entre variables son complejas y no siempre lineales.

**XGBoost** muestra un desempeño muy competitivo, casi igualando a **Random Forest** en términos de accuracy (87.77% vs 87.84%) y obteniendo un F1-Score alto, lo que refleja un excelente balance entre precisión y recall. Esto indica que el modelo es muy eficaz en la clasificación y en la identificación de patrones que predicen el éxito de una segunda cita. **XGBoost** es bueno manejando relaciones complejas y no lineales en los datos. En el contexto de citas, donde existen muchas variables que interactúan de forma compleja, puede aprovechar estas interacciones para hacer predicciones precisas. La ventaja de este modelo es que combina el boosting con árboles de decisión, lo cual permite capturar patrones sutiles y minimizar errores de predicción.

**Prop MLP** obtiene un rendimiento general bueno.. Esto indica que la red neuronal logra captar patrones complejos en los datos, aunque no alcanza la precisión de **Random Forest** o **XGBoost**. Las redes neuronales son potentes para detectar patrones no lineales y relaciones sutiles entre los atributos. Sin embargo, con la

configuración que le hemos dado, tiene una capacidad limitada en comparación con arquitecturas más complejas, lo cual podría explicar por qué su rendimiento es un poco inferior a los modelos de árboles más robustos. Además, las redes neuronales suelen necesitar muchos datos para entrenarse efectivamente, y en este problema, no tenemos un conjunto de datos demasiado grande.

De nuevo, utilizo curvas ROC para estudiar los resultados y sacar conclusiones. En las curvas ROC, cuanto más alto sea el AUC de la clase positiva y más bajo sea el de la negativa, nuestro modelo está siendo más efectivo al detectar los verdaderos positivos y descartar los negativos, que es justo lo que queremos en un problema de clasificación binaria como este. En este caso, he decidido usar un **Column Appender** para poder mostrar las 5 curvas ROC en un mismo gráfico y que así sea más fácil comparar los resultados.



Las dos primeras curvas son de **Decision Tree**, con una AUC de 0.747 para la clase positiva y de 0.253 para la clase negativa. Esto indica que el Decision Tree tiene un rendimiento decente para predecir correctamente cuándo habrá una segunda cita. La AUC de 0.253 en la clase negativa, cercana a 0, indica que el modelo es bastante bueno para identificar correctamente cuándo no habrá una segunda cita, lo cual es favorable en este caso. Esto sugiere que el árbol de decisión puede ser adecuado para

clasificar ambas clases con un balance aceptable, aunque otros modelos presentan un rendimiento superior en la clase positiva.

La tercera y cuarta son de **Random Forest**, con una AUC de 0.885 para la clase positiva y de 0.115 para la clase negativa. Esto sugiere que Random Forest es excelente para identificar correctamente las parejas que probablemente aceptarán una segunda cita. Además, la AUC de 0.115 en la clase negativa es bastante baja, lo cual es positivo, ya que indica que Random Forest también es efectivo en identificar correctamente las parejas que no tendrán interés en una segunda cita. Random Forest se muestra, por tanto, como un modelo robusto para este problema, logrando un buen balance en ambas clases.

La quinta y sexta son de **XGBoost**, con una AUC de 0.885 para la clase positiva y 0.298 para la clase negativa. Están casi solapadas con las de **Random Forest**, cosa que cuadra con los resultados de las métricas que hemos analizado antes, porque estos dos modelos fueron los mejores en la predicción.

Las dos siguientes son de **KNN**, con una AUC de 0.702 para la clase positiva y 0.298 para la clase negativa. Vemos que **KNN** es menos efectivo que **Random Forest** y **XGBoost** para predecir cuándo habrá una segunda cita, pues la curva está más cerca del random classifier. Fijándonos en esto, corroboramos que **KNN** seguido de **Decision Tree**, son los peores algoritmos para este problema de predicción de entre los cinco que hemos probado.

Las dos últimas (aunque no se ven en la leyenda de la foto) son de **MLP**, con una AUC de 0.839 para la clase positiva y de 0.162 para la clase negativa. Esto muestra que la red neuronal es excelente en predecir cuándo habrá una segunda cita, igualando casi el rendimiento de **Random Forest** y **XGBoost** en esta clase.

### 3. Predicción de enfermedades eritemato-escamosas

Algoritmo	Accuracy	Precision	Recall	F1-Score
Random Forest	97.814%	97.787%	97.802%	97.816%
SVM	96.448%	98.013%	96.448%	95.183%
KNN	92.411%	98.013%	92.4%	97.983%
XGBoost	97.268%	97.325%	96.906%	97.31%
Gradient Boosted Trees	92.411%	98.8%	98.8%	98.817%

**Random Forest** obtuvo la mejor combinación de Accuracy, Precision, Recall y F1-Score, con valores todos alrededor del 97.8%. Esto puede deberse a que es un modelo basado en árboles de decisión, que construye múltiples árboles y combina sus predicciones para tomar una decisión final. Es una estructura robusta y menos propensa al sobreajuste que un solo árbol de decisión. Captura bien las variaciones en los datos y se vuelve más preciso y generalizable. Esto le permite obtener buenos resultados en todas las métricas, ya que maneja bien tanto la detección de los positivos como la minimización de los errores.

**SVM** tiene una Precision alta (98%) pero un Recall algo menor (96.4%), lo que reduce un poco su F1-Score. **SVM** funciona creando un hiperplano que separa las clases de datos de manera óptima. Esto hace que sea excelente en distinguir muestras con gran certeza (alta Precision), ya que se enfoca en encontrar la frontera óptima. Sin embargo, en problemas donde algunas clases o casos pueden ser más complejos o mezclarse un poco, **SVM** puede no capturar todos los positivos correctamente, lo que afecta ligeramente el Recall. Es un modelo muy bueno para minimizar falsos positivos, pero puede perder algunos positivos reales, lo que explica su alta Precision y menor Recall.

**KNN** tiene la Precision alta (98%), pero su Accuracy y Recall son los más bajos entre los modelos, alrededor de 92.4%. Esto puede deberse a que clasifica basándose en la similitud con los vecinos más cercanos, y su desempeño depende de la distribución de los datos. Es bueno para capturar los positivos cuando están cerca en el espacio de características (lo que contribuye a su alta Precision), pero tiende a confundirse si hay datos mezclados o con ruido, lo que reduce su Recall y Accuracy.

**XGBoost** tiene métricas bastante altas, con Accuracy, Precision, Recall y F1-Score alrededor del 97.2% - 97.3%. Es una variante muy optimizada de Gradient Boosting, que también utiliza árboles de decisión pero en un enfoque secuencial. En cada iteración, intenta corregir los errores de los árboles previos, lo que permite capturar patrones complejos y mejorar el desempeño. Esto explica por qué sus resultados son casi tan buenos como los de **Random Forest**: ambos pueden manejar bien las complejidades de los datos y equilibrar los falsos positivos y negativos. La principal diferencia con **Random Forest** es que **XGBoost** ajusta mejor a los datos de entrenamiento, lo cual es útil para este tipo de problemas de clasificación.

**Gradient Boosted Trees** tiene la Precision, Recall y F1-Score más altos (98.8%), pero su Accuracy es un poco más baja (92.4%). Es similar a **XGBoost** en que se basa en árboles de decisión y en un enfoque de aprendizaje secuencial. Sin embargo, puede que el hecho de haberse centrado tanto en reducir falsos negativos y falsos positivos, haya hecho que el modelo sobreajuste ligeramente a los casos positivos, priorizando capturar todos los positivos posibles. Esto es ideal en situaciones donde queremos minimizar los errores en la detección de positivos (por ejemplo, en enfermedades

graves), pero afecta su Accuracy general porque puede aumentar los falsos positivos en las clases negativas.

En general, **Random Forest** y **XGBoost** son buenos en balancear todas las métricas y son los modelos más versátiles y confiables. **SVM** destaca en precisión pero puede fallar en capturar todos los casos positivos. **KNN** tiene una precisión alta en ciertas áreas, pero es más sensible a la distribución de datos, lo que afecta su rendimiento general. **Gradient Boosted Trees** sobresale en Recall y Precision, ideal cuando es crucial capturar todos los positivos, pero a costa de una menor Accuracy.

# Interpretación de los datos:

## 1. Predicción de Aprobación de Créditos

En general, analizando el **Decision Tree Predictor** o el **Naives Bayes Predictor**, vemos que un atributo que tiene mucha importancia es *loan\_percent\_income* (porcentaje de las ganancias del cliente que irán destinadas a pagar el crédito), pues si este número es menor que el 30.5%, el 85% de estos clientes no obtienen el crédito. De la gente que tiene un *loan\_percent\_income* > 30.5%, sólo el 70% consigue el crédito. Esto puede ser porque un menor porcentaje de ingresos dedicados al pago del préstamo sugiere una menor carga económica para el cliente, lo cual es favorable. Estos datos nos sugieren que los clientes con menor carga de deuda tienen un perfil de menor riesgo.

Hay algunos factores como la edad del cliente, la cantidad del crédito o los años que lleva trabajando, que no influyen tanto directamente en la aceptación del crédito, pues hay un amplio rango de estos atributos tanto en los clientes a los que se les aceptó el crédito como a los que se les denegó. Quizás sean más influyentes en combinación con otros atributos.

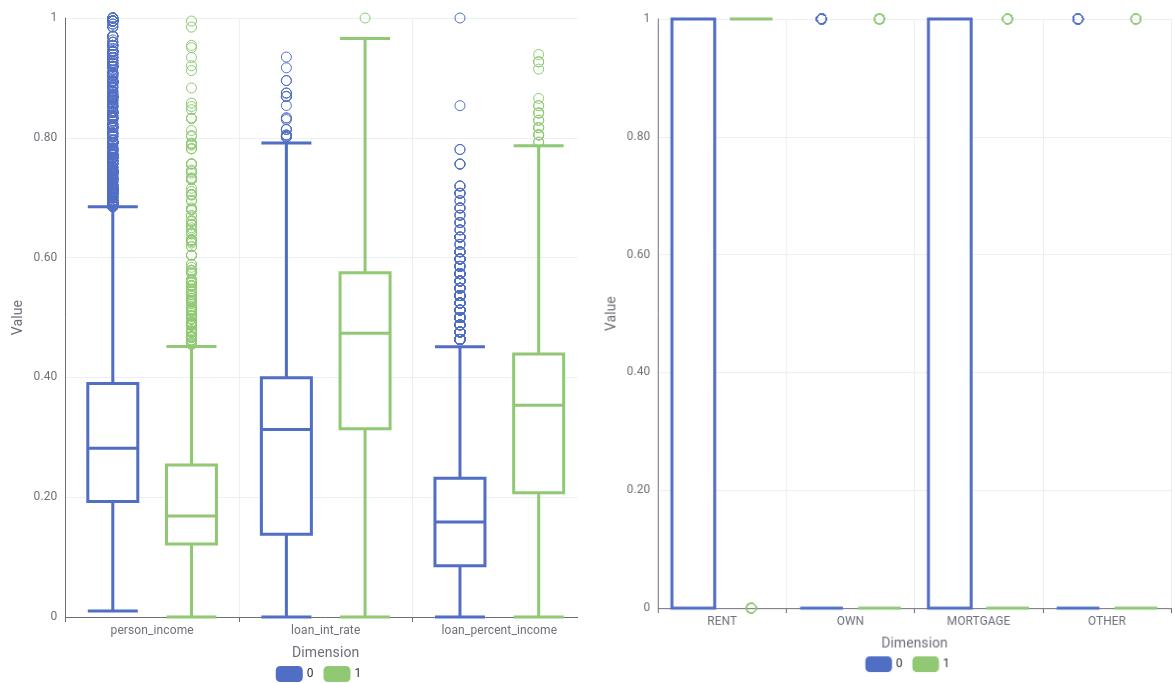
El campo de *person\_home\_ownership* es otro que influye mucho. La gente que tiene una hipoteca es prácticamente imposible que obtenga el crédito, pues puede suponer un alto riesgo por la carga financiera que ya tienen.

El propósito del crédito es bastante influyente, pues si es para fines médicos o para reformar la casa, vemos que es bastante más probable que lo aprueben que si es para otros fines. Esto puede deberse a que son necesidades básicas o inversiones en el hogar y pueden conllevar un menor riesgo que si es para fines personales.

Comparamos esto con algunos **Box Plot**:

Hay una correlación positiva entre el ingreso personal y la probabilidad de aprobación de crédito. Esto sugiere que los ingresos altos juegan un papel importante en la aprobación, posiblemente por ser un indicador de capacidad de pago. Se le suele aprobar el crédito a la gente que compromete mayor parte de sus ingresos para el pago del crédito.

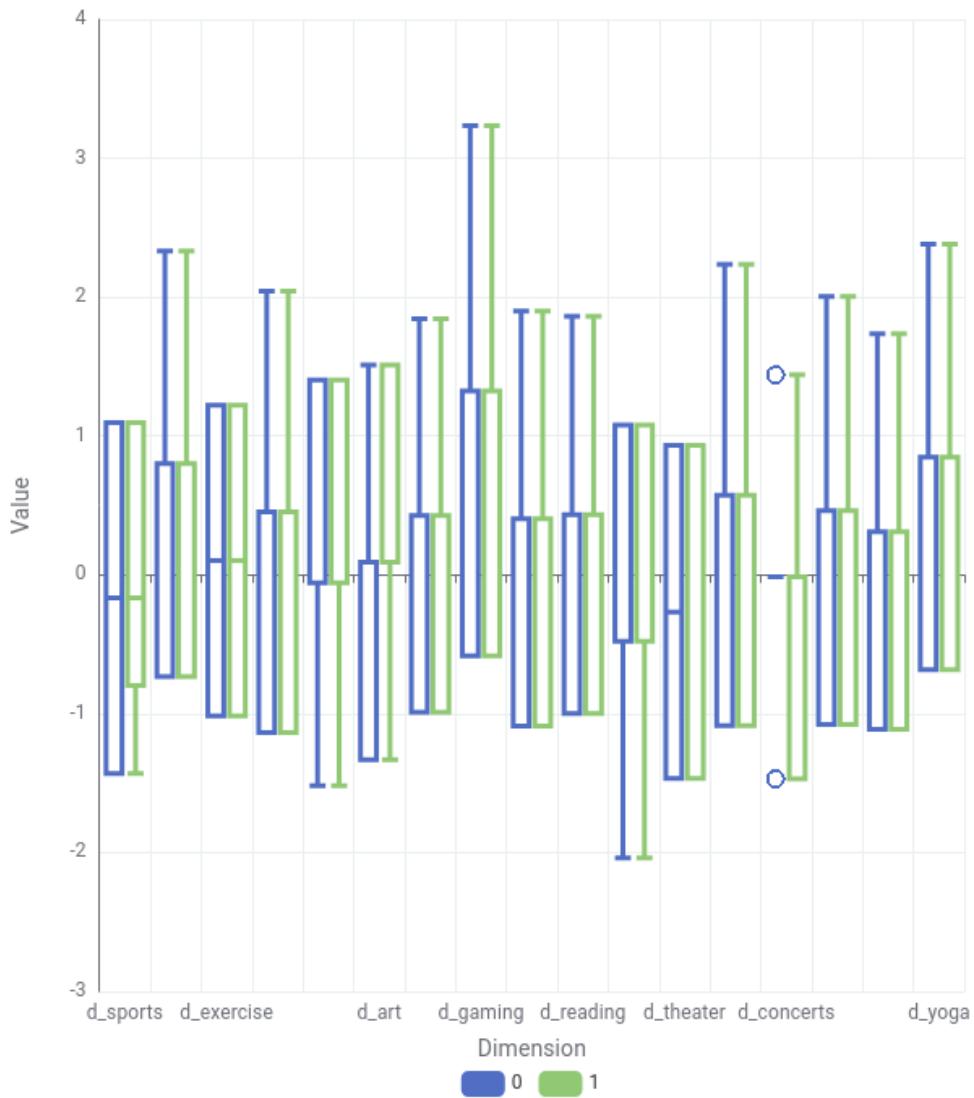
Vemos también que, en cuanto a la categoría *person\_home\_ownership*, una mayor proporción de préstamos serán rechazadas si la persona que lo pide alquila o tiene una hipoteca, como habíamos concluido antes. Las personas que tienen una casa tienden a obtener la aprobación del préstamo. Esta variable podría tener un impacto moderado en la predicción de la aprobación del crédito.



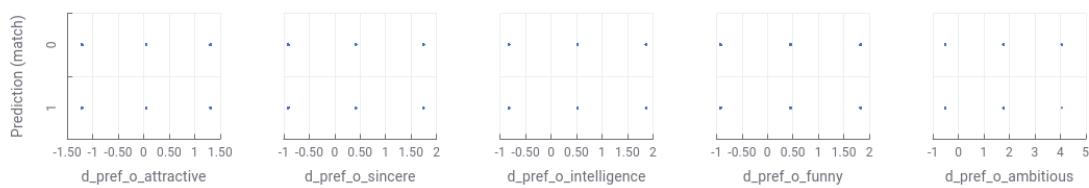
## 2. Predicción de una segunda cita

Para analizar qué factores son los más influyentes en la predicción de si dos personas tendrán una segunda cita, he utilizado varios gráficos y técnicas de análisis.

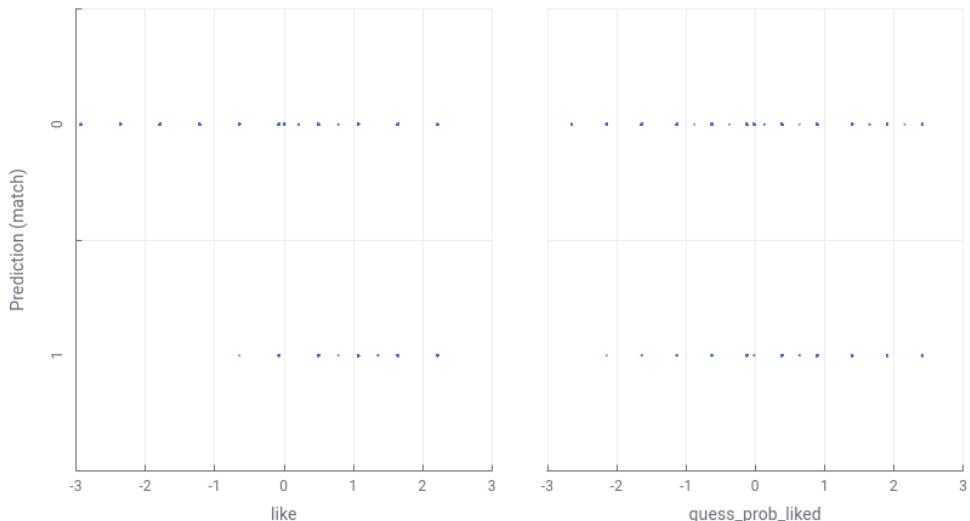
Con el **Box Plot**, observamos la relación entre la diferencia de interés en distintos campos (deportes, arte, gaming, lectura, yoga...) y la predicción de tener una segunda cita. No se perciben grandes diferencias significativas entre las distribuciones de estos intereses para aquellos que obtuvieron una segunda cita y los que no, lo que sugiere que la diferencia de intereses en estas actividades pueden no ser un factor determinante para tomar la decisión de tener una segunda cita.



En la **Scatter Plot Matrix** de las diferencias de preferencias de atributos de percepción de la pareja, podemos observar la relación entre características como diferencia de preferencia en atractivo, sinceridad, inteligencia, sentido del humor y ambición en relación con la predicción de tener una segunda cita. Vemos que aunque las diferencias en estas preferencias sean grandes, se puede tener una segunda cita. Lo que más destaca es que la gente que tiene grandes diferencias en la preferencia de las ambiciones es la que menos probable es que tenga una segunda cita.

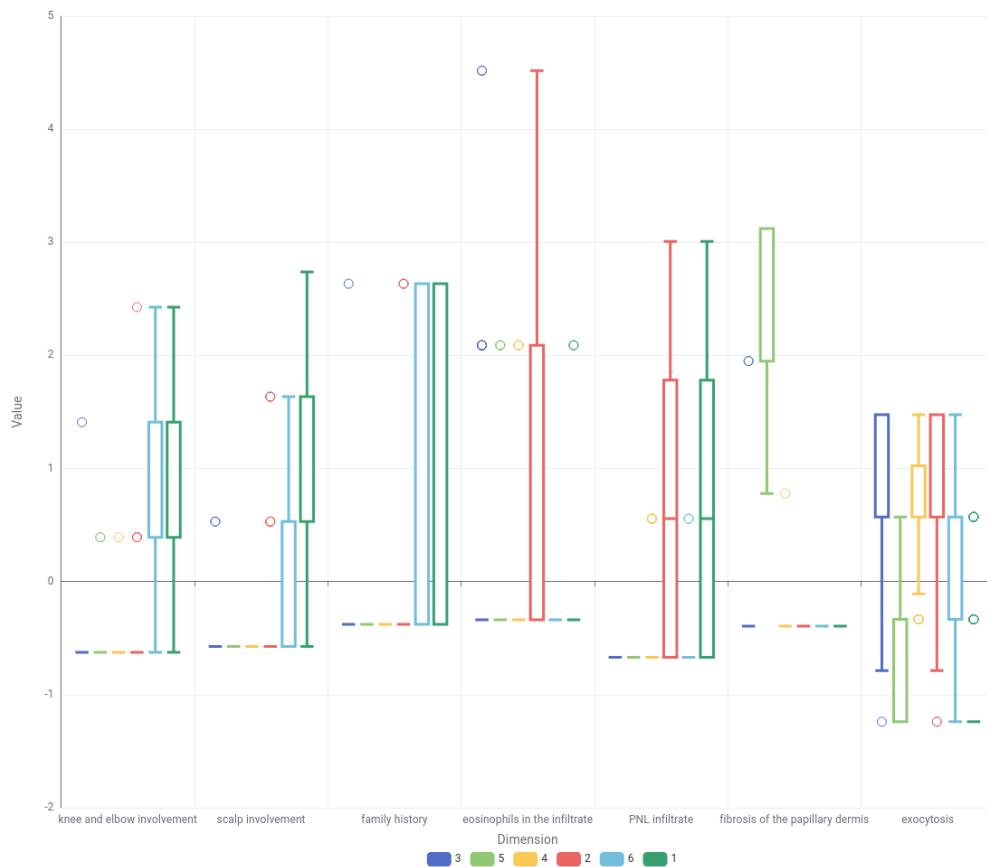


En el siguiente **Scatter Plot Matrix**, se examinan atributos como el gusto por la otra persona (like) y la estimación de probabilidad de ser correspondido (guess\_prob\_liked) en relación con la predicción de la segunda cita. Aquí, se observa que los participantes que tuvieron una mayor afinidad hacia sus parejas y una percepción más positiva de su probabilidad de ser correspondidos muestran una tendencia más alta a querer una segunda cita. Esto indica que los factores emocionales inmediatos como el gusto personal y la percepción de interés mutuo son aspectos críticos en la decisión de la segunda cita, más que características demográficas o intereses específicos.

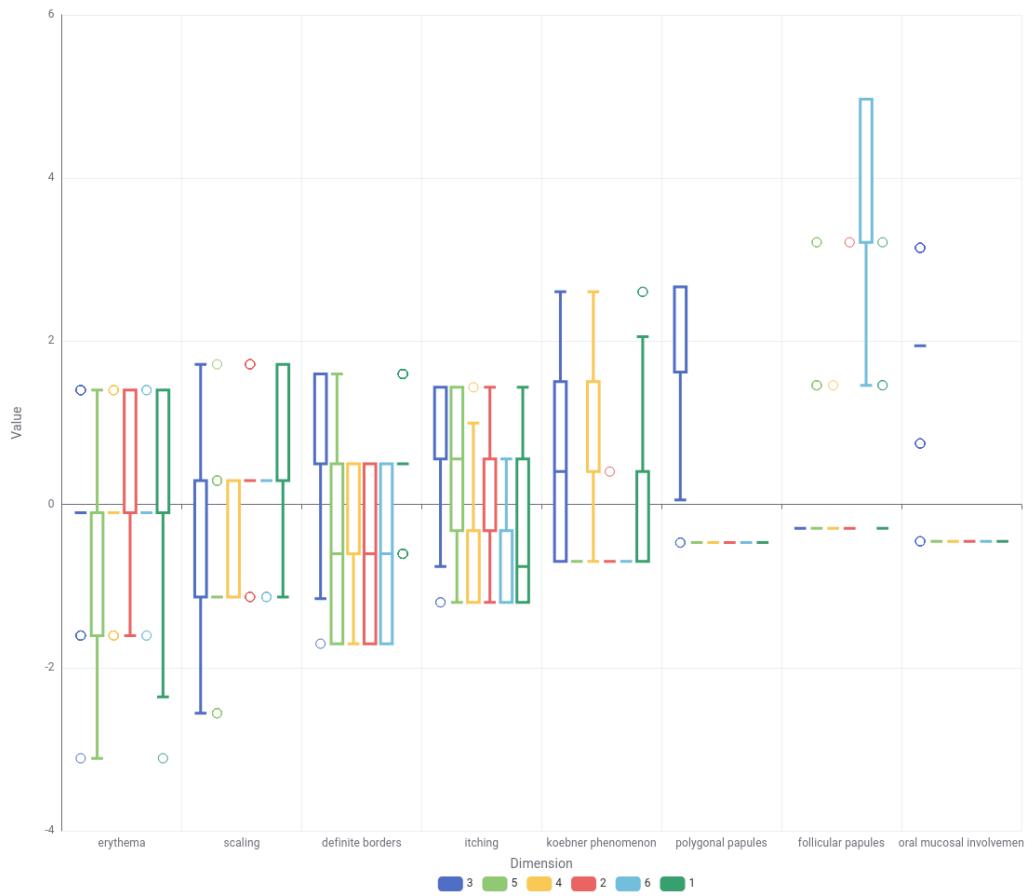


En general, las decisiones de segunda cita parecen depender más de factores subjetivos y emocionales, como el atractivo, la sinceridad y la afinidad emocional inmediata (gustar y probabilidad de ser correspondido). Los intereses personales específicos tienen una influencia menor; no obstante, puede haber ciertos intereses en común que, aunque no decisivos, faciliten la conexión inicial. Los factores como edad o el campo de estudio de los participantes no muestran patrones significativos, por lo que se pueden considerar atributos secundarios en la predicción de la segunda cita.

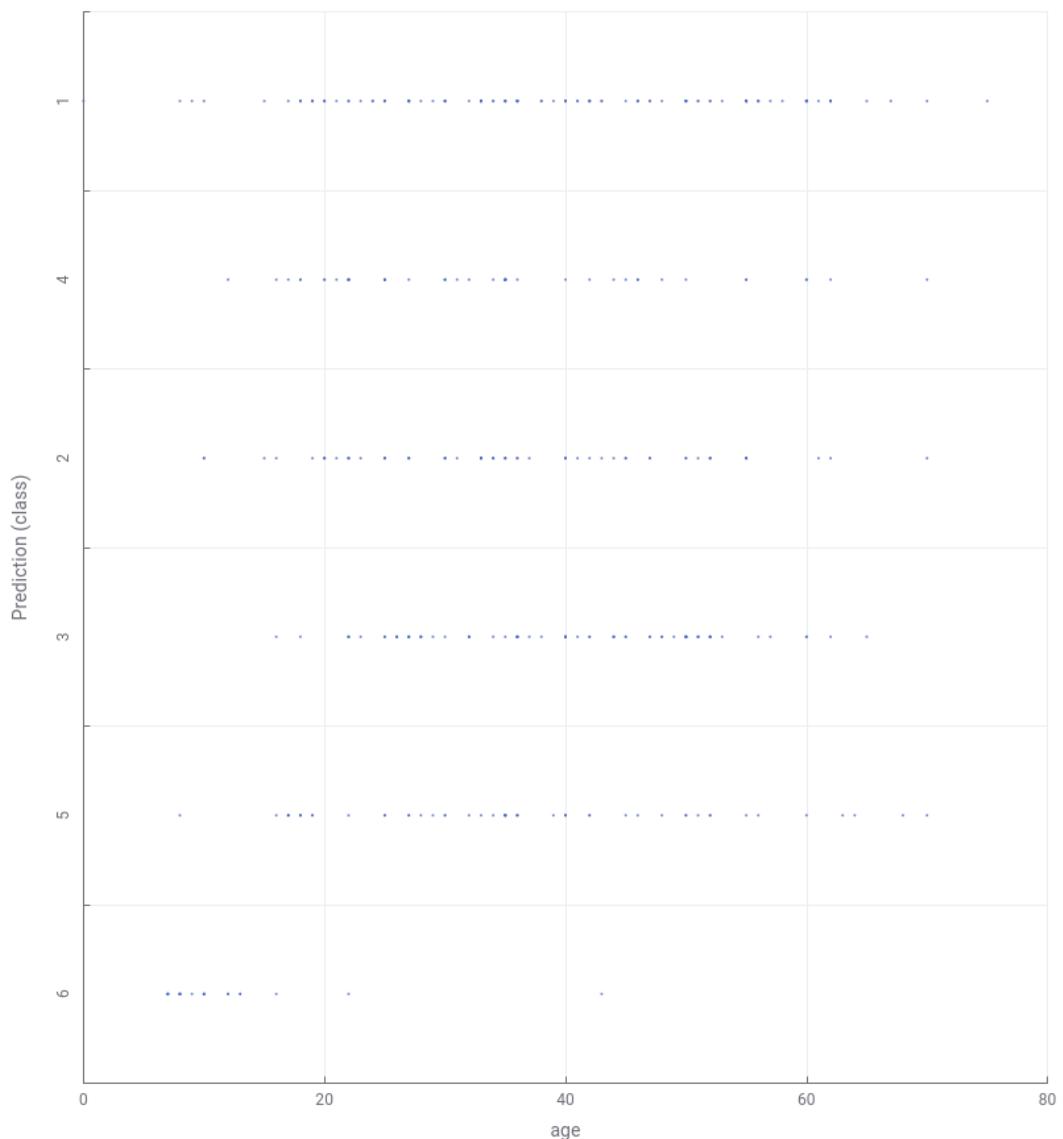
### 3. Predicción de enfermedades eritemato-escamosa



Los gráficos de **Box Plot** muestran cómo diferentes características (como la "presencia en rodillas y codos", "infiltardo de eosinófilos", "exocitosis", "escamas", "picazón", entre otras) están distribuidas en diferentes clases. Se observan variaciones en la distribución de cada característica según la clase. Por ejemplo, el "infiltardo de PNL" y la "fibrosis de la dermis papilar" tienen una mayor dispersión en clases específicas, lo que sugiere que estas características podrían ser más relevantes para diferenciar algunas clases de enfermedades. En el caso de ambos tipos de papilas y "afectación mucosa oral", los valores son bastante bajos y constantes en la mayoría de las clases, lo que sugiere que estas características pueden no tener un impacto significativo en la diferenciación entre clases. Características como el "historial familiar" y la "presencia en el cuero cabelludo" presentan variabilidad en diferentes clases, lo que puede ser indicativo de que estas variables están relacionadas con algunas enfermedades específicas y no con todas. Esto podría ayudar a mejorar la precisión en la clasificación. Las variables relacionadas con síntomas visibles, como "eritema" y "bordes definidos", también presentan una ligera variación según la clase, lo que indica que estas características también son relevantes en la predicción de algunas enfermedades en particular.



Con el **Scatter Plot Matrix** muestro la relación entre la edad del paciente y la clase de enfermedad predicha. No se observa una tendencia clara que relate la edad con una clase específica de enfermedad, ya que las clases parecen distribuidas de manera uniforme en todas las edades. Esto sugiere que, en este caso, la edad podría no ser una variable determinante para diferenciar entre las clases de enfermedades, al menos no de manera directa.



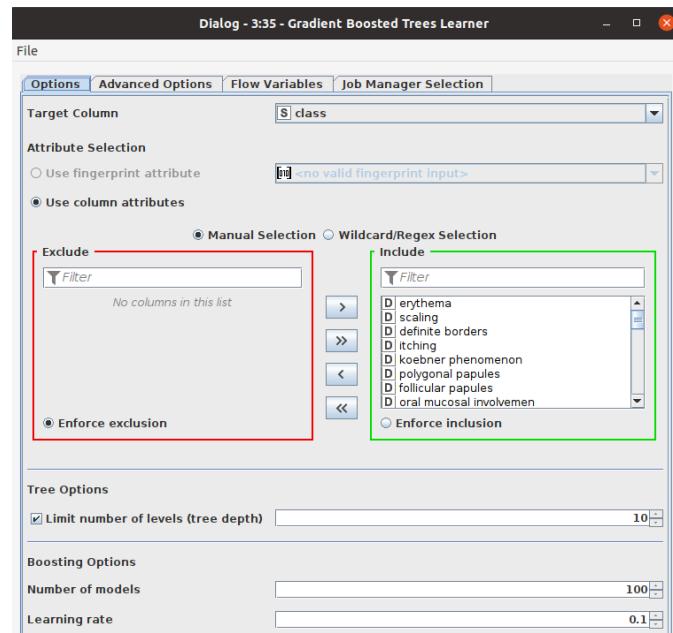
En general, las variables relacionadas con síntomas visibles y específicos, como "infiltrado de eosinófilos", "involucramiento en cuero cabelludo" e "historial familiar", parecen tener un impacto en la predicción de ciertas enfermedades, dada su variabilidad en distintas clases. Características con valores bajos y constantes, como "afectación mucosa oral", podrían ser menos relevantes para la predicción y podrían contribuir menos a diferenciar entre clases. La edad no parece tener un papel significativo en la clasificación, ya que no muestra una correlación clara con las diferentes clases de enfermedades.

## Contenido adicional:

De acuerdo con el punto 5 de las tareas a realizar en esta práctica, he decidido alterar las configuraciones de los algoritmos que utilicé en el ejercicio 3, para ver cómo esto afecta a los resultados.

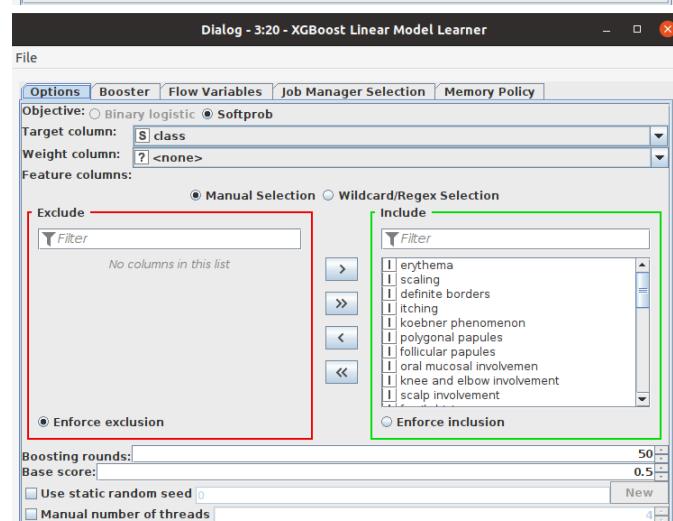
### Gradient Boosted Trees:

He puesto la profundidad máxima del árbol a 10, lo cual limita la complejidad del modelo, evitando el sobreajuste y mejorando la generalización en los datos de prueba. Se debería también reducir el tiempo de entrenamiento y de predicción, pero como nuestro conjunto de datos no es especialmente grande, puede que no sea una diferencia notable.



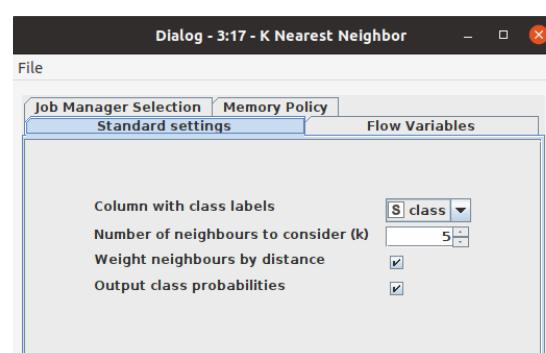
### XGBoost Linear Model:

He disminuido el número de boosting rounds de 100 a 50. Esto debería hacer que el modelo sea menos complejo y reducir el riesgo de sobreajuste a los datos de entrenamiento, es decir, a síntomas específicos.



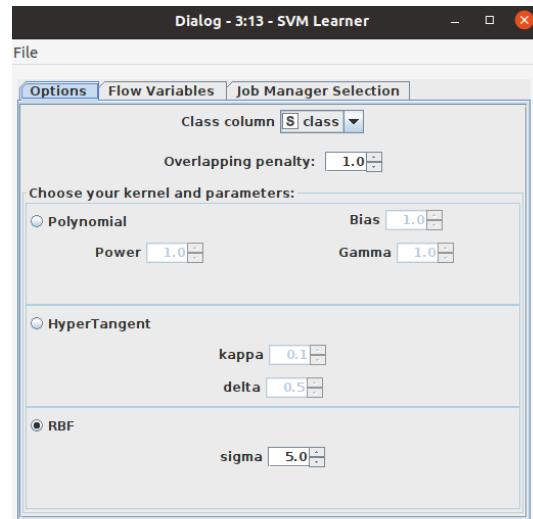
### KNN:

He activado la opción de Weight neighbours by distance. Esto debería aumentar la precisión, pues los datos tienen patrones locales y los casos más cercanos deberían tener una mayor relevancia para la predicción (casos de enfermos con historial similar). Debería volverse también menos sensible al ruido, ya que las observaciones lejanas (potencialmente atípicas o menos representativas del patrón general) no contribuirán tanto al resultado final.



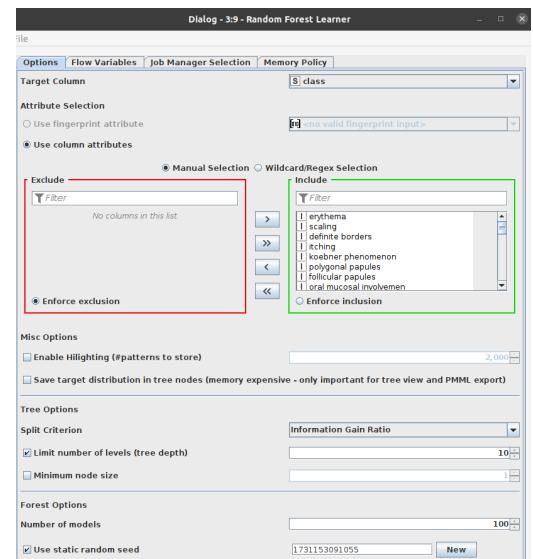
## SVM:

He cambiado el kernel a RBF con sigma=5, de Polynomial con gamma=1. Al hacer este cambio, el modelo usa esta medida de distancia para decidir cuánto debe depender cada punto de referencia o soporte en el espacio. Un valor de sigma alto significa que los puntos tienen menos influencia a medida que se alejan, permitiendo capturar agrupaciones naturales de los datos sin enfocarse demasiado en puntos aislados.



## Random Forest:

He limitado la profundidad máxima a 10 niveles. Esto debería ayudar a reducir el riesgo de capturar demasiado detalle y ruidos específicos de los datos de entrenamiento. En este caso, como algunas enfermedades son difíciles de separar, un árbol menos profundo puede generalizar mejor.



He usado la misma tabla para poder comparar más fácilmente:

Algoritmo	Accuracy	Precision	Recall	F1-Score
Random Forest	98.087%	98.384%	98.086%	98.087%
SVM	94.536%	94.462%	94.533%	94.497%
KNN	95.238%	95.433%	95.683%	95.217%
XGBoost	96.995%	97.048%	96.993%	97.029%
Gradient Boosted Trees	98.661%	98.667%	98.65%	98.683%

Los ajustes realizados en los parámetros de cada algoritmo han afectado sus métricas en diferentes grados, reflejando cambios en precisión, generalización y capacidad para capturar patrones específicos de las enfermedades dermatológicas.

En **Random Forest**, al reducir a 50 árboles y limitar la profundidad a 10 niveles, el modelo se ha vuelto menos complejo, mejorando ligeramente en todas las métricas (particularmente en precisión y F1-Score), ya que ahora se generaliza mejor y evita el sobreajuste.

Con **SVM**, cambiar el kernel a RBF con sigma 5 desde un kernel polinomial con gamma 1 ha hecho el modelo menos sensible a características específicas de cada clase, reduciendo un poco la precisión y el F1-Score. Esto indica que el RBF con sigma alto no capta tan bien las interacciones no lineales entre síntomas.

En **KNN**, activar la ponderación de vecinos por distancia ha mejorado las métricas, especialmente el recall, ya que el modelo ahora se centra más en los vecinos cercanos, reduciendo el impacto de instancias más lejanas y mejorando la clasificación de casos ambiguos.

En **XGBoost**, reducir los boosting rounds a 50 ha permitido un modelo menos ajustado y más eficiente, aunque ha bajado ligeramente en precisión y F1-Score, sugiriendo una menor capacidad para capturar todos los patrones complejos de las clases.

Finalmente, en **Gradient Boosted Trees**, aumentar la profundidad del árbol a 10 ha incrementado significativamente todas las métricas (particularmente el recall y el F1-Score), lo que sugiere que el modelo ahora capta mejor las complejidades de cada clase, aunque con un mayor riesgo de sobreajuste en futuros datos.

## Bibliografía:

- [1].<https://fastercapital.com/content/Outlier-detection--Detecting-Anomalies-with-Three-Sigma-Limits.html#The-Importance-of-Identifying-Outliers>
- [2].<https://forum.knime.com/t/filter-out-the-rows-containing-missing-values/10953/3>
- [3].<https://scikit-learn.org/1.5/modules/tree.html>
- [4].<https://www.geeksforgeeks.org/naive-bayes-classifiers/>
- [5].<https://www.ibm.com/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20>
- [6].[https://scikit-learn.org/1.5/modules/neural\\_networks\\_supervised.html](https://scikit-learn.org/1.5/modules/neural_networks_supervised.html)
- [7].<https://www.youtube.com/watch?v=AclQdjxpGA0>
- [8].<https://www.ibm.com/topics/random-forest>
- [9].<https://scikit-learn.org/1.5/modules/svm.html>