En el siguiente informe presentamos los resultados obtenidos en los datasets de aborto balanceado y no balanceado usando los clasificadores SVM, Random Forest y Red Neuronal.

En la exploración manual de los parámetros los mejores resultados obtenidos fueron con Redes Neuronales en el dataset de aborto no balanceado, en donde obtuvimos los siguientes resultados:

Accuracy: 0,64

F1 micro: 0.64

F1 macro: 0.57

F1 average: 0.62

Si bien en SVM obtuvimos un accuracy de 0,69, no creemos que este sea el mejor clasificador ya que las métricas f1 macro y average caen a 0,35 y 0,41 respectivamente.

Al igual que con los clasificadores usados en el práctico anterior, el Grid Search arrojó peores resultados que la exploración manual. Esta forma de optimizar los hiper parámetros se usó solo para Redes Neuronales.

**SUPPORT VECTOR MACHINES**

Es un algoritmo que busca separar los datos mediante la mejor frontera de decisión (hiperplano). Cuando decimos “mejor”, nos referimos a que esté lo más separada posible de los puntos más cercanos a ella. Los puntos más cercanos son conocidos como “vectores de soporte”, mientras que el espacio entre ellos y el hiperplano se conoce como “margen”.

Este clasificador también es conocido como “clasificador por márgenes” (*margin classifier*).

* **Dataset Aborto No Balanceado**

En la exploración manual de parámetros, partimos con los siguientes valores:

**C = 1.0**

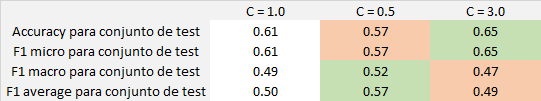
**kernel = "rbf"**

**gamma = 0.1**

**tol = 1e-3**

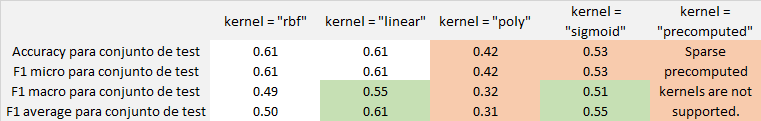
**class\_weight = "balanced"**

Al variar el valor de C, vemos que con un C menor a 1 (C = 0.5) caen las métricas de Accuracy y F1 micro, mientras que suben F1 macro y F1 average. Con un C mayor a 1 (C = 3.0) ocurre lo contrario, suben Accuracy y F1 micro y caen F1 macro y F1 average.

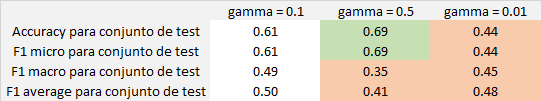


Al variar el valor del kernel, se observa que el peor resultado se produce al hacer kernel = ”poly” ya que caen todas las métricas. Otro aspecto a destacar es que el algoritmo no funciona cuando se fija kernel = “precomputed”.

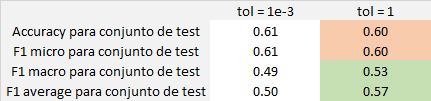
También se probó con kernel = “linear” y kernel = “sigmoid”. En el primer caso no se alteran las métricas Accuracy y F1 micro y mejoran F1 macro y F1 average. En el segundo caso empeoran las métricas Accuracy y F1 micro, mientras que mejoran F1 macro y F1 average.



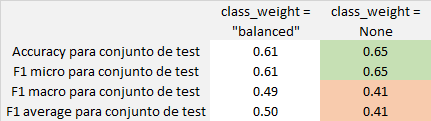
Al variar gamma, vemos que con un valor menor (gamma = 0.01) empeoran todas las métricas. Al fijar un gamma mayor (gamma = 0.5), mejoran las métricas Accuracy y F1 micro, mientras que disminuyen el valor del F1 macro y F1 average.



Al aumentar el valor de tol, se observa una caída en las métricas Accuracy y F1 micro y un aumento en F1 macro y F1 average.



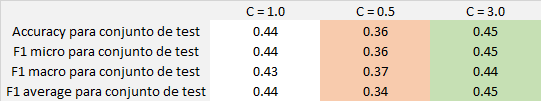
Cuando variamos el parámetro class\_weight, fijando class\_weight = None, mejoran las métricas Accuracy y F1 micro pero empeoran F1 macro y F1 average.



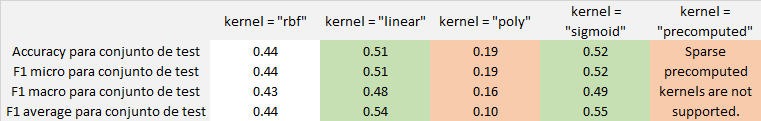
* **Dataset Aborto Balanceado**

Comenzamos la exploración manual con los mismos parámetros que usamos en el dataset de Aborto No Balanceado. La particularidad que notamos en este dataset es que al variar los parámetros, todas las métricas se movían en el mismo sentido. Creemos que esto tiene que ver con que el dataset no presenta desequilibrios de clase, por lo que al variar los parámetros afecta de igual manera a las métricas.

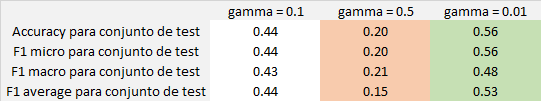
En el caso de C, vemos que al disminuir su valor las métricas empeoran. En tanto, cuando incrementamos su valor las métricas mejoran.



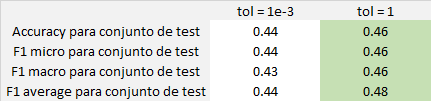
Respecto al parámetro kernel, vemos que las métricas mejoran cuando lo fijamos en “linear” y “sigmoid”, mientras que empeoran cuando hacemos kernel = ”poly”.



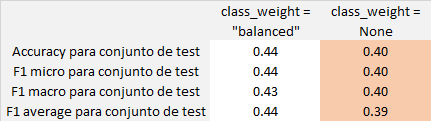
Cuando alteramos el valor de gamma, observamos que un valor por encima de 0.1 hace caer las métricas mientras que un valor por debajo de 0.1 las mejora.



Al aumentar el valor de tol, se observa una mejora en las métricas.



Cuando variamos el parámetro class\_weight, fijando class\_weight = None, empeoran las métricas.



**RANDOM FOREST**

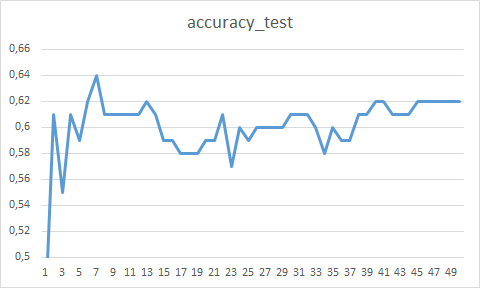
Los random forests simplifican los modelos de árboles de decisión. Cuando se crean árboles de decisión todas las features son consideradas al crear cada nodo. Esto es computacionalmente costoso. Para los random forests en cada nodo se consideran solo M atributos elegidos aleatoriamente (parámetro *max\_features* en sklearn). Por default, el parámetro *max\_features* = ‘auto’ , lo que significa que el algoritmo utiliza la raíz cuadrada del número de atributos.

* **Dataset Aborto No Balanceado**

En la exploración manual de parámetros, partimos con el siguiente valor:

**n\_estimators = 10**

A continuación hay un gráfico en el que podemos ver el valor accuracy para el conjunto de test tomando n\_estimator de 1 a 50. Vemos que el mejor valor se obtiene en n=7 con un accuracy de 0,64 (nos fijamos en f1 micro). El resto de los valores varían entre 0,58 y 0,62 con algunas excepciones (n = 1, 3, 23). Observamos el mismo comportamiento para valores de n\_estimator más altos.

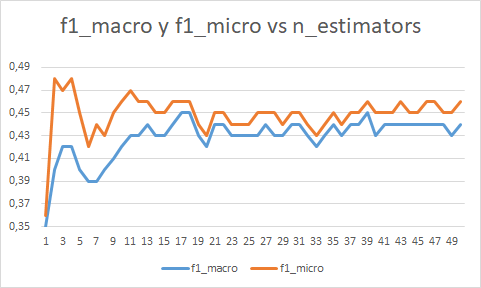


Por otro lado, al incorporar nuevos parámetros observamos que el mejor resultado se seguía dando con los valores por default. Los parámetros incorporados fueron: criterion, min\_samples\_split, min\_samples\_leaf, max\_features.

* **Dataset Aborto Balanceado**

En el siguiente gráfico podemos observar de qué manera cambian los accuracy f1 macro y f1 micro con n\_estimator que varía de 1 a 50. Nuevamente, nos quedamos con estos primeros 50 ya que para números más grandes, el comportamiento es similar.

El mayor valor se obtiene para n\_estimator = 2 con un f1 micro de 0,48. Sin embargo, en ese caso el f1 macro es de 0,4. Si observamos en n = 39 vemos que el f1 micro es de 0,46 y el f1 macro de 0,45. Por lo tanto, creemos que este es un mejor parámetro porque si bien el f1 micro da peor, la diferencia es pequeña y la mejora de f1 macro es grande.



Al modificar el parámetro criterion de “gini” a “entropy” vemos que los resultados se mantienen en los mismos valores. Encontramos una mejora al variar el parámetro min\_sample\_split. Cuando este parámetro es igual a 7, el f1 micro nos da 0,47 y el f1 macro nos da 0,46. En el caso del parámetro min\_samples\_leaf, el valor de default es 1 y al aumentarlo, los resultados de ambas métricas empeoran. Lo mismo sucede cuando variamos el parámetro max\_features de “auto” a “log2”, mientras que con “sqrt” no cambian los resultados.

Por lo tanto los mejores resultados se obtuvieron con los siguientes valores para los parámetros:

* n\_estimators = 39
* criterion = "gini"
* min\_samples\_split = 7
* min\_samples\_leaf = 1
* max\_features = "auto"

**REDES NEURONALES**

* **Dataset Aborto NO Balanceado**

Se realizó un primer entrenamiento con 1000 epoch y batch size = 1, observamos que aproximadamente luego del epoch 50 comienza a descender el accuracy y a aumentar el valor de loss, obteniendo las siguientes métricas:

*Accuracy para conjunto de test: 0.64*

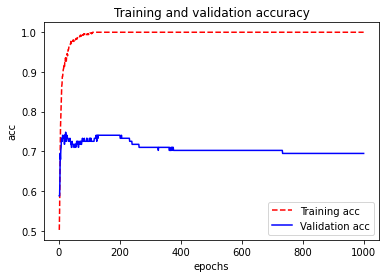
*F1 micro para conjunto de test: 0.64*

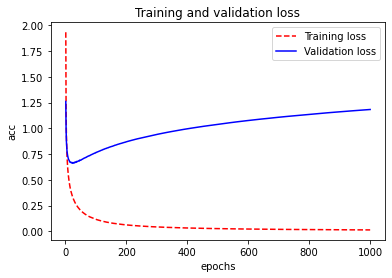
*F1 macro para conjunto de test: 0.57*

*F1 average para conjunto de test: 0.62*

*Exactitud del algoritmo para conjunto de test: 0.64*

En las siguientes gráficas podemos observar que a partir de 30 epochs, comienza a disminuir el valor de accuracy y aumentar el valor de loss en el conjunto de validación.





Decidimos realizar un segundo entrenamiento pero solo con 50 epochs, obteniendo las siguientes métricas:

*Accuracy para conjunto de test: 0.64*

*F1 micro para conjunto de test: 0.64*

*F1 macro para conjunto de test: 0.54*

*F1 average para conjunto de test: 0.64*

*Exactitud del algoritmo para conjunto de test: 0.64*

**Para GridSearchCV utilizamos 50 epochs y batch\_size = 1 para ver las diferencias entre funciones de activación.**

***Best: 0.647399 using {'activation': 'softmax'}***

**Otros resultados:**

*0.647399 (0.031072) with: {'activation': 'softmax'}*

*0.628407 (0.037785) with: {'activation': 'softplus'}*

*0.365201 (0.191502) with: {'activation': 'softsign'}*

*0.346154 (0.150780) with: {'activation': 'relu'}*

*0.291282 (0.175276) with: {'activation': 'tanh'}*

*0.534414 (0.054128) with: {'activation': 'sigmoid'}*

*0.385000 (0.089372) with: {'activation': 'hard\_sigmoid'}*

*0.181300 (0.180304) with: {'activation': 'linear'}*

**Obtenemos el mayor valor de accuracy para la función de activación Softmax.**

Métricas obtenidas para el set de test:

*Accuracy para conjunto de test: 0.61*

*F1 micro para conjunto de test: 0.61*

*F1 macro para conjunto de test: 0.52*

*F1 average para conjunto de test: 0.62*

*Exactitud del algoritmo para conjunto de test: 0.61*

**VOTING**

clf1 = RandomForestClassifier(random\_state=2, n\_estimators=7, criterion='entropy')

clf2 = SVC(C = 3.0, kernel = "linear", gamma=0.5, tol = 1)

clf3 = KerasClassifier(build\_fn=create\_model, epochs=50, batch\_size=1, verbose=0)

**Utilizando los 3 clasificadores anteriores, obtenemos las siguientes métricas para Voting.**

*Accuracy para conjunto de test: 0.60*

*F1 micro para conjunto de test: 0.60*

*F1 macro para conjunto de test: 0.54*

*F1 average para conjunto de test: 0.62*

*Exactitud del algoritmo para conjunto de test: 0.60*

* **Dataset Aborto Balanceado**

Se realizó un primer entrenamiento con 1000 epoch y batch size = 1, observamos que aproximadamente luego del epoch 50 comienza a descender el accuracy y a aumentar el valor de loss, obteniendo las siguientes métricas**:**

*Accuracy para conjunto de test: 0.51*

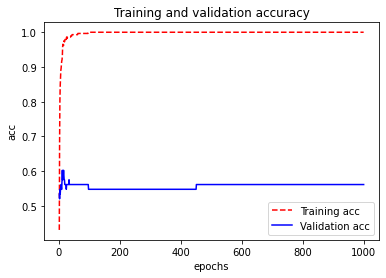
*F1 micro para conjunto de test: 0.51*

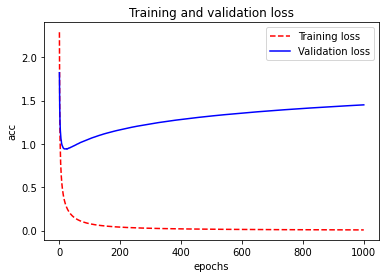
*F1 macro para conjunto de test: 0.48*

*F1 average para conjunto de test: 0.55*

*Exactitud del algoritmo para conjunto de test: 0.51*

En las siguientes gráficas podemos observar que a partir de 20 epochs, comienza a disminuir el valor de accuracy y aumentar el valor de loss en el conjunto de validación.





Decidimos realizar un segundo entrenamiento pero solo con 50 epochs, obteniendo las siguientes métricas.

*Accuracy para conjunto de test: 0.50*

*F1 micro para conjunto de test: 0.50*

*F1 macro para conjunto de test: 0.47*

*F1 average para conjunto de test: 0.53*

*Exactitud del algoritmo para conjunto de test: 0.50*

**Para GridSearchCV utilizamos 50 epochs y batch\_size = 1 para ver las diferencias entre funciones de activación.**

***Best: 0.672414 using {'activation': 'softmax'}***

**Otros resultados:**

*0.672414 (0.043618) with: {'activation': 'softmax'}*

*0.651724 (0.044159) with: {'activation': 'softplus'}*

*0.289655 (0.159294) with: {'activation': 'softsign'}*

*0.382759 (0.085445) with: {'activation': 'relu'}*

*0.227586 (0.156660) with: {'activation': 'tanh'}*

*0.575862 (0.041666) with: {'activation': 'sigmoid'}*

*0.637931 (0.044960) with: {'activation': 'hard\_sigmoid'}*

*0.251724 (0.145728) with: {'activation': 'linear'}*

**Obtenemos el mayor valor de accuracy para la función de activación Softmax.**

Métricas obtenidas con el set de test:

*Accuracy para conjunto de test: 0.52*

*F1 micro para conjunto de test: 0.52*

*F1 macro para conjunto de test: 0.49*

*F1 average para conjunto de test: 0.55*

*Exactitud del algoritmo para conjunto de test: 0.52*

**VOTING**

clf1 = RandomForestClassifier(random\_state=2, n\_estimators=39, min\_samples\_split=7)

clf2 = SVC(C = 3.0, kernel = "linear", gamma=0.01, tol = 1)

clf3 = KerasClassifier(build\_fn=create\_model, epochs=50, batch\_size=1, verbose=0)

**Utilizando los 3 clasificadores anteriores, obtenemos las siguientes métricas para Voting.**

*Accuracy para conjunto de test: 0.50*

*F1 micro para conjunto de test: 0.50*

*F1 macro para conjunto de test: 0.49*

*F1 average para conjunto de test: 0.55*

*Exactitud del algoritmo para conjunto de test: 0.50*

COMENTARIOS