EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Matamoros

Materia: Inteligencia de negocios.

Proyecto final.

Unidad 4.

Docente: Wendy Aracely Sánchez Gómez.

Alumnos:

Omar Eduardo Amaro Pech

Raymundo Barrientos Alemán

Alex Fernández Sánchez

Adriel Eduardo Zayas Hernández

Ing. Sistemas computacionales. Grupo A. 8vo Semestre.

H. Matamoros, Tamaulipas, México. 1 de junio del 2023.

Introducción.

La capacidad de procesar grandes cantidades de datos y extraer información valiosa se ha convertido en una tarea fundamental en la era digital. En este contexto, el aprendizaje máquina ha surgido como una disciplina poderosa que permite a las máquinas aprender patrones y tomar decisiones automatizadas a partir de datos.

El procesamiento de datos con aprendizaje máquina abarca un amplio espectro de aplicaciones en diversos campos, como la medicina, la industria, las finanzas y la tecnología. Su objetivo principal es encontrar patrones, tendencias y relaciones ocultas en los datos que puedan ser utilizados para tomar decisiones informadas y generar predicciones precisas.

En este proyecto se utilizaron 2 software de análisis de datos que son Weka y Jupyter, para la realización de este análisis tuvimos que analizar los datos y balancear las clases en ambos softwares para que estos dieran un resultado mas certero.

Los modelos utilizados fueron Zerorules, IBk, NB, SMO, J48, RandomForest, AB J48, AB, ExtraTrees, KNeighbors; de los cuales destacaron Ibk, RandomForest y ExtraTrees.

El mejor de los resultados en Accuracy fue 0.637, en Precision fue 0.644, en Recall fue 0.637 y de FM fue 0.638.

Marco conceptual.

Aprendizaje de máquina. También conocido como machine learning en inglés, es una rama de la inteligencia artificial que se centra en el desarrollo de algoritmos y modelos que permiten a las máquinas aprender de los datos y tomar decisiones automatizadas sin ser explícitamente programadas.

Clasificador. Es un algoritmo o modelo que se utiliza para asignar objetos o instancias a diferentes categorías o clases. Su objetivo principal es predecir la clase o categoría a la que pertenece una nueva instancia desconocida basándose en el conocimiento adquirido a partir de un conjunto de datos de entrenamiento.

ZeroRules. Es un clasificador muy simple que predice siempre la clase mayoritaria en el conjunto de datos de entrenamiento. No utiliza ninguna información sobre los atributos, por lo que es un buen punto de referencia para comparar otros clasificadores más complejos.

IBk. Es un clasificador de vecinos más cercanos (K-nearest neighbors) que asigna una instancia desconocida a la clase más común entre sus K vecinos más cercanos en el espacio de características.

Naive Bayes (NB). Es un clasificador probabilístico basado en el teorema de Bayes. Asigna una instancia desconocida a la clase más probable dada la evidencia proporcionada por sus atributos, asumiendo que los atributos son independientes entre sí.

SMO (Sequential Minimal Optimization). Es un algoritmo de aprendizaje para las máquinas de vectores de soporte (SVM) que busca encontrar un hiperplano óptimo que separe las instancias de diferentes clases en un espacio de características de mayor dimensión.

J48. Es una implementación del algoritmo de árboles de decisión C4.5. Construye un árbol de decisión a partir del conjunto de datos de entrenamiento, donde cada nodo representa una pregunta sobre un atributo y cada rama representa una respuesta a esa pregunta.

RandomForest. Es un conjunto de árboles de decisión que se construyen a partir de muestras aleatorias del conjunto de datos de entrenamiento. Cada árbol en el conjunto emite una votación y la clase más común se elige la predicción final.

AdaBoost con J48 (AB J48). Es una variante del algoritmo AdaBoost que utiliza árboles de decisión J48 como clasificadores débiles en cada iteración. El algoritmo se enfoca en mejorar el rendimiento al asignar un peso mayor a las instancias mal clasificadas en cada iteración sucesiva.

AdaBoost (AB). Es un algoritmo de conjunto que combina varios clasificadores débiles para formar un clasificador fuerte. Cada clasificador débil se entrena en una versión ponderada del conjunto de datos, y los clasificadores débiles se combinan mediante votación ponderada para obtener la predicción final.

ExtraTrees. Es una variante de Random Forest que utiliza múltiples árboles de decisión construidos a partir de muestras aleatorias y selecciona aleatoriamente los umbrales de división en cada nodo del árbol.

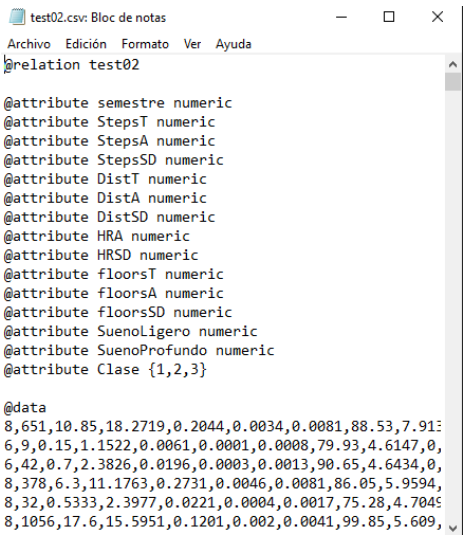
KNeighbors (K-vecinos más cercanos). Es un clasificador basado en instancias que asigna una instancia desconocida a la clase más común entre sus K vecinos más cercanos en el espacio de características.

Desarrollo.

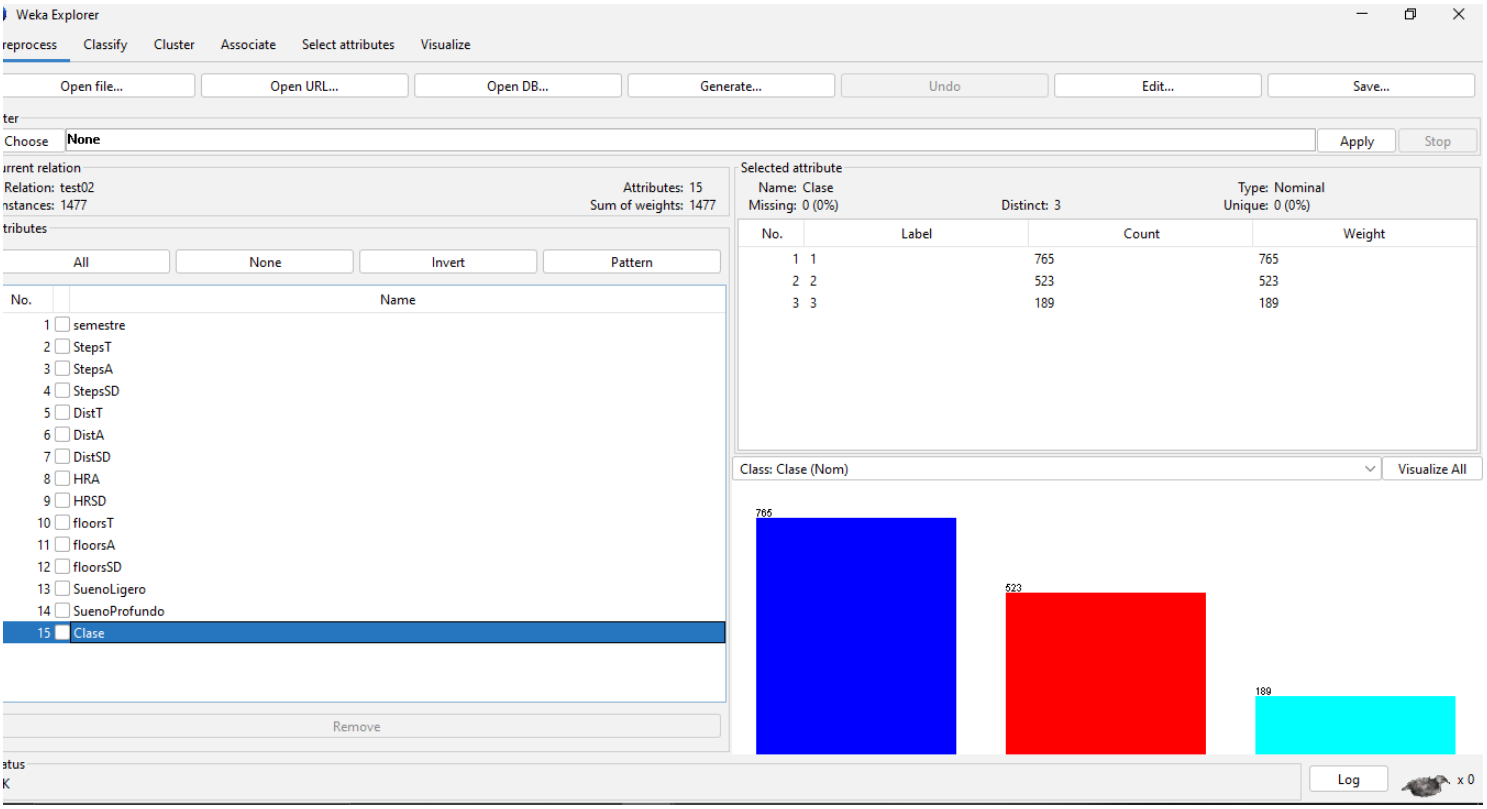
Para la realización del análisis utilizaremos dos softwares de análisis de datos, en este proyecto empleamos Jupyter Notebook y Weka.

Weka.

Para comenzar el análisis de datos en Weka necesitaremos cambiar la clase a nominal, el archivo .csv tiene que abrirse en formato de texto, después debemos de cambiarlo a. arff.

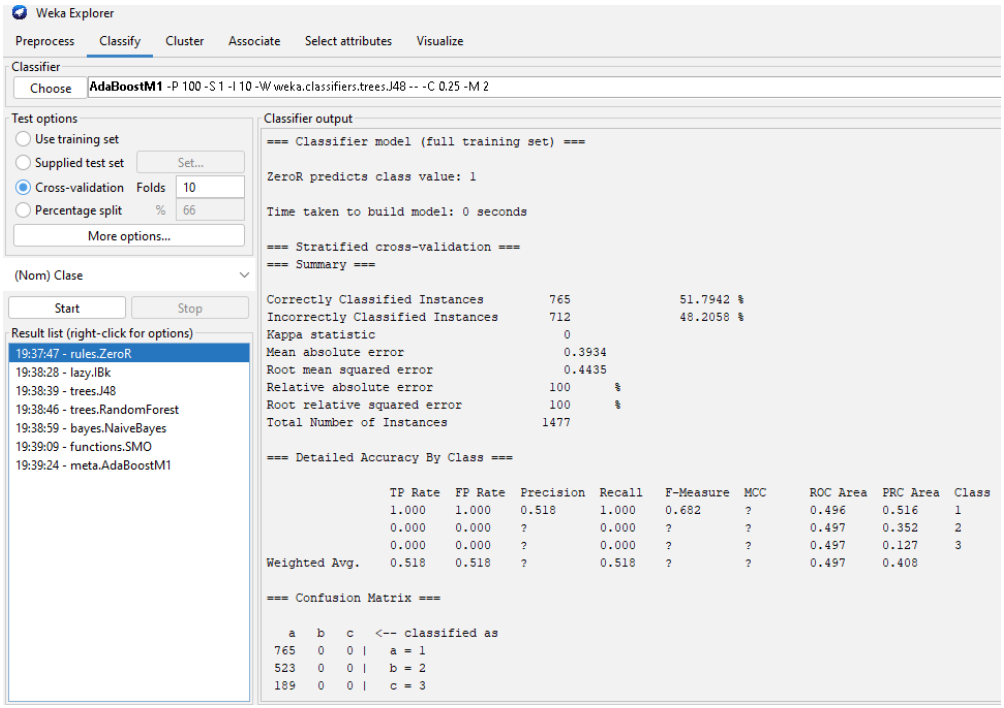


Aquí ya aplicado el cambio de numérico a nominal.

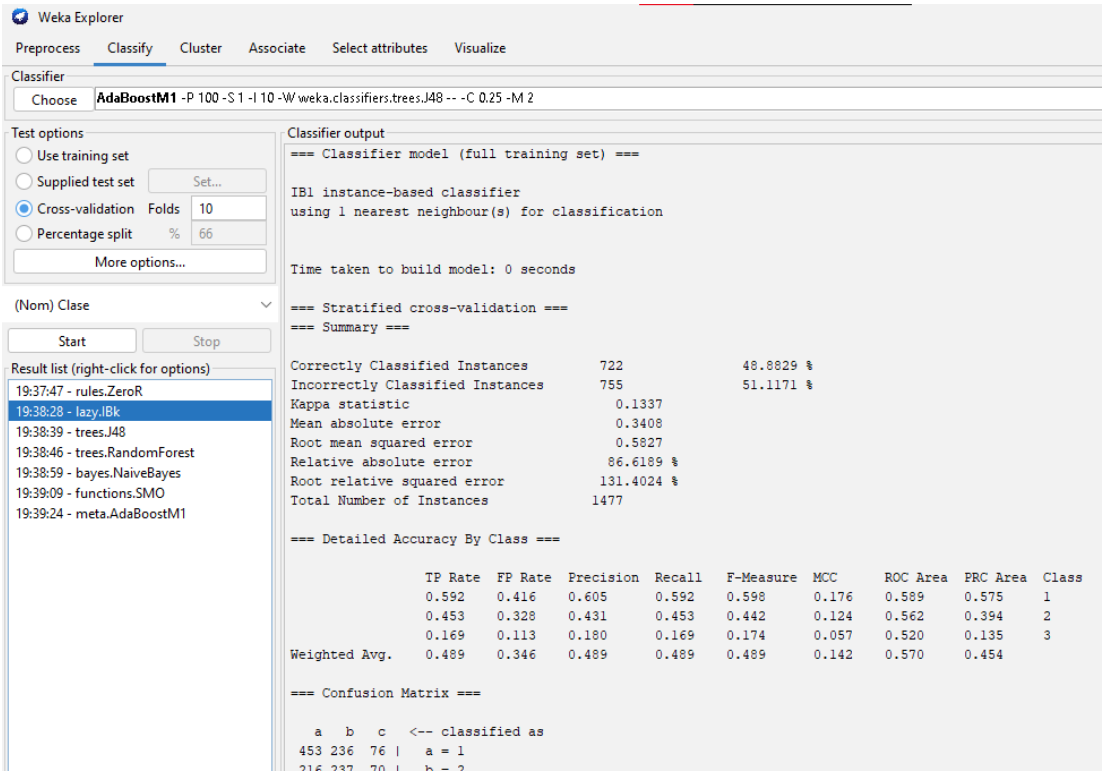


A continuación, empezaremos con la clasificación de modelos

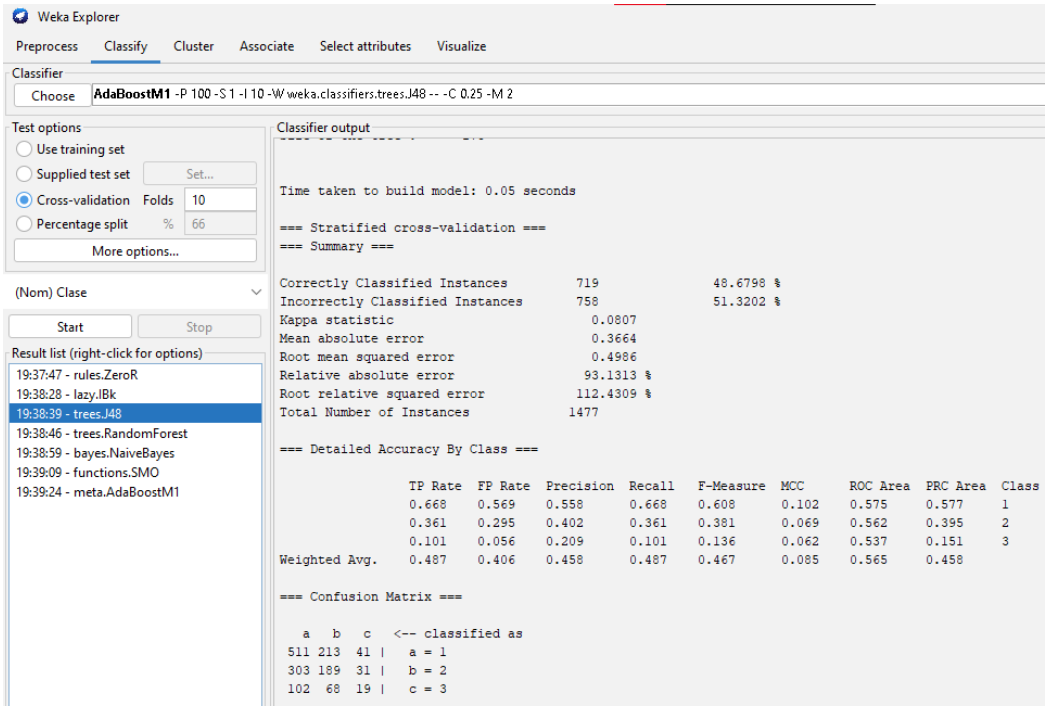
ZeroRules. Aplicamos el modelo zero rules, el cual dio como resultado:



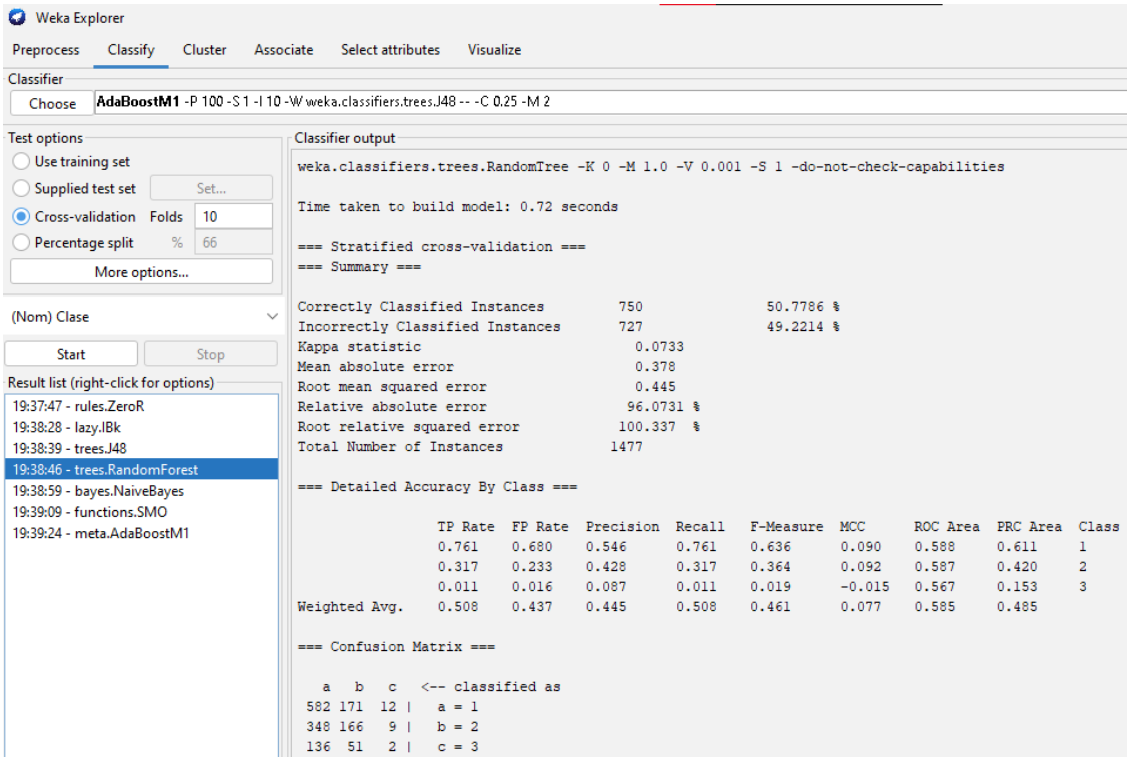
IBk. Aplicamos el modelo IBk, el cual dio como resultado:



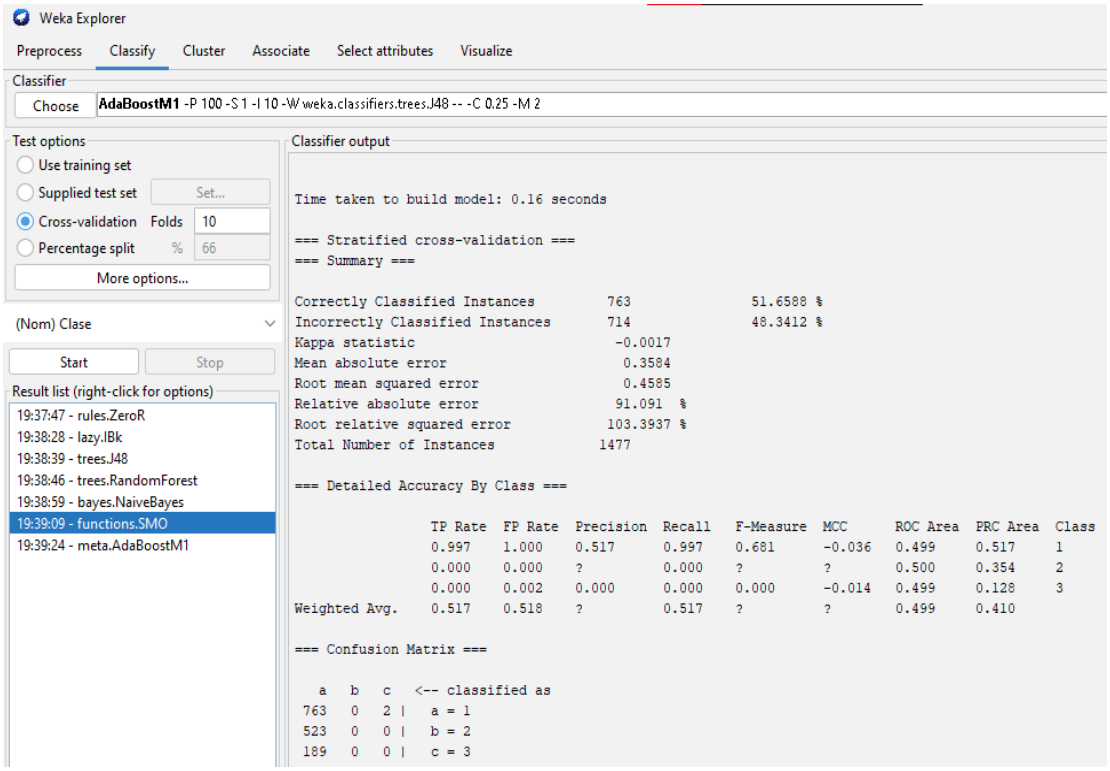
J48. Aplicamos el modelo J48, el cual dio como resultado:



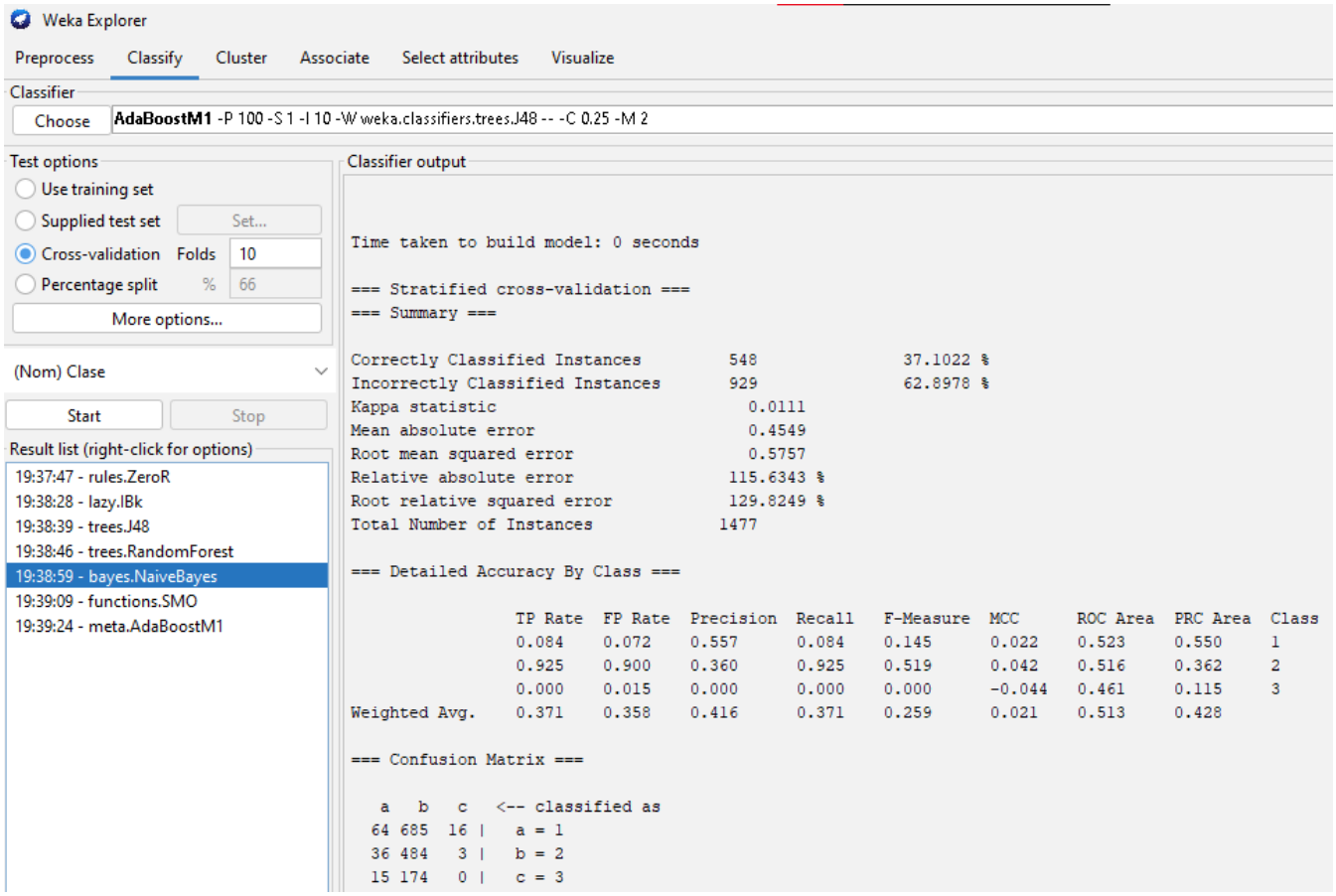
RandomForest. Aplicamos el modelo RandomForest, el cual dio como resultado:



NB. Aplicamos el modelo NB, el cual dio como resultado:



SMO. Aplicamos el modelo SMO, el cual dio como resultado:

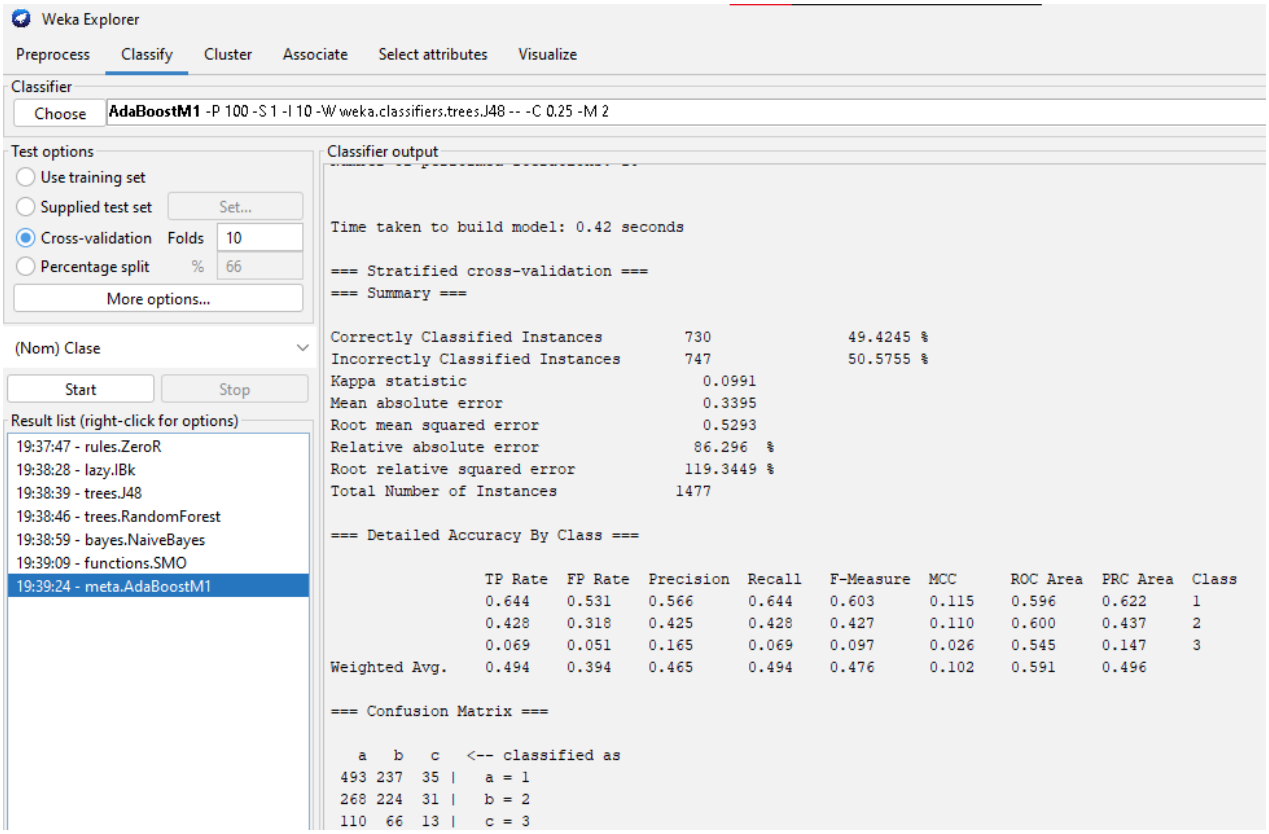


AB J48. Aplicamos el modelo AB J48, el cual dio como resultado:

Tabla completa.

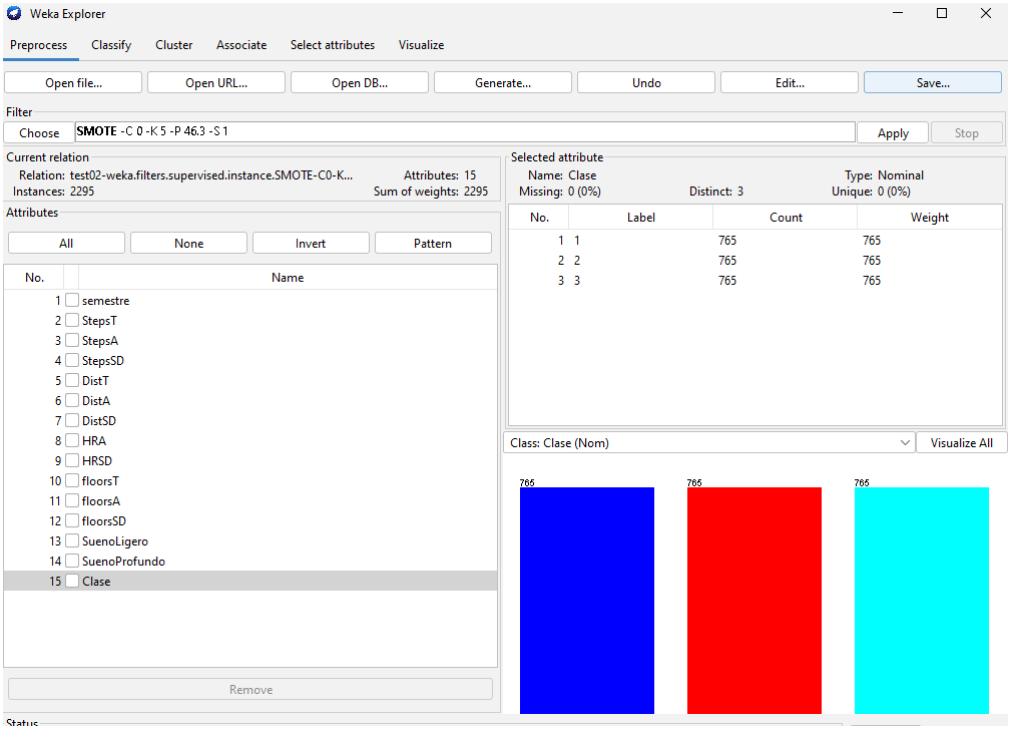
Test02				
	Accuracy	**Precision**	**Recall**	**FM**
ZR	0.517	?	0.518	?
IBk	0.488	0.489	0.489	0.489
NB	0.371	0.416	0.371	0.259
SMO	0.516	?	0.517	?
J48	0.486	0.458	0.487	0.467
RF	0.507	0.445	0.508	0.461
AB-J48	0.482	0.465	0.494	0.476

En esta prueba el mejor modelo fue el IBk, zerorules y smo contaban con un mayor porcentaje de precisión y recall, pero al no contar con un porcentaje de precisión y FM decidimos no ponerlos como válidos.

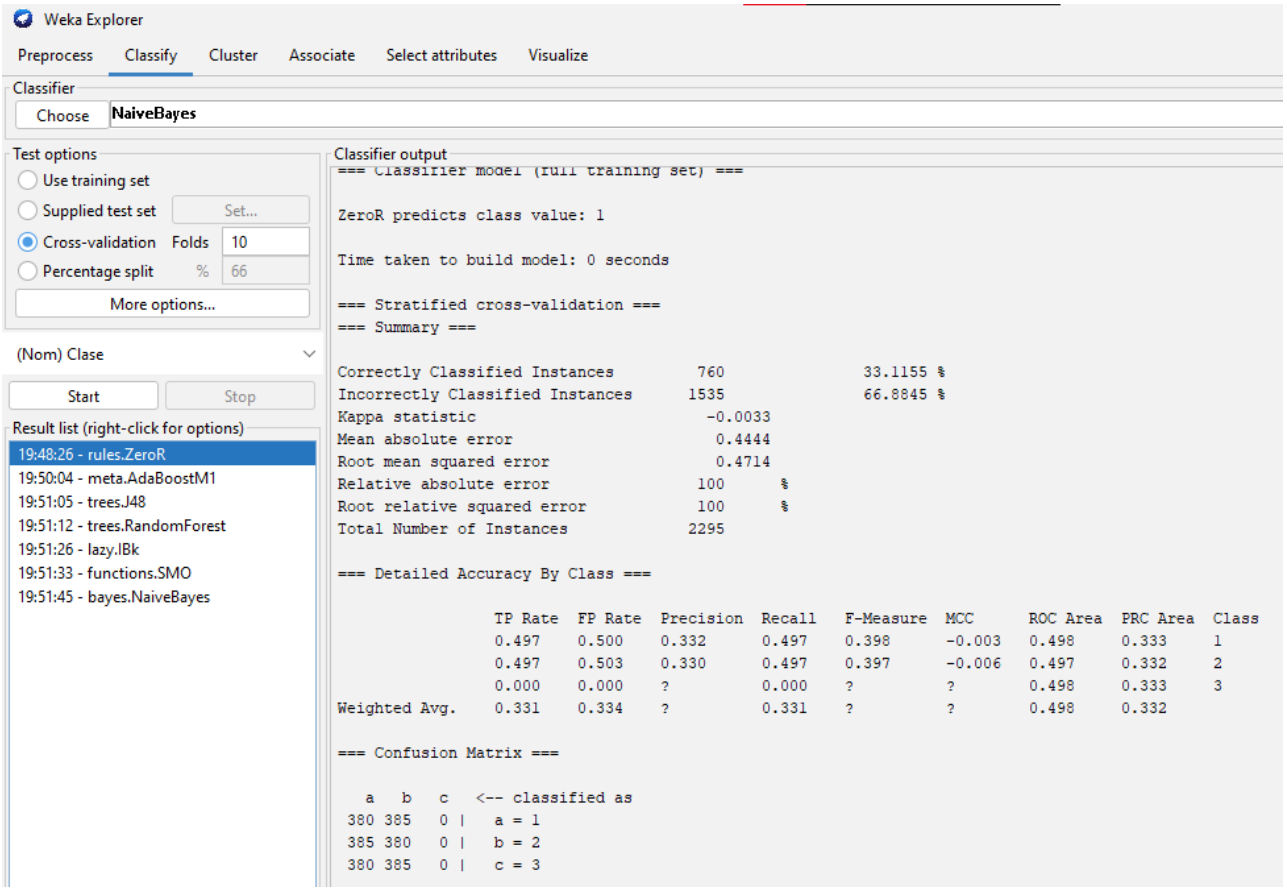


Weka Classify con smote.

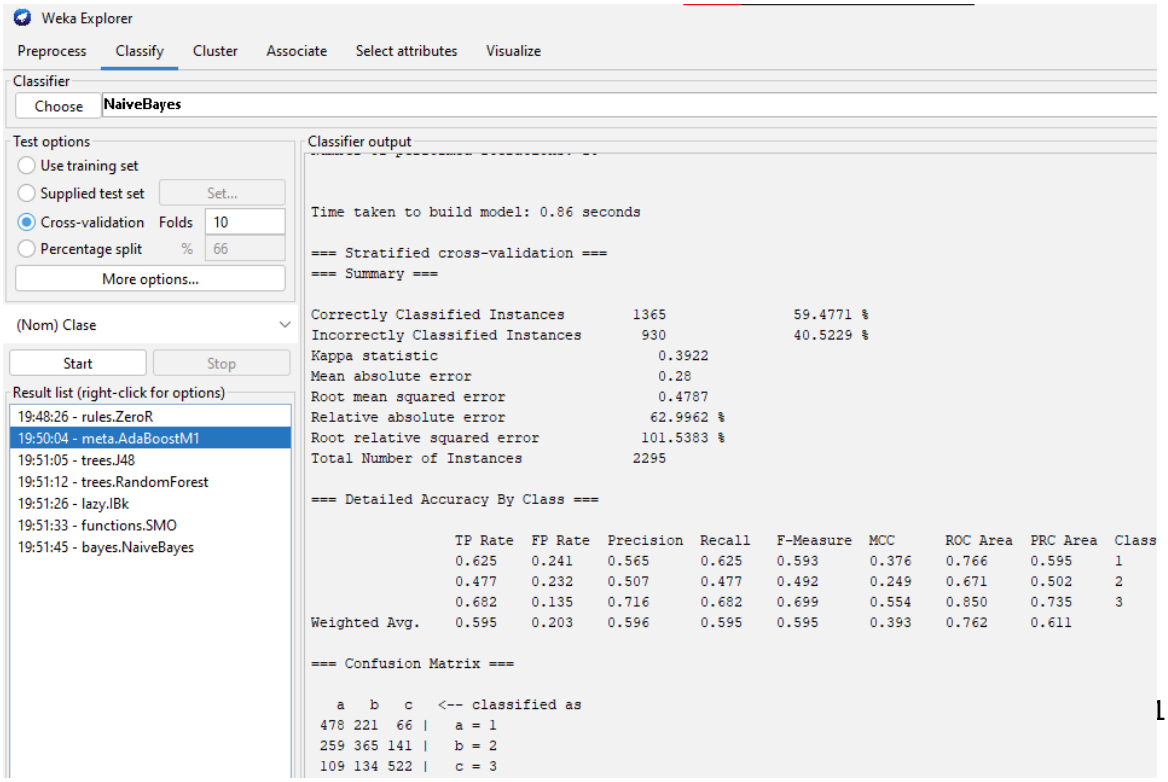
Aquí aplicamos el SMOTE con los porcentajes 305 y 46.6 para nivelar las clases, de esta forma tendremos las clases ya balanceadas y todas se encuentran en 765.



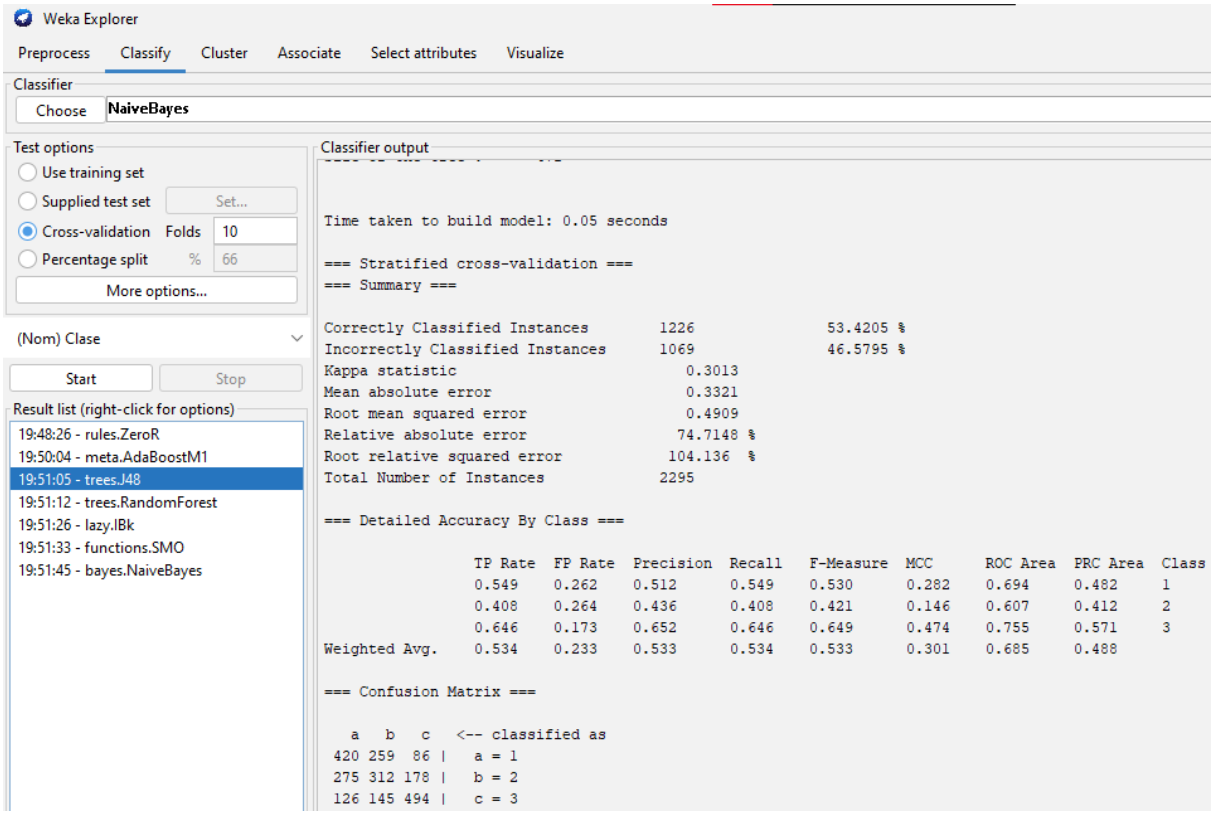
ZeroRules. Aplicamos el modelo ZeroRules, el cual dio como resultado:



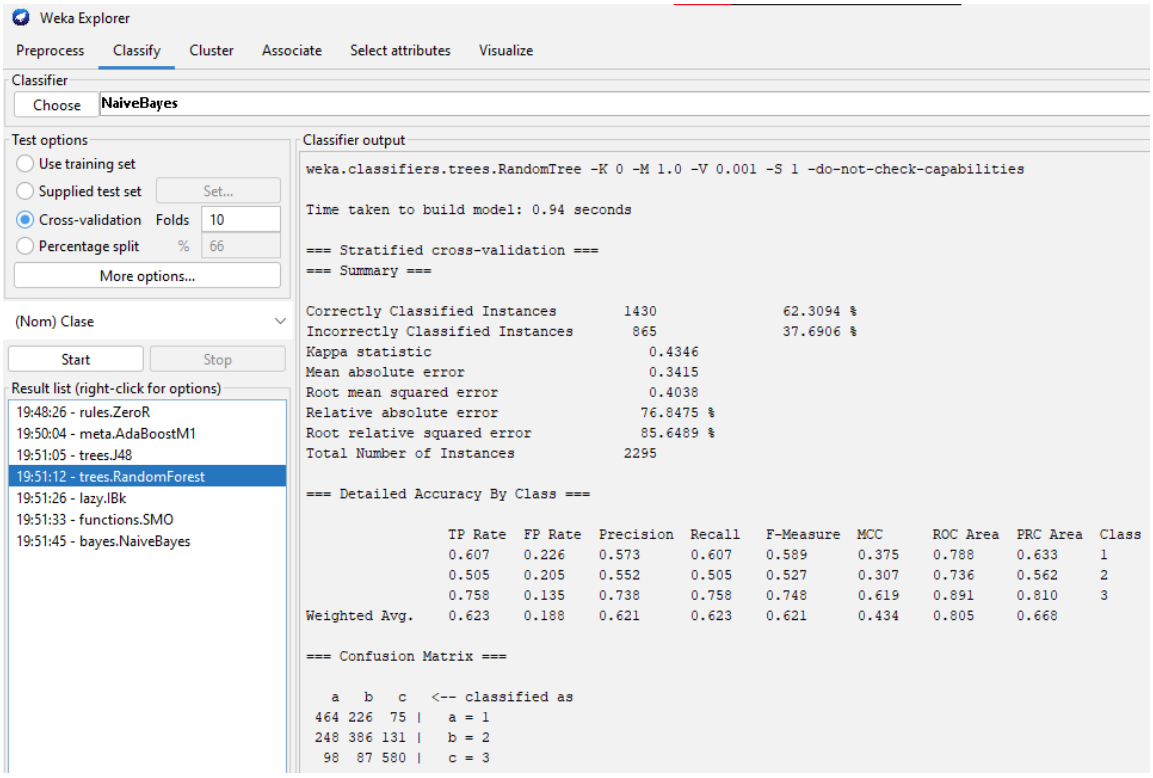
AB J48. Aplicamos el modelo AB J48, el cual dio como resultado:



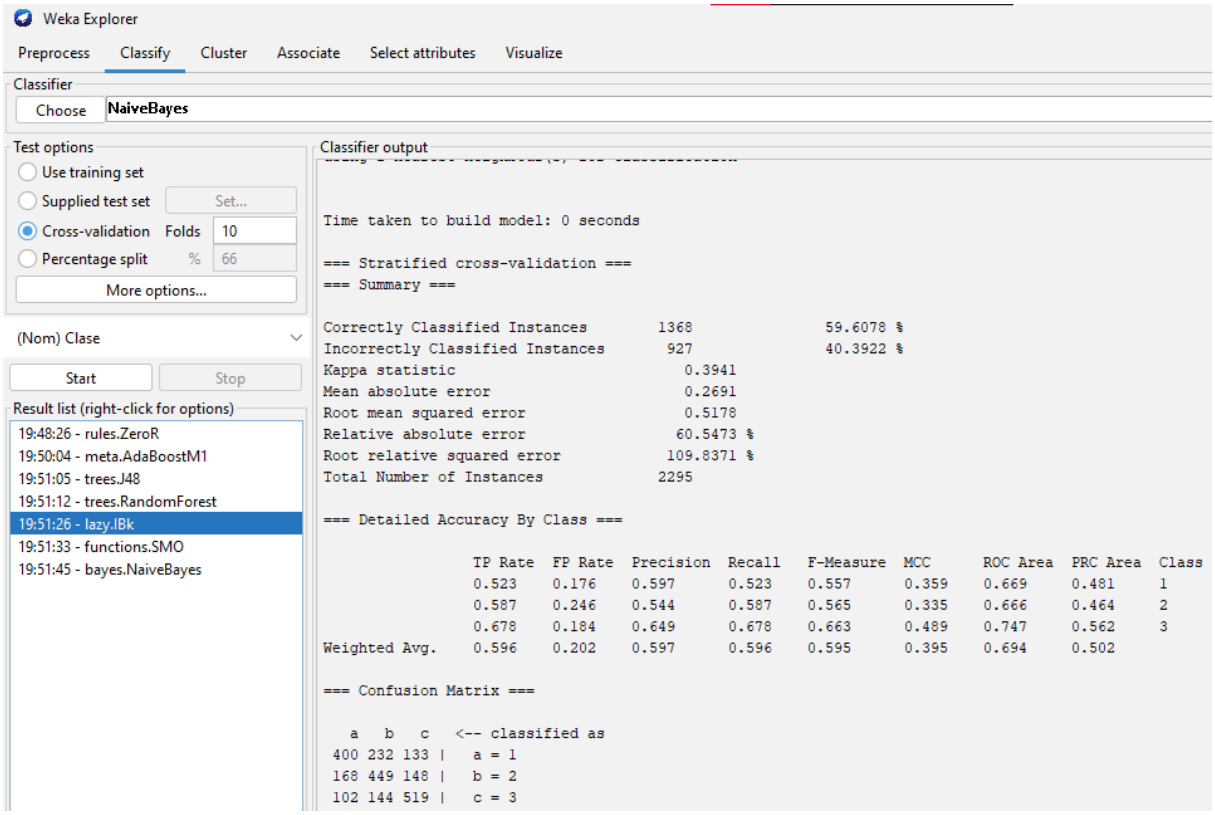
J48. Aplicamos el modelo J48, el cual dio como resultado:



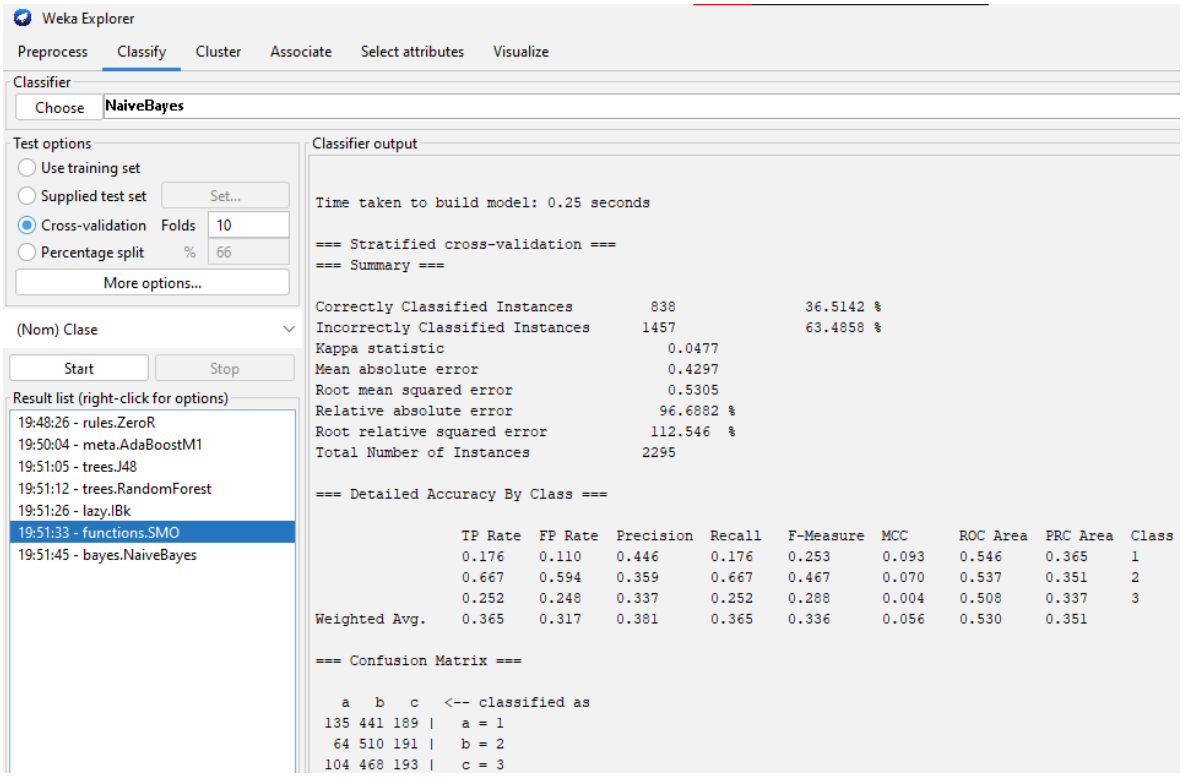
RandomForest. Aplicamos el modelo RandomForest, el cual dio como resultado:



IBK. Aplicamos el modelo IBk, el cual dio como resultado:



SMO. Aplicamos el modelo SMO, el cual dio como resultado:



NB. Aplicamos el modelo NB, el cual dio como resultado:

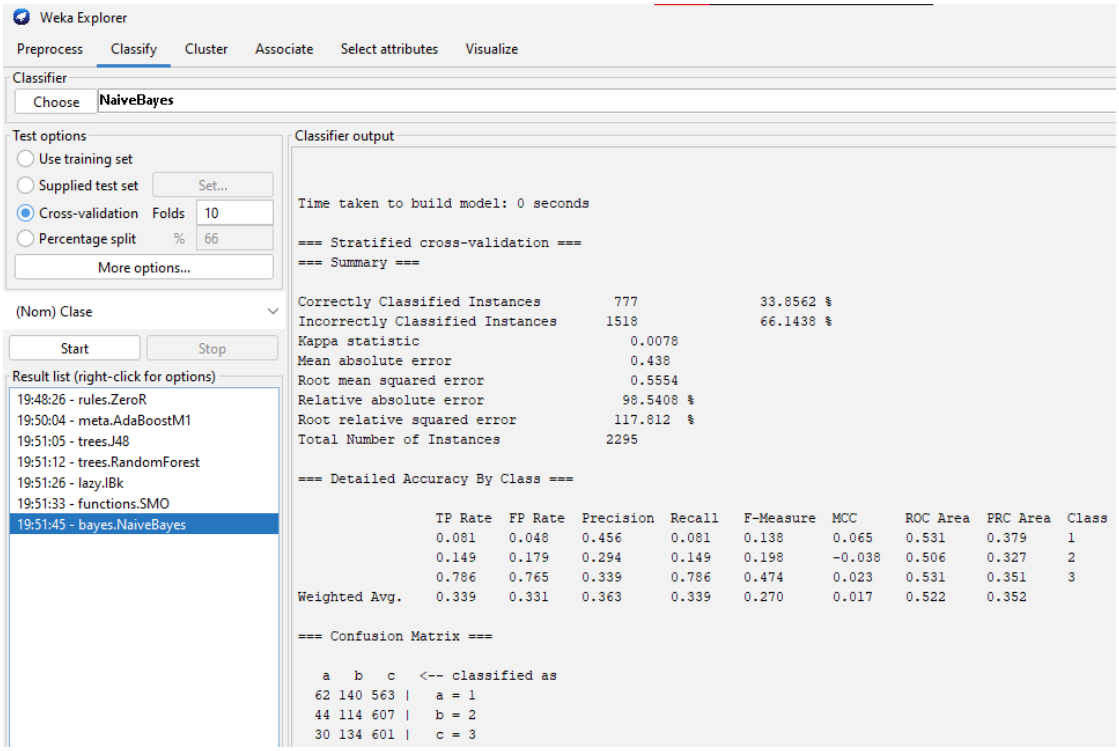


Tabla completa.

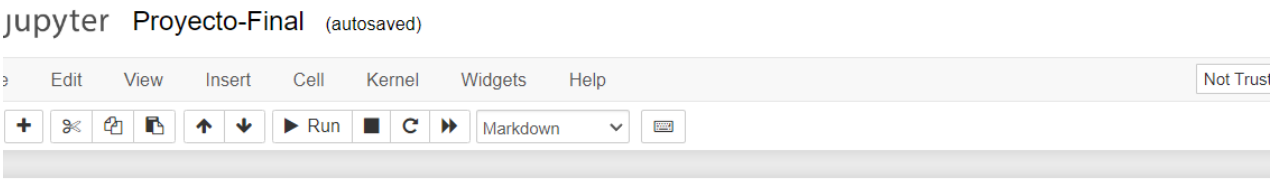
Test02 SMOTE				
	Accuracy	**Precision**	**Recall**	**FM**
ZR	0.331	?	0.331	?
IBk	0.596	0.597	0.596	0.595
NB	0.338	0.363	0.339	0.27
SMO	0.365	0.381	0.365	0.336
J48	0.534	0.533	0.534	0.533
RF	0.623	0.621	0.623	0.621
AB-J48	0.594	0.596	0.595	0.595

En esta prueba con SMOTE, el mejor modelo fue el RandomForest, al balancear las clases pudimos observar que SMO logro registrar datos en el precisión y FM. Además de que todos los modelos mejoraron significativamente su porcentaje en todos los parámetros.

Jupyter Notebook.

En Jupyter notebook se maneja a través de pestañas en la cual la primera la utilizaremos para proporcionar el nombre del proyecto.

En la siguiente pestaña importamos todas las librerias que utilizamos junto con los clasificadores.



Proyecto Final

```
In [1]: #Importar librerias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import cross_validate
#Classifiers
from sklearn.svm import LinearSVC, SVC #smo
from sklearn.tree import DecisionTreeClassifier #j48
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, ExtraTreesClassifier #adaboost
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB #NB
```

En la siguiente cargaremos el dataset y lo guardaremos en “df” y la imprimimos.

```
In [2]: #Cargar dataset
df = pd.read_csv('C:/Users/azayas/Downloads/test02.csv')
df
```

Out[2]:

	semestre	StepsT	StepsA	StepsSD	DistT	DistA	DistSD	HRA	HRSD	floorsT	floorsA	floorsSD	SuenoLigero	SuenoProfundo	Clase
0	8	651	10.8500	18.2719	0.2044	0.0034	0.0081	88.53	7.9134	1	0.0167	0.128	81	483	1
1	6	9	0.1500	1.1522	0.0061	0.0001	0.0008	79.93	4.6147	0	0.0000	0.000	20	285	2
2	6	42	0.7000	2.3826	0.0196	0.0003	0.0013	90.65	4.6434	0	0.0000	0.000	43	320	2
3	8	378	6.3000	11.1763	0.2731	0.0046	0.0081	86.05	5.9594	0	0.0000	0.000	16	475	1
4	8	32	0.5333	2.3977	0.0221	0.0004	0.0017	75.28	4.7049	0	0.0000	0.000	21	184	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1472	8	294	4.9000	10.9601	0.0000	0.0000	0.0000	69.47	4.1282	0	0.0000	0.000	37	378	1
1473	8	17	0.2833	1.5285	0.0000	0.0000	0.0000	78.57	4.0635	0	0.0000	0.000	13	483	1
1474	8	190	3.1667	9.4572	0.0000	0.0000	0.0000	67.05	5.6875	0	0.0000	0.000	13	483	2
1475	8	128	2.1333	5.5811	0.0000	0.0000	0.0000	70.80	3.3754	0	0.0000	0.000	37	378	3
1476	6	0	0.0000	0.0000	0.0063	0.0001	0.0006	80.53	2.6550	0	0.0000	0.000	59	470	1

1477 rows × 15 columns

A continuación, analizaremos los datos para lograr observar cual es el contenido de los datos y después de esto pedimos solo 5 datos para poder ver algo del contenido del dataset.

Entendimiento de los datos

In [3]:
```
df.describe()
```

Out[3]:

	semestre	StepsT	StepsA	StepsSD	DistT	DistA	DistSD	HRA	HRSD	floorsT	floorsA	floors
unt	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000
ean	7.248477	643.677725	5.158917	9.089846	0.453915	0.003034	0.005682	83.511547	6.700330	0.657414	0.005902	0.039
std	0.968966	10061.841653	7.617330	8.244214	7.566942	0.004574	0.005566	10.677834	3.218145	9.197496	0.017612	0.269
nin	6.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	50.670000	0.000000	0.000000	0.000000	0.000
5%	6.000000	43.000000	0.716700	2.933500	0.018100	0.000300	0.001300	76.570000	4.561800	0.000000	0.000000	0.000
0%	8.000000	159.000000	2.633300	7.321400	0.087600	0.001400	0.004400	82.680000	6.161200	0.000000	0.000000	0.000
5%	8.000000	374.000000	6.200000	12.531100	0.233300	0.003800	0.008200	90.130000	8.173300	0.000000	0.000000	0.000
nax	8.000000	365453.000000	68.000000	49.174200	254.960900	0.043100	0.033600	147.310000	48.226500	333.000000	0.300000	10.000

In [4]:
```
df.head()
```

Out[4]:

	semestre	StepsT	StepsA	StepsSD	DistT	DistA	DistSD	HRA	HRSD	floorsT	floorsA	floorsSD	SuenoLigero	SuenoProfundo	Clase
0	8	651	10.8500	18.2719	0.2044	0.0034	0.0081	88.53	7.9134	1	0.0167	0.128	81	483	1
1	6	9	0.1500	1.1522	0.0061	0.0001	0.0008	79.93	4.6147	0	0.0000	0.000	20	285	2
2	6	42	0.7000	2.3826	0.0196	0.0003	0.0013	90.65	4.6434	0	0.0000	0.000	43	320	2
3	8	378	6.3000	11.1763	0.2731	0.0046	0.0081	86.05	5.9594	0	0.0000	0.000	16	475	1
4	8	32	0.5333	2.3977	0.0221	0.0004	0.0017	75.28	4.7049	0	0.0000	0.000	21	184	2

Luego le pedimos que nos imprima los nombres de las columnas, que nos muestre la forma en que este hecho y por último le pedimos que sumara todos los nulos de cada columna y los mostrara.

In [5]:
```
df.columns
```

Out[5]:
```
Index(['semestre', 'StepsT', 'StepsA', 'StepsSD', 'DistT', 'DistA', 'DistSD',
       'HRA', 'HRSD', 'floorsT', 'floorsA', 'floorsSD', 'SuenoLigero',
       'SuenoProfundo', 'Clase'],
      dtype='object')
```

In [6]:
```
df.shape
```

Out[6]:
```
(1477, 15)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
semestre         0
StepsT           0
StepsA           0
StepsSD          0
DistT            0
DistA            0
DistSD           0
HRA              0
HRSD             0
floorsT          0
floorsA          0
floorsSD         0
SuenoLigero      0
SuenoProfundo    0
Clase            0
dtype: int64
```

A continuación, solicitamos que nos describiera la información que contenía el dataset, con las columnas, cuantas instancias tiene cada atributo, cuantos vacíos tiene cada uno y de qué tipo de dato es cada uno.

Y en la segunda se pide imprimir la cuenta de cada clase para saber cómo están distribuido los datos.

```
In [8]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1477 entries, 0 to 1476
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   semestre        1477 non-null   int64
 1   StepsT          1477 non-null   int64
 2   StepsA          1477 non-null   float64
 3   StepsSD         1477 non-null   float64
 4   DistT           1477 non-null   float64
 5   DistA           1477 non-null   float64
 6   DistSD          1477 non-null   float64
 7   HRA             1477 non-null   float64
 8   HRSD            1477 non-null   float64
 9   floorsT         1477 non-null   int64
 10  floorsA         1477 non-null   float64
 11  floorsSD        1477 non-null   float64
 12  SuenoLigero     1477 non-null   int64
 13  SuenoProfundo   1477 non-null   int64
 14  Clase           1477 non-null   int64
dtypes: float64(9), int64(6)
memory usage: 173.2 KB
```

```
In [9]: df['Clase'].value_counts()
```

```
Out[9]: 1    765
        2    523
        3    189
        Name: Clase, dtype: int64
```

En el siguiente paso, preprocesamiento de datos, asignamos la variable “x” y “y” a la clase.

Y con la función oversample balanceamos las clases.

Preprocesamiento de datos

In [10]:
```
X = df.drop(columns=['Clase'])
Y = df['Clase']
```

In [11]:
```
#Balanceo de clases
oversample =SMOTE()
X, Y = oversample.fit_resample(X, Y)
Y.value_counts()
```

Out[11]:
```
1    765
2    765
3    765
Name: Clase, dtype: int64
```

División del dataset

In [12]:
```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.30)
```

Clasificación.

Continuando con el proceso, debemos de utilizar los modelos brindados en jupyter, para escoger el modelo optimo.

Decision Tree. Decision tree es el primer modelo de jupyter que utilizamos.

Clasificador

Decision Tree

In [13]:
```
#decision tree
model = DecisionTreeClassifier()
model.fit(x_train, y_train)
print("accurancy: ",model.score(x_test, y_test)*100)
```

```
accurancy:  56.313497822931794
```

In [14]:
```
scoring=['accuracy','precision_weighted','recall_weighted','f1_weighted']
result = cross_validate(model,X,Y,cv=10,scoring=scoring)
```

In [15]:
```
results = pd.DataFrame(result)
results.shape
```

Out[15]:
```
(10, 6)
```

En esta pestaña pedimos que muestre los resultados de 10, pero lo que necesitamos es otro dato así que en los otros solicitamos la accuracy, recall, precisión y FM.

```
In [16]: results.head(10)
```

Out[16]:

	fit_time	score_time	test_accuracy	test_precision_weighted	test_recall_weighted	test_f1_weighted
0	0.038660	0.011257	0.473913	0.490720	0.473913	0.473583
1	0.022529	0.010001	0.373913	0.386377	0.373913	0.377046
2	0.031123	0.007792	0.621739	0.639323	0.621739	0.626507
3	0.031513	0.010342	0.513043	0.512987	0.513043	0.510783
4	0.029001	0.008019	0.560870	0.551246	0.560870	0.553685
5	0.032202	0.008114	0.537118	0.523694	0.537118	0.527349
6	0.039818	0.014012	0.519651	0.510769	0.519651	0.505646
7	0.031672	0.006288	0.532751	0.533407	0.532751	0.518150
8	0.023808	0.008691	0.628821	0.650457	0.628821	0.619214
9	0.028638	0.007998	0.606987	0.607544	0.606987	0.601526

```
In [17]: results['test_accuracy'].mean()
```

```
Out[17]: 0.5368805771786596
```

```
In [18]: results['test_recall_weighted'].mean()
```

```
Out[18]: 0.5368805771786596
```

```
In [19]: results['test_precision_weighted'].mean()
```

```
Out[19]: 0.5406524351855075
```

```
In [20]: results['test_f1_weighted'].mean()
```

```
Out[20]: 0.531348935143637
```

Random Forest. Random Forest es el siguiente modelo de jupyter.

Random Forest

In [21]:
```
#decision tree
model = RandomForestClassifier()
model.fit(x_train, y_train)
scoring=['accuracy','precision_weighted','recall_weighted','f1_weighted']
result = cross_validate(model,X,Y,cv=10,scoring=scoring)
results = pd.DataFrame(result)
results.shape
```

Out[21]: (10, 6)

In [22]:
```
results['test_accuracy'].mean()
```

Out[22]: 0.622225175621796

In [23]:
```
results['test_recall_weighted'].mean()
```

Out[23]: 0.622225175621796

In [24]:
```
results['test_precision_weighted'].mean()
```

Out[24]: 0.6353308539103764

In [25]:
```
results['test_f1_weighted'].mean()
```

Out[25]: 0.6109398312899644

AdaBoost. AdaBoost es el siguiente modelo de jupyter.

AdaBoost

```
In [26]: model = AdaBoostClassifier()
         model.fit(x_train, y_train)
         scoring=['accuracy','precision_weighted','recall_weighted','f1_weighted']
         result = cross_validate(model,X,Y,cv=10,scoring=scoring)
         results = pd.DataFrame(result)
         results.shape
Out[26]: (10, 6)

In [27]: results['test_accuracy'].mean()
Out[27]: 0.45008353901651804

In [28]: results['test_recall_weighted'].mean()
Out[28]: 0.45008353901651804

In [29]: results['test_precision_weighted'].mean()
Out[29]: 0.454566851818501

In [30]: results['test_f1_weighted'].mean()
Out[30]: 0.4444765692121728
```

ExtraTrees. Extra Trees es el siguiente modelo de jupyter.

ExtraTrees

```
In [31]: model = ExtraTreesClassifier()
         model.fit(x_train, y_train)
         scoring=['accuracy','precision_weighted','recall_weighted','f1_weighted']
         result = cross_validate(model,X,Y,cv=10,scoring=scoring)
         results = pd.DataFrame(result)
         results.shape
Out[31]: (10, 6)

In [32]: results['test_accuracy'].mean()
Out[32]: 0.6370666413518132

In [33]: results['test_recall_weighted'].mean()
Out[33]: 0.6370666413518132

In [34]: results['test_precision_weighted'].mean()
Out[34]: 0.6446944889784526

In [35]: results['test_f1_weighted'].mean()
Out[35]: 0.6286747234522039
```

KNeighbors. KNeighbors es el siguiente modelo de jupyter.

KNeighbors

```
In [36]: model = KNeighborsClassifier()
         model.fit(x_train, y_train)
         scoring=['accuracy','precision_weighted','recall_weighted','f1_weighted']
         result = cross_validate(model,X,Y,cv=10,scoring=scoring)
         results = pd.DataFrame(result)
         results.shape
Out[36]: (10, 6)

In [37]: results['test_accuracy'].mean()
Out[37]: 0.547748243782039

In [38]: results['test_recall_weighted'].mean()
Out[38]: 0.547748243782039

In [39]: results['test_precision_weighted'].mean()
Out[39]: 0.5419250297948728

In [40]: results['test_f1_weighted'].mean()
Out[40]: 0.540811064847248
```

GaussinNB. GaussinNB es el siguiente modelo de jupyter.

GaussinNB

```
In [41]: model = GaussianNB()
         model.fit(x_train, y_train)
         scoring=['accuracy','precision_weighted','recall_weighted','f1_weighted']
         result = cross_validate(model,X,Y,cv=10,scoring=scoring)
         results = pd.DataFrame(result)
         results.shape
Out[41]: (10, 6)

In [42]: results['test_accuracy'].mean()
Out[42]: 0.36427757736852096

In [43]: results['test_recall_weighted'].mean()
Out[43]: 0.36427757736852096

In [44]: results['test_precision_weighted'].mean()
Out[44]: 0.39089002522144123

In [45]: results['test_f1_weighted'].mean()
Out[45]: 0.29786594655419496
```

Tabla completa.

Test02 Jupyter				
	Accuracy	**Precision**	**Recall**	**FM**
J48	0.563	0.540	0.536	0.531
RandomForest	0.622	0.635	0.622	0.610
AdaBoost	0.450	0.454	0.450	0.444
ExtraTrees	0.637	0.644	0.637	0.638
KNeighbors	0.547	0.541	0.547	0.540
NB	0.364	0.390	0.364	0.297

En esta prueba con jupyter, el mejor modelo fue el ExtraTrees.

Comparación de las 3 tablas.

Test02				
	Accuracy	Precision	Recall	FM
ZR	0.517	?	0.518	?
IBk	0.488	0.489	0.489	0.489
NB	0.371	0.416	0.371	0.259
SMO	0.516	?	0.517	?
J48	0.486	0.458	0.487	0.467
RF	0.507	0.445	0.508	0.461
AB-J48	0.482	0.465	0.494	0.476

Test02 SMOTE				
	Accuracy	Precision	Recall	FM
ZR	0.331	?	0.331	?
IBk	0.596	0.597	0.596	0.595
NB	0.338	0.363	0.339	0.27
SMO	0.365	0.381	0.365	0.336
J48	0.534	0.533	0.534	0.533
RF	0.623	0.621	0.623	0.621
AB-J48	0.594	0.596	0.595	0.595

Test02 Jupyter				
	Accuracy	Precision	Recall	FM
J48	0.563	0.540	0.536	0.531
RandomForest	0.622	0.635	0.622	0.610
AdaBoost	0.450	0.454	0.450	0.444
ExtraTrees	0.637	0.644	0.637	0.638
KNeighbors	0.547	0.541	0.547	0.540
NB	0.364	0.390	0.364	0.297

Al comparar las 3 tablas, logramos ver que Extratrees es el mejor de los modelos.

La razón por la cual ExtraTrees obtuvo un rendimiento más alto en comparación con los otros algoritmos puede deberse a varias razones. ExtraTrees es un algoritmo de ensamblaje (ensemble) que utiliza múltiples árboles de decisión y combina sus predicciones para obtener un resultado final. Esta técnica de ensamblaje puede mejorar la precisión y la generalización del modelo al reducir el sesgo y la varianza inherentes a los árboles de decisión individuales.

Además, es posible que las características y la estructura del conjunto de datos utilizado sean más adecuadas para el algoritmo ExtraTrees, lo que le permite capturar patrones y relaciones más complejas entre las variables y el nivel de estrés de los estudiantes. También es posible que los hiperparámetros del algoritmo, como el número de árboles y la profundidad máxima, estén configurados de manera óptima para este conjunto de datos en particular.

Conclusión.

En conclusión, al realizar una comparación exhaustiva de varios algoritmos de clasificación en nuestro estudio, hemos seleccionado ExtraTrees como el clasificador más adecuado para nuestro problema. Esta elección se basa en su alto rendimiento y precisión en todos los ámbitos evaluados.

ExtraTrees se destacó por su capacidad para manejar conjuntos de datos complejos y de gran tamaño, y logró una alta precisión en la clasificación de instancias desconocidas. Además, su enfoque en la aleatoriedad en la construcción de árboles de decisión y selección de umbrales de división proporcionó una mayor robustez y generalización en comparación con otros algoritmos.

Es importante destacar que la elección del clasificador puede variar dependiendo del problema específico y las características del conjunto de datos. Otros algoritmos también pueden ofrecer un rendimiento sólido en diferentes escenarios. Por lo tanto, es esencial realizar un análisis comparativo detallado y considerar diversos factores antes de seleccionar el clasificador más adecuado.

La elección de ExtraTrees como nuestro clasificador se basa en su capacidad para obtener resultados precisos y confiables en nuestro conjunto de datos. Estamos seguros de que esta elección nos brindará un sólido rendimiento en nuestras tareas de clasificación y nos permitirá obtener información valiosa de nuestros datos.

Sin embargo, es importante destacar que la selección del clasificador es solo uno de los pasos en el proceso de aprendizaje automático. La calidad de los datos, la selección de atributos relevantes, la validación cruzada y otros aspectos también desempeñan un papel crucial en el éxito de nuestro análisis. Por lo tanto, debemos considerar todos estos factores en conjunto para obtener resultados sólidos y confiables en nuestro proyecto de clasificación de datos.

En última instancia, estamos emocionados de aplicar ExtraTrees en nuestro análisis y confiamos en que este clasificador nos permitirá obtener conocimientos valiosos y tomar decisiones informadas basadas en nuestros datos.