



**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**

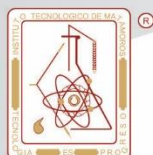
**Alex Fernández Sánchez**

**Adriel Eduardo Zayas Hernández**

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

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

**Excelencia en Educación Tecnológica®**  
**Tecnología es progreso®**



Introducción.	2
Marco conceptual.	3
Desarrollo.	5
Weka.	5
ZeroRules.	6
IBk.	6
J48.	7
RandomForest.	7
NB.	8
SMO.	8
AB J48.	9
Tabla completa.	9
Weka Classify con smote.	10
ZeroRules.	11
AB J48.	11
J48.	12
RandomForest.	12
IBK.	13
SMO.	13
NB.	14
Tabla completa.	14
Jupyter Notebook.	15
Clasificación.	18
Decision Tree.	18
Random Forest.	20
AdaBoost.	21
ExtraTrees.	22
KNeighbors.	23
GaussianNB.	24
Tabla completa.	24
Comparación de las 3 tablas.	25
Conclusión.	27

## **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 contruidos 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.

```
test02.csv: Bloc de notas
Archivo Edición Formato Ver Ayuda
relation test02

@attribute semestre numeric
@attribute StepsT numeric
@attribute StepsA numeric
@attribute StepsSD numeric
@attribute DistT numeric
@attribute DistA numeric
@attribute DistSD numeric
@attribute HRA numeric
@attribute HRSD numeric
@attribute floorsT numeric
@attribute floorsA numeric
@attribute floorsSD numeric
@attribute SuenoLigero numeric
@attribute SuenoProfundo numeric
@attribute Clase {1,2,3}

@data
8,651,10.85,18.2719,0.2044,0.0034,0.0081,88.53,7.913
6,9,0.15,1.1522,0.0061,0.0001,0.0008,79.93,4.6147,0,
6,42,0.7,2.3826,0.0196,0.0003,0.0013,90.65,4.6434,0,
8,378,6.3,11.1763,0.2731,0.0046,0.0081,86.05,5.9594,
8,32,0.5333,2.3977,0.0221,0.0004,0.0017,75.28,4.7045
8,1056,17.6,15.5951,0.1201,0.002,0.0041,99.85,5.609,
```

Aquí ya aplicado el cambio de numérico a nominal.

The screenshot shows the Weka Explorer interface. The 'Class' attribute is selected, and its distribution is displayed as a bar chart. The bar chart shows three categories: 1 (blue bar, count 765), 2 (red bar, count 523), and 3 (cyan bar, count 189). The 'Class' attribute is also listed in the 'Selected attribute' table.

No.	Label	Count	Weight
1	1	765	765
2	2	523	523
3	3	189	189

A continuación, empezaremos con la clasificación de modelos

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

**Classifier output**

```

=== Classifier model (full training set) ===

ZeroR predicts class value: 1

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances      765      51.7942 %
Incorrectly Classified Instances    712      48.2058 %
Kappa statistic                     0
Mean absolute error                 0.3934
Root mean squared error            0.4435
Relative absolute error             100 %
Root relative squared error         100 %
Total Number of Instances          1477

=== Detailed Accuracy By Class ===

```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1.000	1.000	0.518	1.000	0.682	?	0.496	0.516	1
	0.000	0.000	?	0.000	?	?	0.497	0.352	2
	0.000	0.000	?	0.000	?	?	0.497	0.127	3
Weighted Avg.	0.518	0.518	?	0.518	?	?	0.497	0.408	

**Confusion Matrix**

```

a b c <-- classified as
765 0 0 | a = 1
523 0 0 | b = 2
189 0 0 | c = 3

```

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

**Classifier output**

```

=== Classifier model (full training set) ===

IBk instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances      722      48.8829 %
Incorrectly Classified Instances    755      51.1171 %
Kappa statistic                     0.1337
Mean absolute error                 0.3408
Root mean squared error            0.5827
Relative absolute error             86.6189 %
Root relative squared error         131.4024 %
Total Number of Instances          1477

=== Detailed Accuracy By Class ===

```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.592	0.416	0.605	0.592	0.598	0.176	0.589	0.575	1
	0.453	0.328	0.431	0.453	0.442	0.124	0.562	0.394	2
	0.169	0.113	0.180	0.169	0.174	0.057	0.520	0.135	3
Weighted Avg.	0.489	0.346	0.489	0.489	0.489	0.142	0.570	0.454	

**Confusion Matrix**

```

a b c <-- classified as
453 236 76 | a = 1
216 237 20 | b = 2

```

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

**Classifier**  
Choose: **AdaBoostM1** -P 100 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2

**Test options**  
☐ Use training set  
☐ Supplied test set (Set...)  
☒ Cross-validation Folds: 10  
☐ Percentage split %: 66  
 More options...

**(Nom) Clase**  
 Start Stop  
 Result list (right-click for options):  
 19:37:47 - rules.ZeroR  
 19:38:28 - lazy.IBk  
 19:38:39 - trees.J48  
 19:38:46 - trees.RandomForest  
 19:38:59 - bayes.NaiveBayes  
 19:39:09 - functions.SMO  
 19:39:24 - meta.AdaBoostM1

**Classifier output**

Time taken to build model: 0.05 seconds

=== Stratified cross-validation ===  
 === Summary ===

Correctly Classified Instances	719	48.6798 %
Incorrectly Classified Instances	758	51.3202 %
Kappa statistic	0.0807	
Mean absolute error	0.3664	
Root mean squared error	0.4986	
Relative absolute error	93.1313 %	
Root relative squared error	112.4309 %	
Total Number of Instances	1477	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.668	0.569	0.558	0.668	0.608	0.102	0.575	0.577	1
	0.361	0.295	0.402	0.361	0.381	0.069	0.562	0.395	2
	0.101	0.056	0.209	0.101	0.136	0.062	0.537	0.151	3
Weighted Avg.	0.487	0.406	0.458	0.487	0.467	0.085	0.565	0.458	

=== Confusion Matrix ===

a	b	c	-- classified as
511	213	41	a = 1
303	189	31	b = 2
102	68	19	c = 3

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

**Classifier**  
Choose: **AdaBoostM1** -P 100 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2

**Test options**  
☐ Use training set  
☐ Supplied test set (Set...)  
☒ Cross-validation Folds: 10  
☐ Percentage split %: 66  
 More options...

**(Nom) Clase**  
 Start Stop  
 Result list (right-click for options):  
 19:37:47 - rules.ZeroR  
 19:38:28 - lazy.IBk  
 19:38:39 - trees.J48  
 19:38:46 - trees.RandomForest  
 19:38:59 - bayes.NaiveBayes  
 19:39:09 - functions.SMO  
 19:39:24 - meta.AdaBoostM1

**Classifier output**

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.72 seconds

=== Stratified cross-validation ===  
 === Summary ===

Correctly Classified Instances	750	50.7786 %
Incorrectly Classified Instances	727	49.2214 %
Kappa statistic	0.0733	
Mean absolute error	0.378	
Root mean squared error	0.445	
Relative absolute error	96.0731 %	
Root relative squared error	100.337 %	
Total Number of Instances	1477	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.761	0.680	0.546	0.761	0.636	0.090	0.588	0.611	1
	0.317	0.233	0.428	0.317	0.364	0.092	0.587	0.420	2
	0.011	0.016	0.087	0.011	0.019	-0.015	0.567	0.153	3
Weighted Avg.	0.508	0.437	0.445	0.508	0.461	0.077	0.585	0.485	

=== Confusion Matrix ===

a	b	c	-- classified as
582	171	12	a = 1
348	166	9	b = 2
136	51	2	c = 3



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

**Weka Explorer**

Preprocess **Classify** Cluster Associate Select attributes Visualize

Classifier: Choose **AdaBoostM1** -P 100 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2

Test options:  
☐ Use training set  
☐ Supplied test set Set...  
☒ Cross-validation Folds **10**  
☐ Percentage split % **66**  
 More options...

(Nom) Clase: **Start** **Stop**

Result list (right-click for options):  
 19:37:47 - rules.ZeroR  
 19:38:28 - lazy.IBk  
 19:38:39 - trees.J48  
 19:38:46 - trees.RandomForest  
 19:38:59 - bayes.NaiveBayes  
**19:39:09 - functions.SMO**  
 19:39:24 - meta.AdaBoostM1

Classifier output:

Time taken to build model: 0.16 seconds

=== Stratified cross-validation ===  
 === Summary ===

Correctly Classified Instances	763	51.6588 %
Incorrectly Classified Instances	714	48.3412 %
Kappa statistic	-0.0017	
Mean absolute error	0.3584	
Root mean squared error	0.4585	
Relative absolute error	91.091 %	
Root relative squared error	103.3937 %	
Total Number of Instances	1477	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.997	1.000	0.517	0.997	0.681	-0.036	0.499	0.517	1
	0.000	0.000	?	0.000	?	?	0.500	0.354	2
	0.000	0.002	0.000	0.000	0.000	-0.014	0.499	0.128	3
Weighted Avg.	0.517	0.518	?	0.517	?	?	0.499	0.410	

=== Confusion Matrix ===

a	b	c	<-- classified as
763	0	2	a = 1
523	0	0	b = 2
189	0	0	c = 3

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

**Weka Explorer**

Preprocess **Classify** Cluster Associate Select attributes Visualize

Classifier: Choose **AdaBoostM1** -P 100 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2

Test options:  
☐ Use training set  
☐ Supplied test set Set...  
☒ Cross-validation Folds **10**  
☐ Percentage split % **66**  
 More options...

(Nom) Clase: **Start** **Stop**

Result list (right-click for options):  
 19:37:47 - rules.ZeroR  
 19:38:28 - lazy.IBk  
 19:38:39 - trees.J48  
 19:38:46 - trees.RandomForest  
**19:38:59 - bayes.NaiveBayes**  
 19:39:09 - functions.SMO  
 19:39:24 - meta.AdaBoostM1

Classifier output:

Time taken to build model: 0 seconds

=== Stratified cross-validation ===  
 === Summary ===

Correctly Classified Instances	548	37.1022 %
Incorrectly Classified Instances	929	62.8978 %
Kappa statistic	0.0111	
Mean absolute error	0.4549	
Root mean squared error	0.5757	
Relative absolute error	115.6343 %	
Root relative squared error	129.8249 %	
Total Number of Instances	1477	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.084	0.072	0.557	0.084	0.145	0.022	0.523	0.550	1
	0.925	0.900	0.360	0.925	0.519	0.042	0.516	0.362	2
	0.000	0.015	0.000	0.000	0.000	-0.044	0.461	0.115	3
Weighted Avg.	0.371	0.358	0.416	0.371	0.259	0.021	0.513	0.428	

=== Confusion Matrix ===

a	b	c	<-- classified as
64	685	16	a = 1
36	484	3	b = 2
15	174	0	c = 3

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.

The screenshot shows the Weka Explorer interface with the 'Classify' tab selected. The classifier chosen is 'AdaBoostM1' with parameters '-P 100 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2'. The 'Test options' section shows 'Cross-validation' selected with 10 folds and a 66% split. The 'Result list' on the left shows several classifiers, with '19:39:24 - meta.AdaBoostM1' selected. The 'Classifier output' pane displays the following results:

```

Time taken to build model: 0.42 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      730           49.4245 %
Incorrectly Classified Instances    747           50.5755 %
Kappa statistic                    0.0991
Mean absolute error                 0.3395
Root mean squared error             0.5293
Relative absolute error             86.296 %
Root relative squared error         119.3449 %
Total Number of Instances          1477

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
          0.644    0.531    0.566     0.644    0.603     0.115    0.596     0.622     1
          0.428    0.318    0.425     0.428    0.427     0.110    0.600     0.437     2
          0.069    0.051    0.165     0.069    0.097     0.026    0.545     0.147     3
Weighted Avg.   0.494    0.394    0.465     0.494    0.476     0.102    0.591     0.496

=== Confusion Matrix ===

  a   b   c  <-- classified as
493 237  35 |   a = 1
268 224  31 |   b = 2
110  66  13 |   c = 3
  
```

## 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.

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Open file... Open URL... Open DB... Generate... Undo Edit... Save...

Filter: Choose **SMOTE -C 0 -K 5 -P 46.3 -S 1** Apply Stop

Current relation: test02-weka.filters.supervised.instance.SMOTE-C0-K... Attributes: 15  
Instances: 2295 Sum of weights: 2295

Attributes: All None Invert Pattern

No.	Name
1	<input type="checkbox"/> semestre
2	<input type="checkbox"/> StepsT
3	<input type="checkbox"/> StepsA
4	<input type="checkbox"/> StepsSD
5	<input type="checkbox"/> DistT
6	<input type="checkbox"/> DistA
7	<input type="checkbox"/> DistSD
8	<input type="checkbox"/> HRA
9	<input type="checkbox"/> HRSD
10	<input type="checkbox"/> floorsT
11	<input type="checkbox"/> floorsA
12	<input type="checkbox"/> floorsSD
13	<input type="checkbox"/> SuenoLigero
14	<input type="checkbox"/> SuenoProfundo
15	<input checked="" type="checkbox"/> Clase

Remove

Selected attribute: Name: Clase Missing: 0 (0%) Distinct: 3 Type: Nominal Unique: 0 (0%)

No.	Label	Count	Weight
1	1	765	765
2	2	765	765
3	3	765	765

Class: Clase (Nom) Visualize All

765 765 765

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

**Weka Explorer**

Preprocess **Classify** Cluster Associate Select attributes Visualize

Classifier: Choose **NaiveBayes**

Test options:  
☐ Use training set  
☐ Supplied test set (Set...)  
☒ Cross-validation Folds: 10  
☐ Percentage split %: 66  
 More options...

(Nom) Clase  
 Start Stop

Result list (right-click for options):  
 19:48:26 - rules.ZeroR  
 19:50:04 - meta.AdaBoostM1  
 19:51:05 - trees.J48  
 19:51:12 - trees.RandomForest  
 19:51:26 - lazy.IBk  
 19:51:33 - functions.SMO  
 19:51:45 - bayes.NaiveBayes

**Classifier output**

```

=== Classifier model (full training set) ===

ZeroR predicts class value: 1

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      760          33.1155 %
Incorrectly Classified Instances    1535          66.8845 %
Kappa statistic                    -0.0033
Mean absolute error                 0.4444
Root mean squared error             0.4714
Relative absolute error              100 %
Root relative squared error          100 %
Total Number of Instances          2295

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
          0.497    0.500    0.332     0.497    0.398     -0.003   0.498    0.333     1
          0.497    0.503    0.330     0.497    0.397     -0.006   0.497    0.332     2
          0.000    0.000    ?         0.000    ?         ?        0.498    0.333     3
Weighted Avg.   0.331    0.334    ?         0.331    ?         ?        0.498    0.332

=== Confusion Matrix ===

  a  b  c  <-- classified as
380 385  0 |  a = 1
385 380  0 |  b = 2
380 385  0 |  c = 3
  
```

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

**Weka Explorer**

Preprocess **Classify** Cluster Associate Select attributes Visualize

Classifier: Choose **NaiveBayes**

Test options:  
☐ Use training set  
☐ Supplied test set (Set...)  
☒ Cross-validation Folds: 10  
☐ Percentage split %: 66  
 More options...

(Nom) Clase  
 Start Stop

Result list (right-click for options):  
 19:48:26 - rules.ZeroR  
 19:50:04 - meta.AdaBoostM1  
 19:51:05 - trees.J48  
 19:51:12 - trees.RandomForest  
 19:51:26 - lazy.IBk  
 19:51:33 - functions.SMO  
 19:51:45 - bayes.NaiveBayes

**Classifier output**

```

Time taken to build model: 0.86 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      1365          59.4771 %
Incorrectly Classified Instances     930          40.5229 %
Kappa statistic                    0.3922
Mean absolute error                 0.28
Root mean squared error             0.4787
Relative absolute error              62.9962 %
Root relative squared error          101.5383 %
Total Number of Instances          2295

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
          0.625    0.241    0.565     0.625    0.593     0.376   0.766    0.595     1
          0.477    0.232    0.507     0.477    0.492     0.249   0.671    0.502     2
          0.682    0.135    0.716     0.682    0.699     0.554   0.850    0.735     3
Weighted Avg.   0.595    0.203    0.596     0.595    0.595     0.393   0.762    0.611

=== Confusion Matrix ===

  a  b  c  <-- classified as
478 221  66 |  a = 1
259 365 141 |  b = 2
109 134 522 |  c = 3
  
```

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

The screenshot shows the Weka Explorer interface with the 'Classify' tab selected. The classifier chosen is 'NaiveBayes'. The test options are set to 'Cross-validation' with 10 folds and a 66% percentage split. The result list on the left shows several models, with '19:51:05 - trees.J48' selected. The classifier output on the right displays the following information:

Time taken to build model: 0.05 seconds

=== Stratified cross-validation ===

=== Summary ===

Metric	Value	Percentage
Correctly Classified Instances	1226	53.4205 %
Incorrectly Classified Instances	1069	46.5795 %
Kappa statistic	0.3013	
Mean absolute error	0.3321	
Root mean squared error	0.4909	
Relative absolute error	74.7148 %	
Root relative squared error	104.136 %	
Total Number of Instances	2295	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.549	0.262	0.512	0.549	0.530	0.282	0.694	0.482	1
	0.408	0.264	0.436	0.408	0.421	0.146	0.607	0.412	2
	0.646	0.173	0.652	0.646	0.649	0.474	0.755	0.571	3
Weighted Avg.	0.534	0.233	0.533	0.534	0.533	0.301	0.685	0.488	

=== Confusion Matrix ===

a	b	c	<-- classified as
420	259	86	a = 1
275	312	178	b = 2
126	145	494	c = 3

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

The screenshot shows the Weka Explorer interface with the 'Classify' tab selected. The classifier chosen is 'NaiveBayes'. The test options are set to 'Cross-validation' with 10 folds and a 66% percentage split. The result list on the left shows several models, with '19:51:12 - trees.RandomForest' selected. The classifier output on the right displays the following information:

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.94 seconds

=== Stratified cross-validation ===

=== Summary ===

Metric	Value	Percentage
Correctly Classified Instances	1430	62.3094 %
Incorrectly Classified Instances	865	37.6906 %
Kappa statistic	0.4346	
Mean absolute error	0.3415	
Root mean squared error	0.4038	
Relative absolute error	76.8475 %	
Root relative squared error	85.6489 %	
Total Number of Instances	2295	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.607	0.226	0.573	0.607	0.589	0.375	0.788	0.633	1
	0.505	0.205	0.552	0.505	0.527	0.307	0.736	0.562	2
	0.758	0.135	0.738	0.758	0.748	0.619	0.891	0.810	3
Weighted Avg.	0.623	0.188	0.621	0.623	0.621	0.434	0.805	0.668	

=== Confusion Matrix ===

a	b	c	<-- classified as
464	226	75	a = 1
248	386	131	b = 2
98	87	580	c = 3

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

**Classifier**  
Choose **NaiveBayes**

**Test options**  
☐ Use training set  
☐ Supplied test set Set...  
☒ Cross-validation Folds **10**  
☐ Percentage split % **66**  
 More options...

**(Nom) Clase**  
 Start Stop

**Result list (right-click for options)**  
 19:48:26 - rules.ZeroR  
 19:50:04 - meta.AdaBoostM1  
 19:51:05 - trees.J48  
 19:51:12 - trees.RandomForest  
**19:51:26 - lazy.IBk**  
 19:51:33 - functions.SMO  
 19:51:45 - bayes.NaiveBayes

**Classifier output**

Time taken to build model: 0 seconds

=== Stratified cross-validation ===  
 === Summary ===

Correctly Classified Instances	1368	59.6078 %
Incorrectly Classified Instances	927	40.3922 %
Kappa statistic	0.3941	
Mean absolute error	0.2691	
Root mean squared error	0.5178	
Relative absolute error	60.5473 %	
Root relative squared error	109.8371 %	
Total Number of Instances	2295	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.523	0.176	0.597	0.523	0.557	0.359	0.669	0.481	1
	0.587	0.246	0.544	0.587	0.565	0.335	0.666	0.464	2
	0.678	0.184	0.649	0.678	0.663	0.489	0.747	0.562	3
Weighted Avg.	0.596	0.202	0.597	0.596	0.595	0.395	0.694	0.502	

=== Confusion Matrix ===

a	b	c	<-- classified as
400	232	133	a = 1
168	449	148	b = 2
102	144	519	c = 3

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

**Classifier**  
Choose **NaiveBayes**

**Test options**  
☐ Use training set  
☐ Supplied test set Set...  
☒ Cross-validation Folds **10**  
☐ Percentage split % **66**  
 More options...

**(Nom) Clase**  
 Start Stop

**Result list (right-click for options)**  
 19:48:26 - rules.ZeroR  
 19:50:04 - meta.AdaBoostM1  
 19:51:05 - trees.J48  
 19:51:12 - trees.RandomForest  
 19:51:26 - lazy.IBk  
**19:51:33 - functions.SMO**  
 19:51:45 - bayes.NaiveBayes

**Classifier output**

Time taken to build model: 0.25 seconds

=== Stratified cross-validation ===  
 === Summary ===

Correctly Classified Instances	838	36.5142 %
Incorrectly Classified Instances	1457	63.4858 %
Kappa statistic	0.0477	
Mean absolute error	0.4297	
Root mean squared error	0.5305	
Relative absolute error	96.6882 %	
Root relative squared error	112.546 %	
Total Number of Instances	2295	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.176	0.110	0.446	0.176	0.253	0.093	0.546	0.365	1
	0.667	0.594	0.359	0.667	0.467	0.070	0.537	0.351	2
	0.252	0.248	0.337	0.252	0.288	0.004	0.508	0.337	3
Weighted Avg.	0.365	0.317	0.381	0.365	0.336	0.056	0.530	0.351	

=== Confusion Matrix ===

a	b	c	<-- classified as
135	441	189	a = 1
64	510	191	b = 2
104	468	193	c = 3

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

**Classifier output**

Time taken to build model: 0 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	777	33.8562 %
Incorrectly Classified Instances	1518	66.1438 %
Kappa statistic	0.0078	
Mean absolute error	0.438	
Root mean squared error	0.5554	
Relative absolute error	98.5408 %	
Root relative squared error	117.812 %	
Total Number of Instances	2295	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.081	0.048	0.456	0.081	0.138	0.065	0.531	0.379	1
	0.149	0.179	0.294	0.149	0.198	-0.038	0.506	0.327	2
	0.786	0.765	0.339	0.786	0.474	0.023	0.531	0.351	3
Weighted Avg.	0.339	0.331	0.363	0.339	0.270	0.017	0.522	0.352	

=== Confusion Matrix ===

a	b	c	<-- classified as
62	140	563	a = 1
44	114	607	b = 2
30	134	601	c = 3

Tabla completa.

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

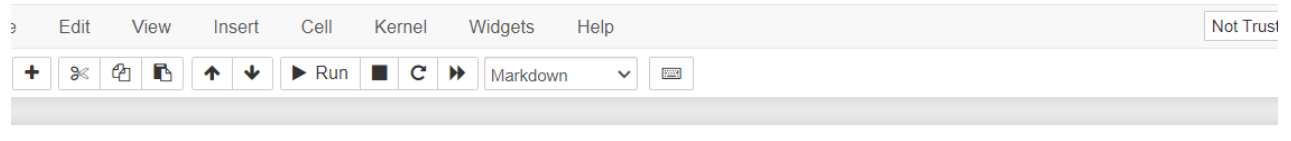
En esta prueba con SMOTE, el mejor modelo fue el RandomForest, al balancear las clases pudimos observar que SMO logro registrar datos en el precisión y FM. Además de que todos los modelos mejoraron significativamente su porcentaje en todos los parámetros.

## 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 librerías que utilizamos junto con los clasificadores.

jupyter Proyecto-Final (autosaved)



### Proyecto Final

```
In [1]: #Importar Librerías
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	floorsSD
count	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000	1477.000000
mean	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
min	6.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	50.670000	0.000000	0.000000	0.000000	0.000
25%	6.000000	43.000000	0.716700	2.933500	0.018100	0.000300	0.001300	76.570000	4.561800	0.000000	0.000000	0.000
50%	8.000000	159.000000	2.633300	7.321400	0.087600	0.001400	0.004400	82.680000	6.161200	0.000000	0.000000	0.000
75%	8.000000	374.000000	6.200000	12.531100	0.233300	0.003800	0.008200	90.130000	8.173300	0.000000	0.000000	0.000
max	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("accuracy: ", model.score(x_test, y_test)*100)

         accuracy: 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



GaussianNB. GaussianNB es el siguiente modelo de jupyter.

GaussianNB

```
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.