Aprendizaje Automático Profundo

Clase 5 - 2019

Profs: Franco Ronchetti - Facundo Quiroga

EVALUACIÓN DE MODELOS

Etapas en la creación de un modelo de ML

Datos

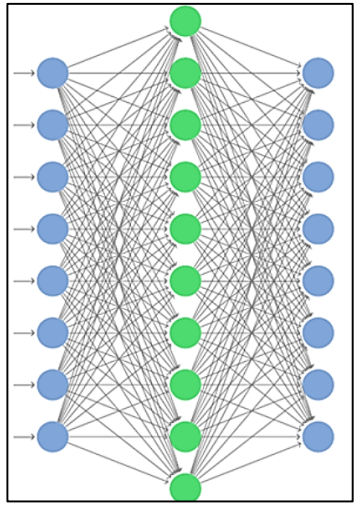
1	2	3	C
0,5	0,8	0,1	0,1
0,6	0,2	0,4	0,9
0,6	0,7	0,3	0,2
0,2	0,1	0,2	0,3
0,6	0,5	0,9	0,3

Preprocesamiento

1	2	3	C
0,5	0,8	0,1	0,1
0,6	0,2	0,4	0,9
0,6	0,7	0,3	0,2
0,2	0,1	0,2	0,3
0,6	0,5	0,9	0,3

Entrenamiento

Genera



Modelo

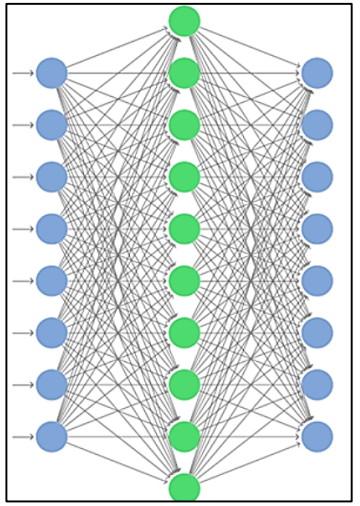
Training / Testing

Datos

1	2	3	C
0,5	0,8	0,1	0,1
0,6	0,2	0,4	0,9
0,6	0,7	0,3	0,2
0,2	0,1	0,2	0,3
0,6	0,5	0,9	0,3

Entrenamiento

Genera

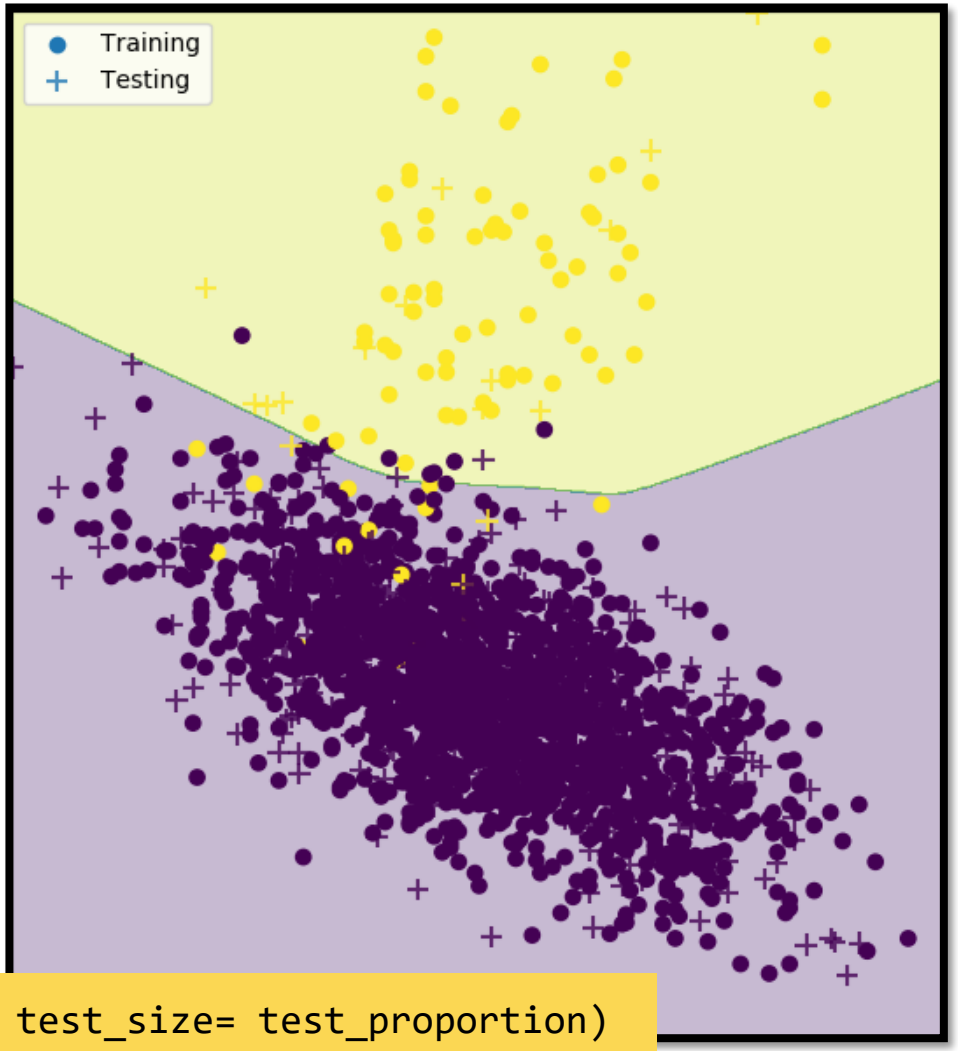


Modelo

Evaluar

Training / Testing

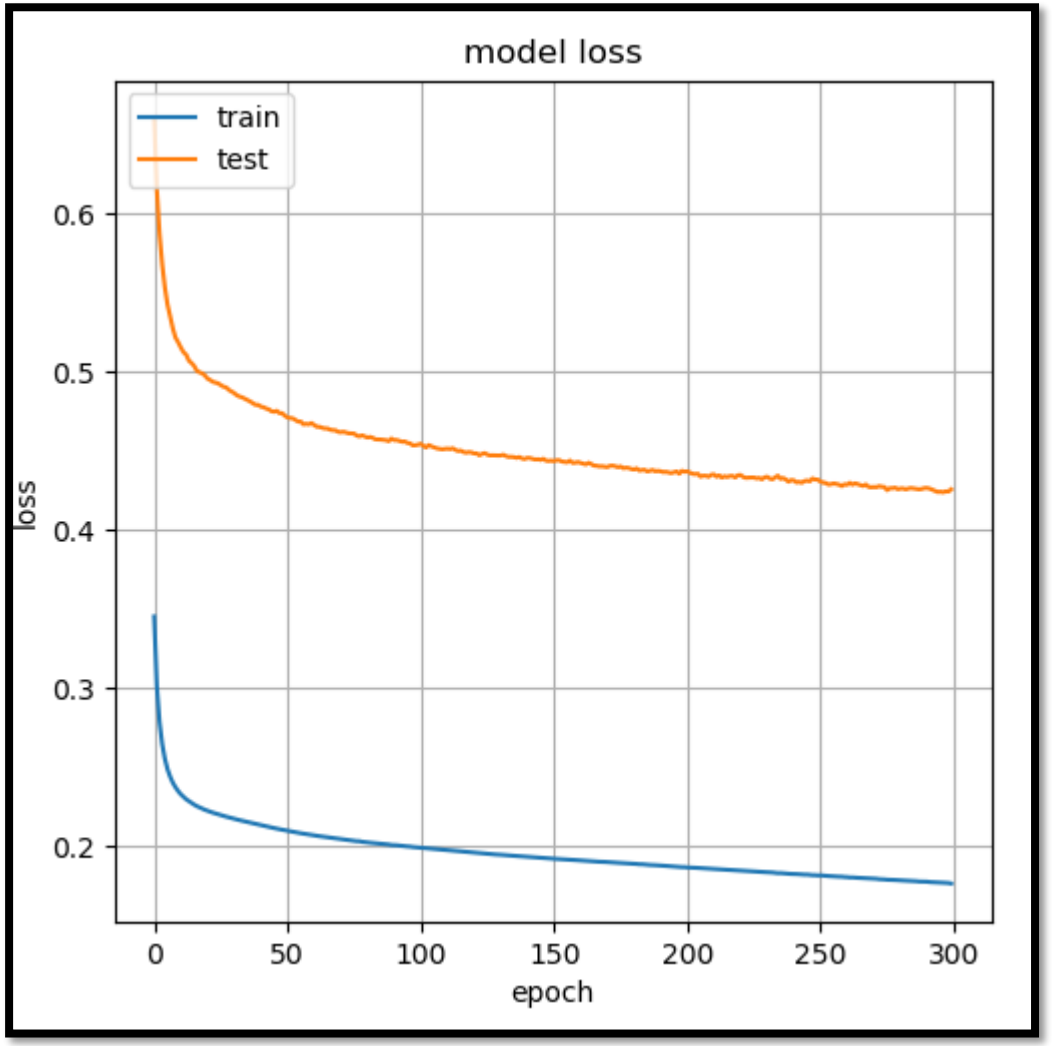
- El conjunto de entrenamiento (training) lo utilizamos para entrenar un modelo.
- El conjunto de evaluación (testing) lo utilizamos para verificar cómo funciona nuestro modelo con datos que nunca vio.
- Generalmente se utiliza un porcentaje mayor de muestras para el entrenamiento. Una configuración típica sería 90%-10%, o 80%-20%.
- Existen diferentes formas de obtener el *test set*. Lo más usual es tomar una porción aleatoria del *dataset* aunque las bases de datos más robustas ya tienen sus conjuntos separados.




```
X_train, X_test, Y_train, Y_test = AAPutils.dividir_train_test(X, Y, test_size= test_proportion)
AAPutils.plot_frontera_de_decision_2D(model, X_train, Y_train, x_test=X_test, y_test=Y_test)
```

Curvas de entrenamiento

- Las curvas de entrenamiento nos permiten ver cómo se comporta el modelo (error) en cada iteración.
- No siempre la última iteración será el mejor modelo.
- La diferencia entre la curva de training y de testing nos permite visualizar el comportamiento del modelo con nuevos datos.

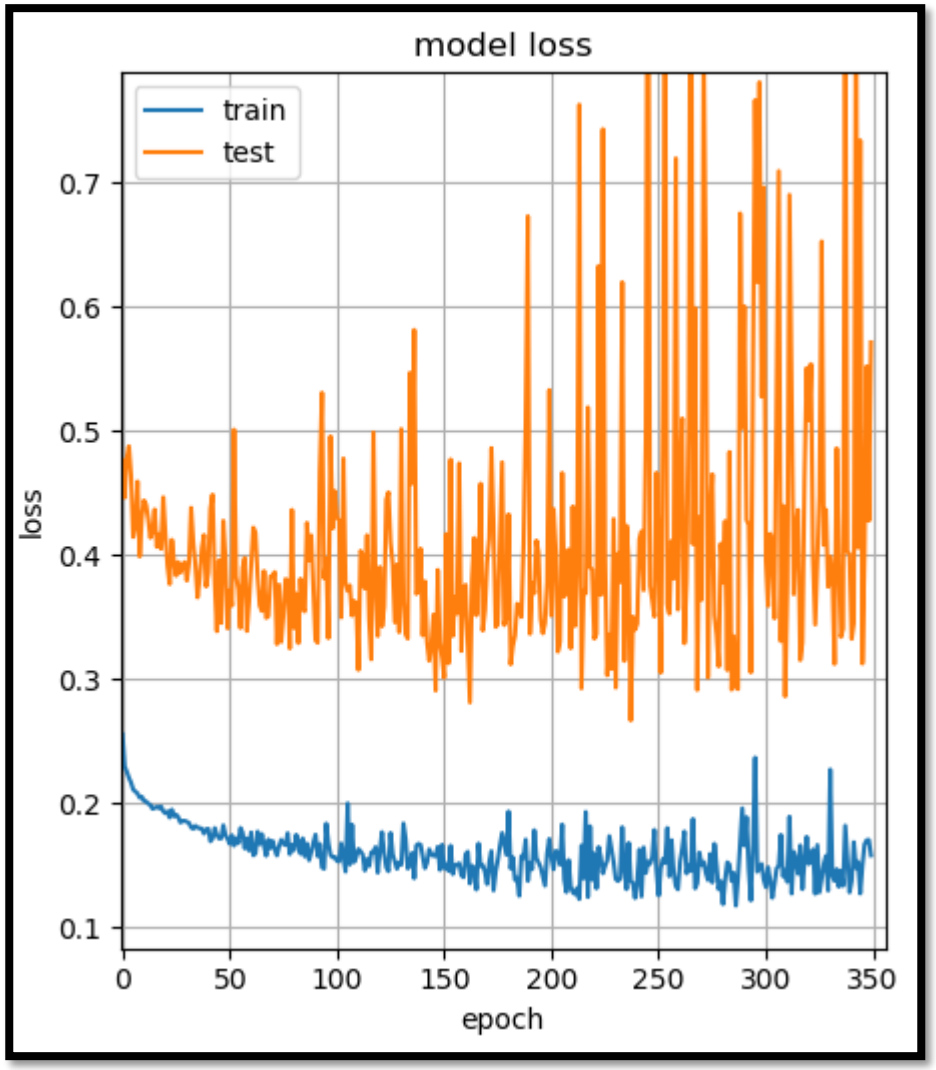


Curvas de entrenamiento

- También nos permiten visualizar errores en la configuración de los hiperparámetros. Por ejemplo **¿Qué ocurre aquí?**

Alfa demasiado grande!

```
history= model.fit(  X_train, Y_train,
                     epochs= EPOCAS,
                     batch_size= LOTE,
                     validation_data=(X_test, Y_test))
AAPutils.plot_training_curves(history)
```



Métricas para clasificación binaria

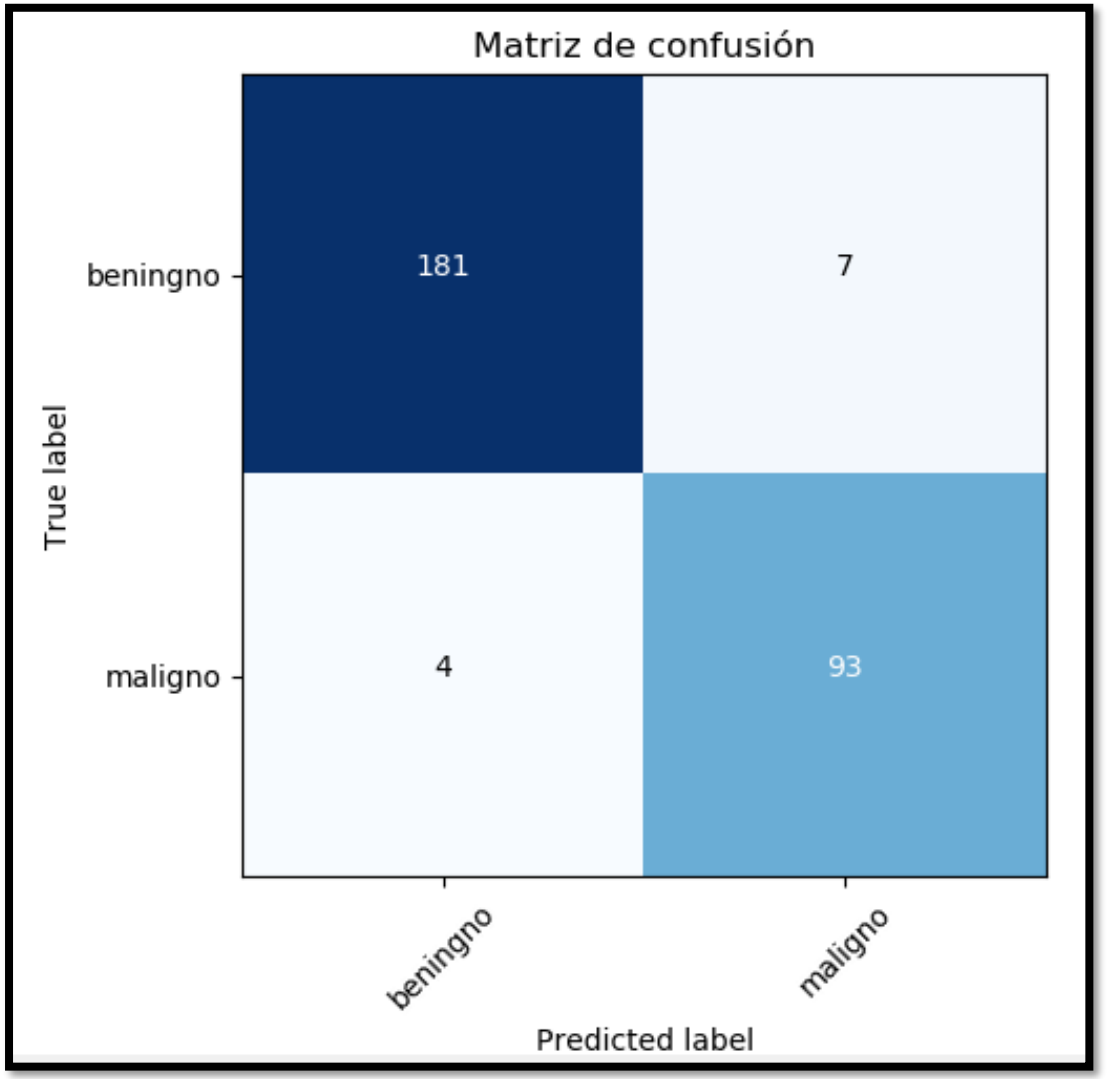
- El error (entropía cruzada) nos permite saber cómo se ajustan las probabilidades del modelo pero no nos dice nada sobre el resultado final de la clasificación.

- Veremos a continuación una serie de métricas que nos permitirán observar con más detalle este comportamiento.

Matriz de confusión (clasificación binaria)

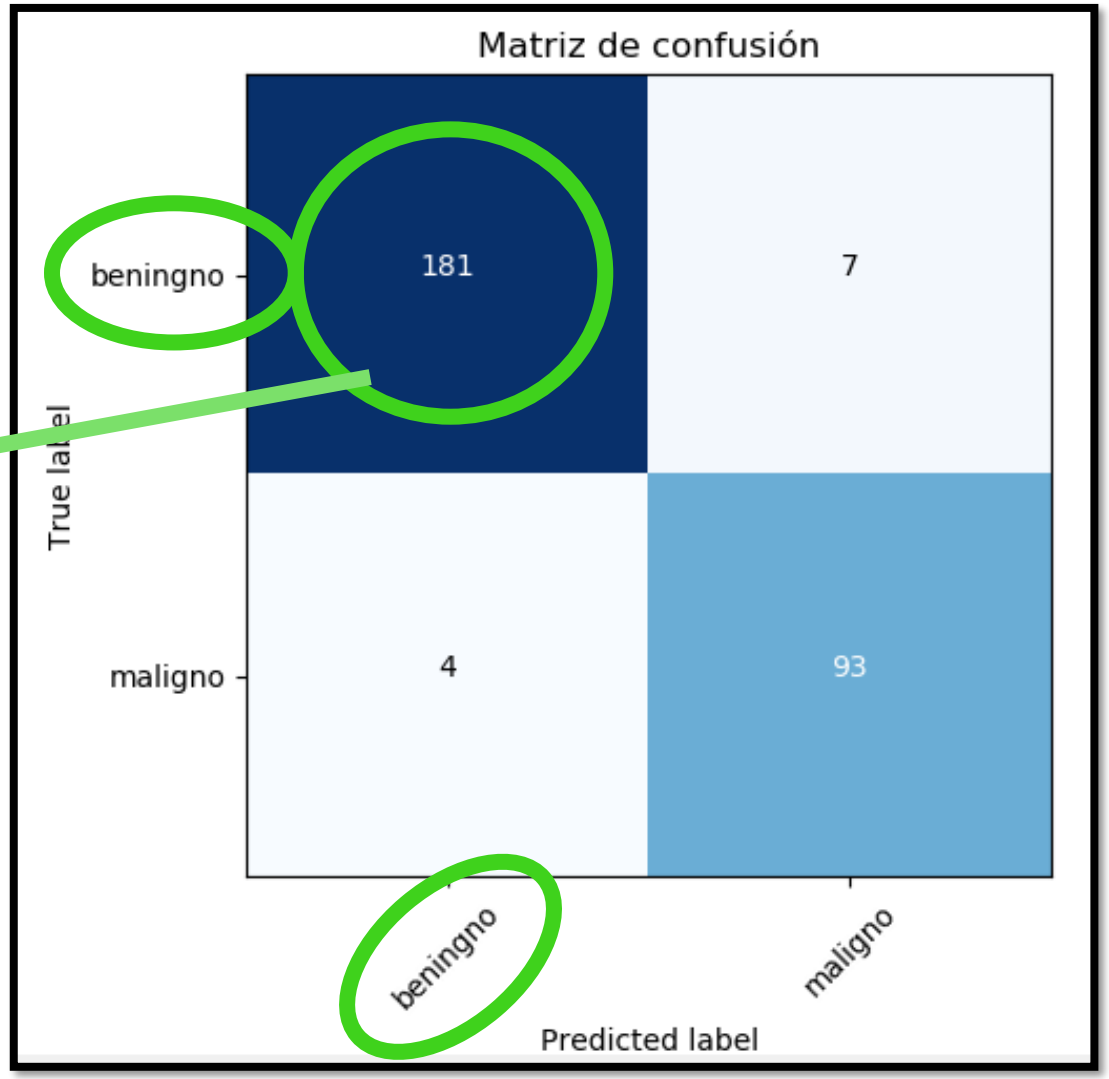
- Una matriz de confusión permite visualizar el funcionamiento del modelo de clasificación.
- Las filas representan la etiqueta real de cada clase, mientras que las columnas representan la predicción del modelo entrenado.
- La diagonal principal representa el correcto funcionamiento del modelo.

```
AAPutils.plot_confusion_matrix(y_true, y_pred)
```



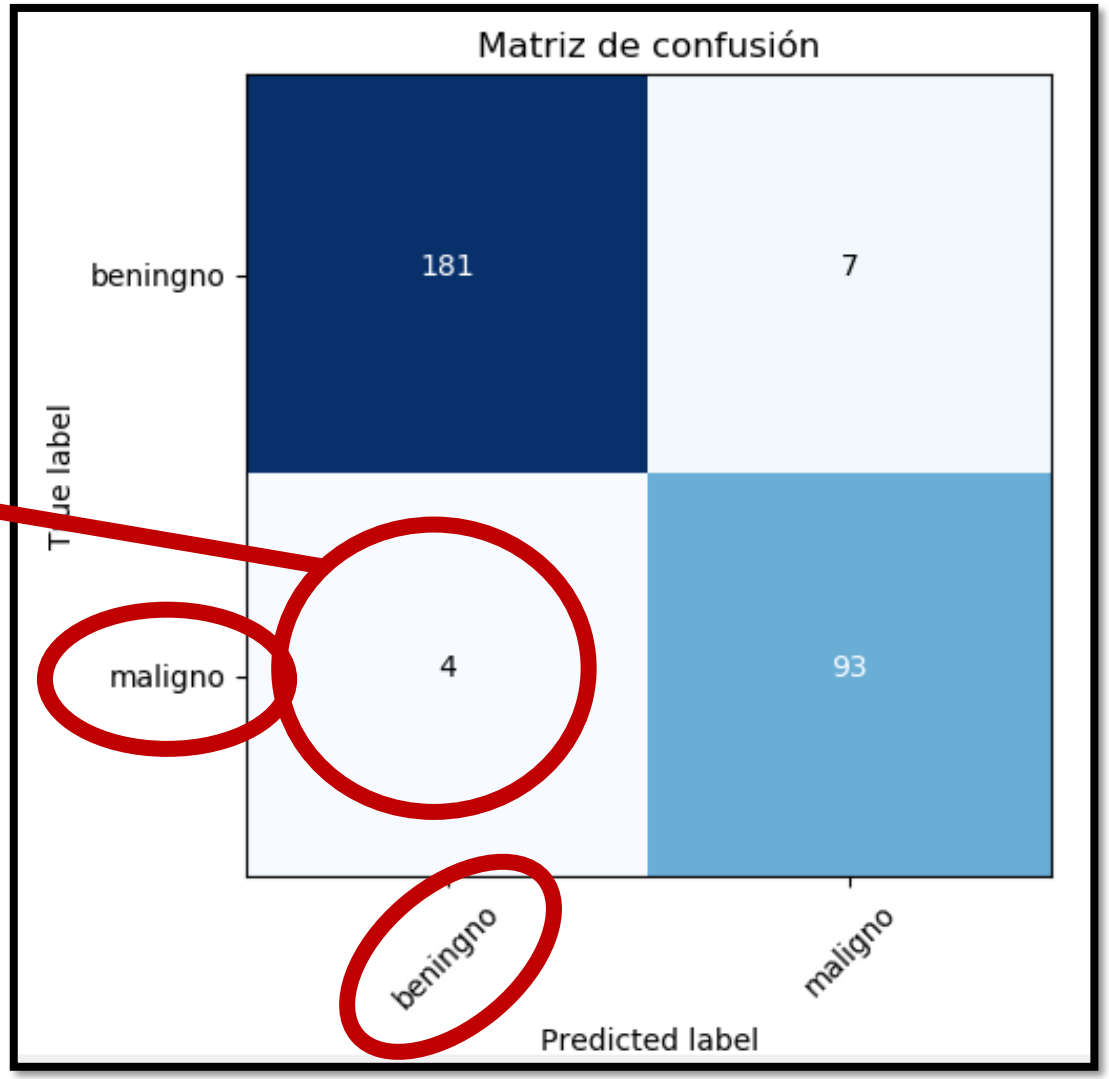
Matriz de confusión (clasificación binaria)

181 ejemplos eran Benignos y el modelo los clasificó como Benignos.



Matriz de confusión (clasificación binaria)

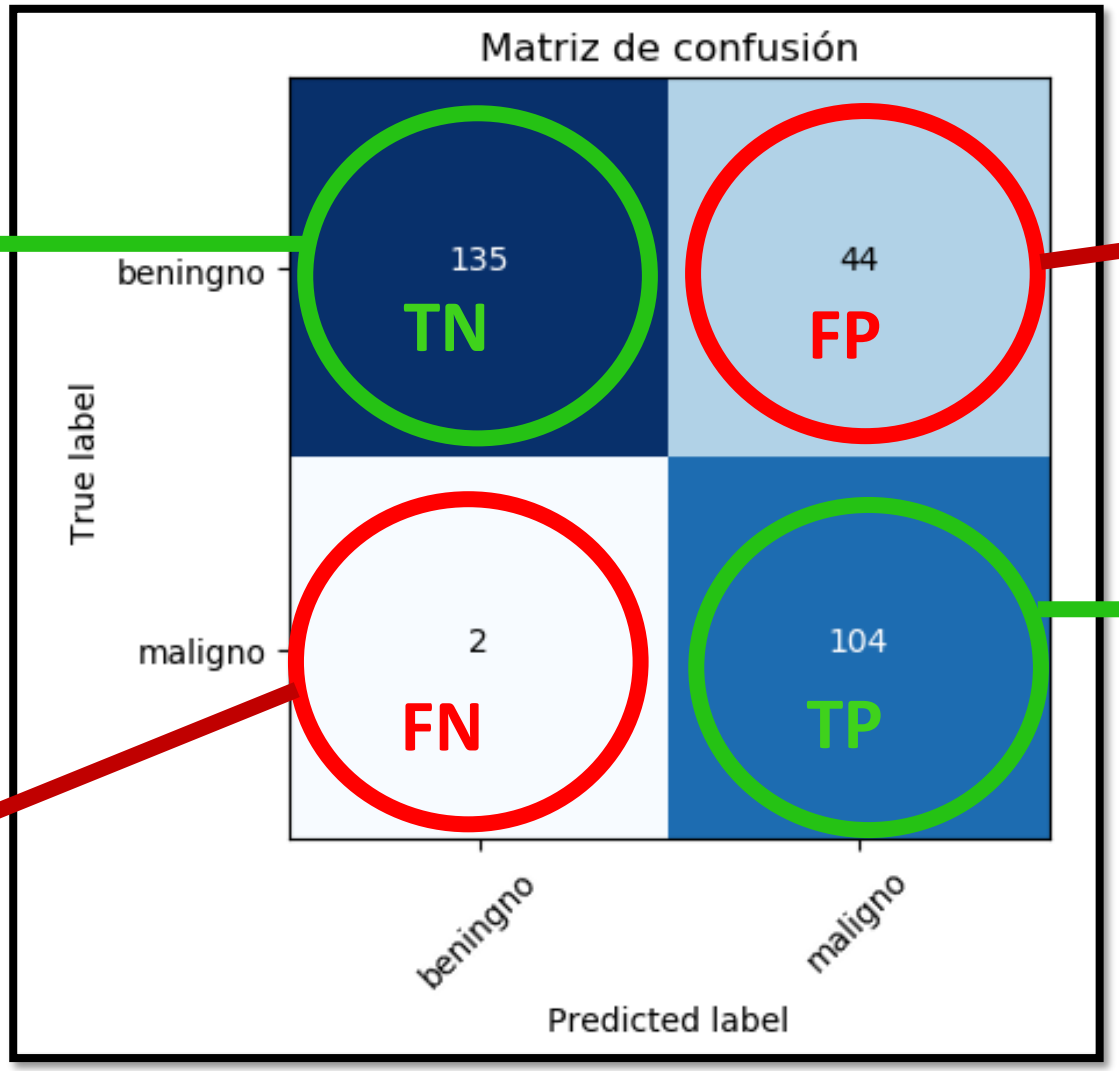
4 ejemplos eran Malignos y el modelo los clasificó como Benignos (Error).



Métricas

True Negatives

Falses Positives



True Positives

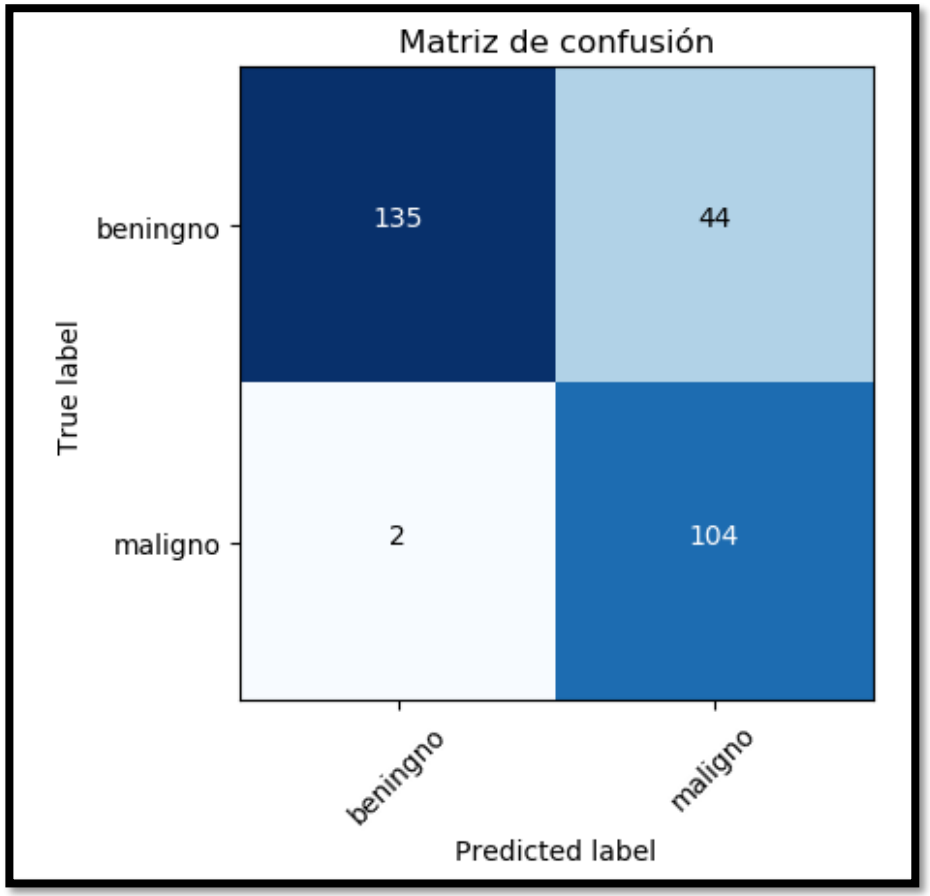
Falses Negatives

Métricas - Accuracy

$$\textbf{Accuracy} = \frac{num\ predicc.\ correctas}{total\ predicciones} = \frac{TP + TN}{(TP+TN+FP+FN)}$$

$$\textbf{Acc} = \frac{1}{n}\sum_i^n (y\hat{}_i - y_i)$$

Acc = (104+135) / (104+135+44+2)= **0,84**

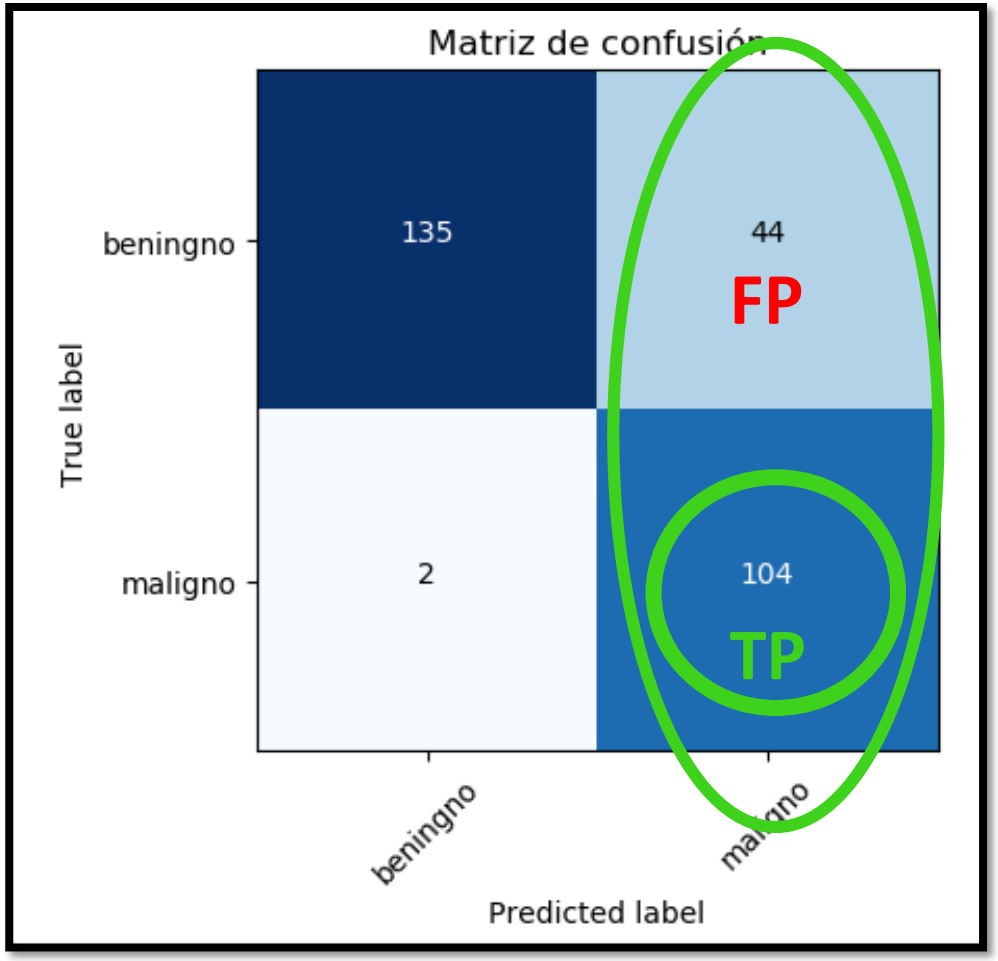


Métricas - Precision

$$\textbf{Precision} = \frac{TP}{(TP+FP)}$$

Prec = 104 / (104+44)= **0,70**

La **precisión** nos dice cuántos ítems reconocidos son realmente relevantes

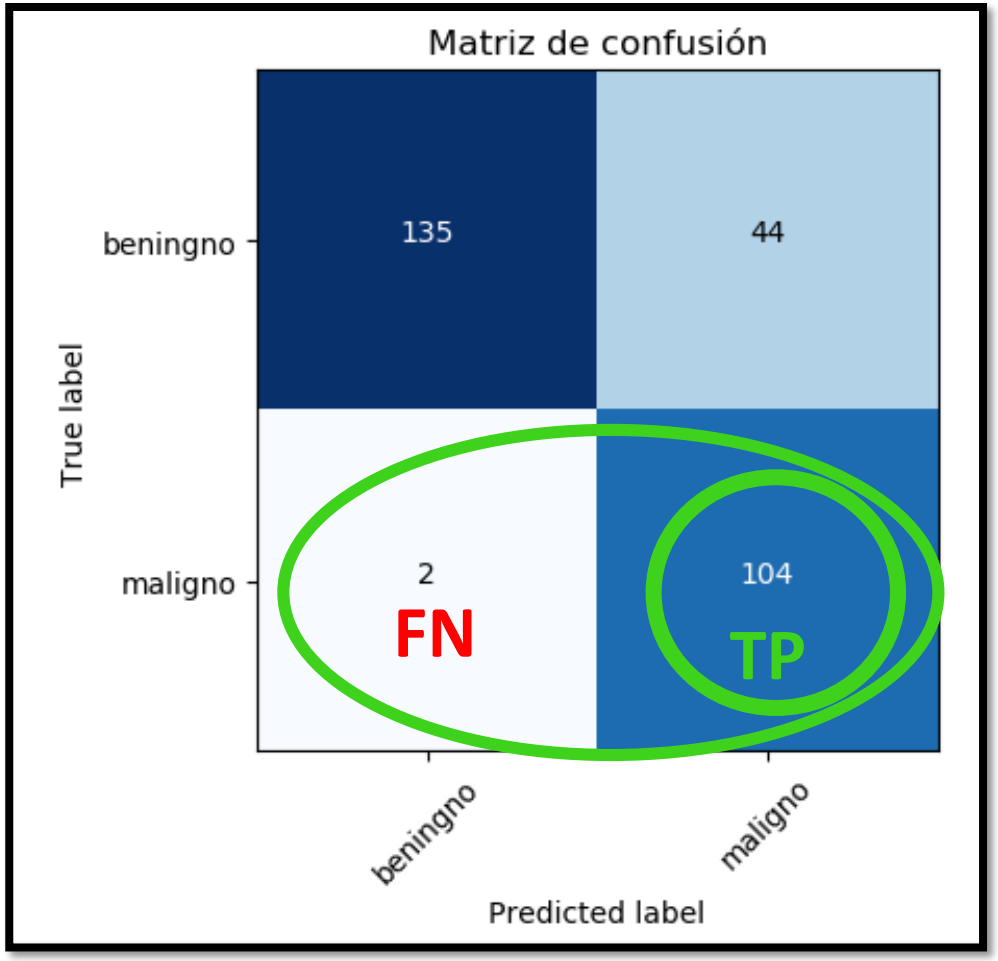


Métricas - Recall

$$\textbf{Recall} = \frac{TP}{(TP+FN)}$$

Rec = 104 / (104+2)= **0,98**

El **recall** nos dice cuántos ítems relevantes fueron realmente seleccionados



Métricas - Resumen

Acc = 0,84

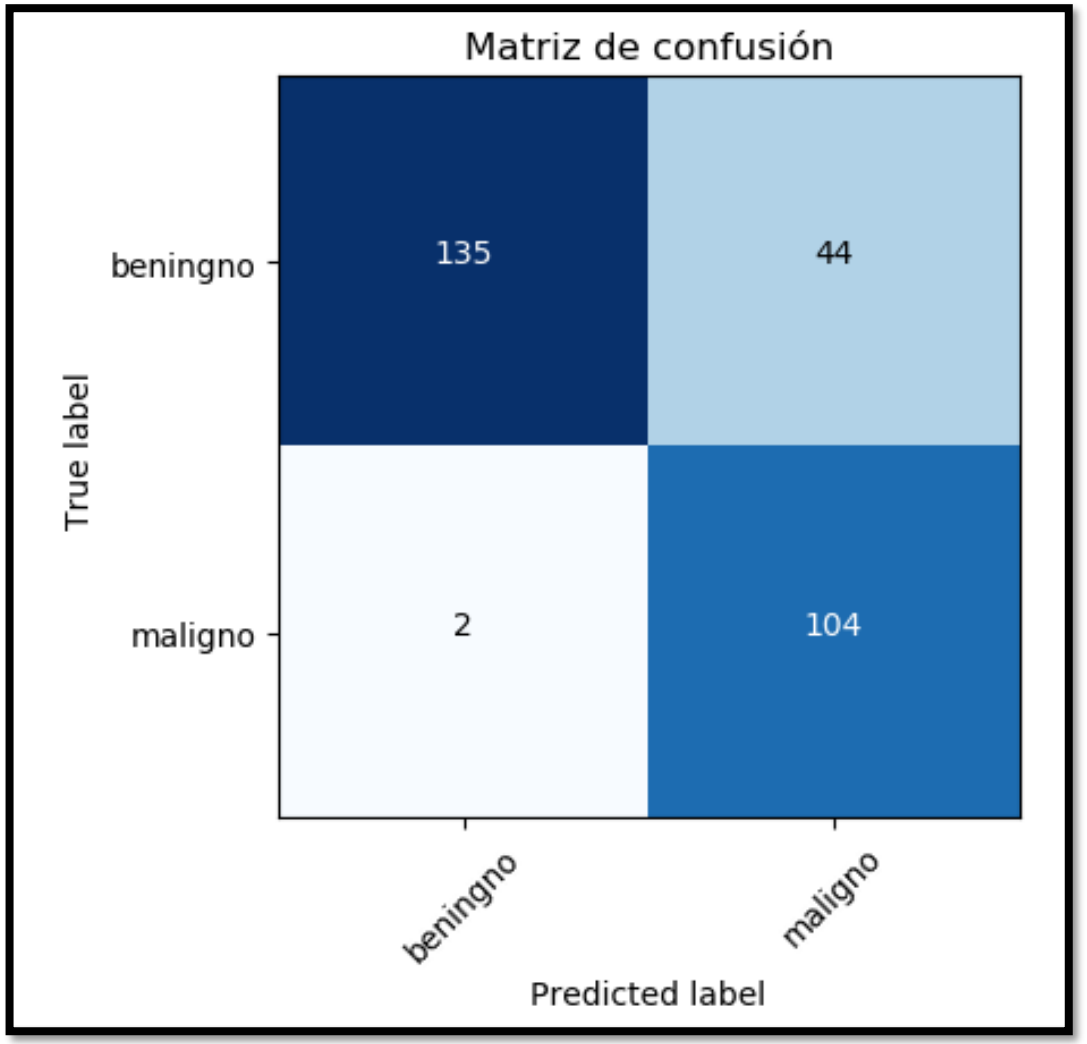
Prec = 0,70

Rec = 0,98

El **Acc**. nos muestra cómo se comporta el modelo teniendo en cuenta tanto los TP como los TN.

El **Prec**. Y **Rec**. Solo tienen en cuenta los TP.

En clasificación binaria generalmente nos va a interesar más el Prec. Y Rec.



Métricas – F-measure

Para encontrar un balance entre estas métricas, suele utilizarse la media armónica, también conocida como *f-measure* (o *F1-score*):

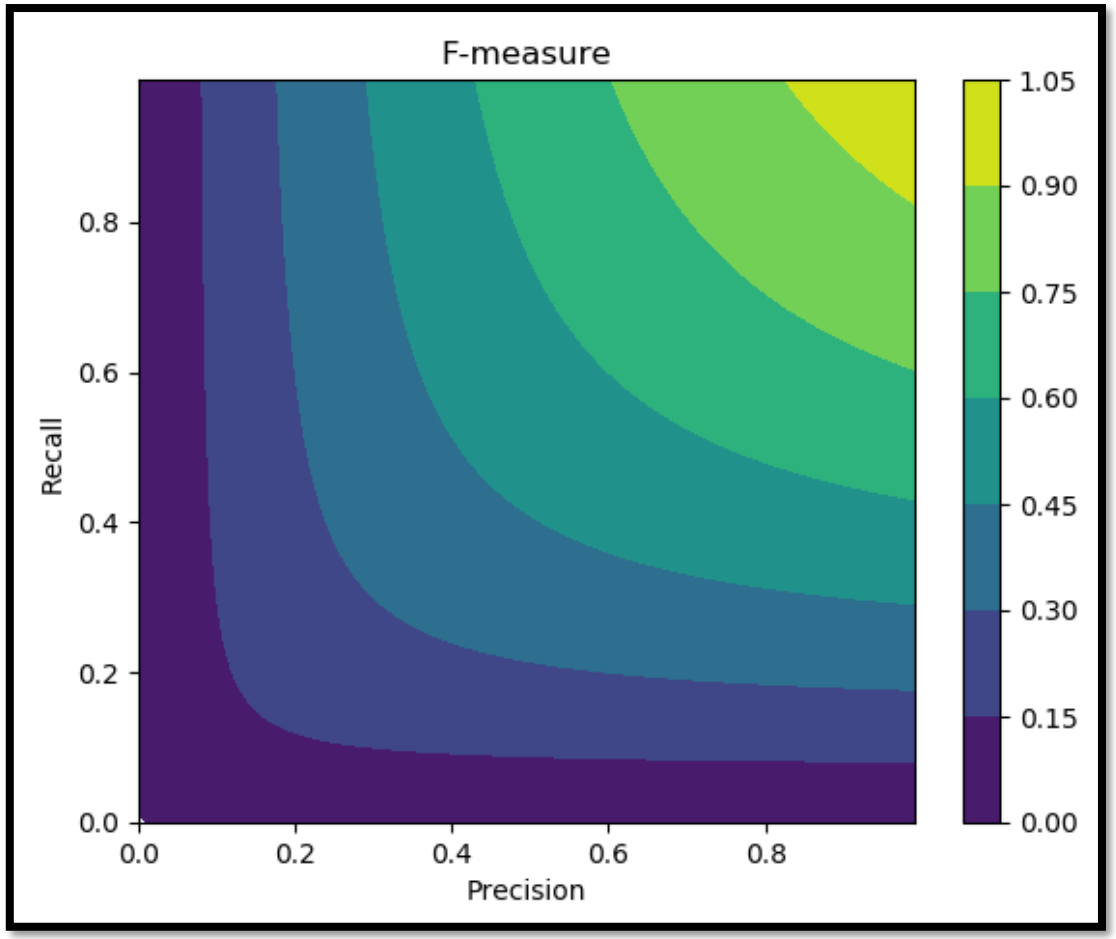
$$f - measure = 2 \times \frac{precision \times recall}{precision + recall}$$

Acc = 0,84

Prec = 0,70

Rec = 0,98

F-measure = 0,82



Ejemplos métricas – Dataset desbalanceado

Acc = 0,98

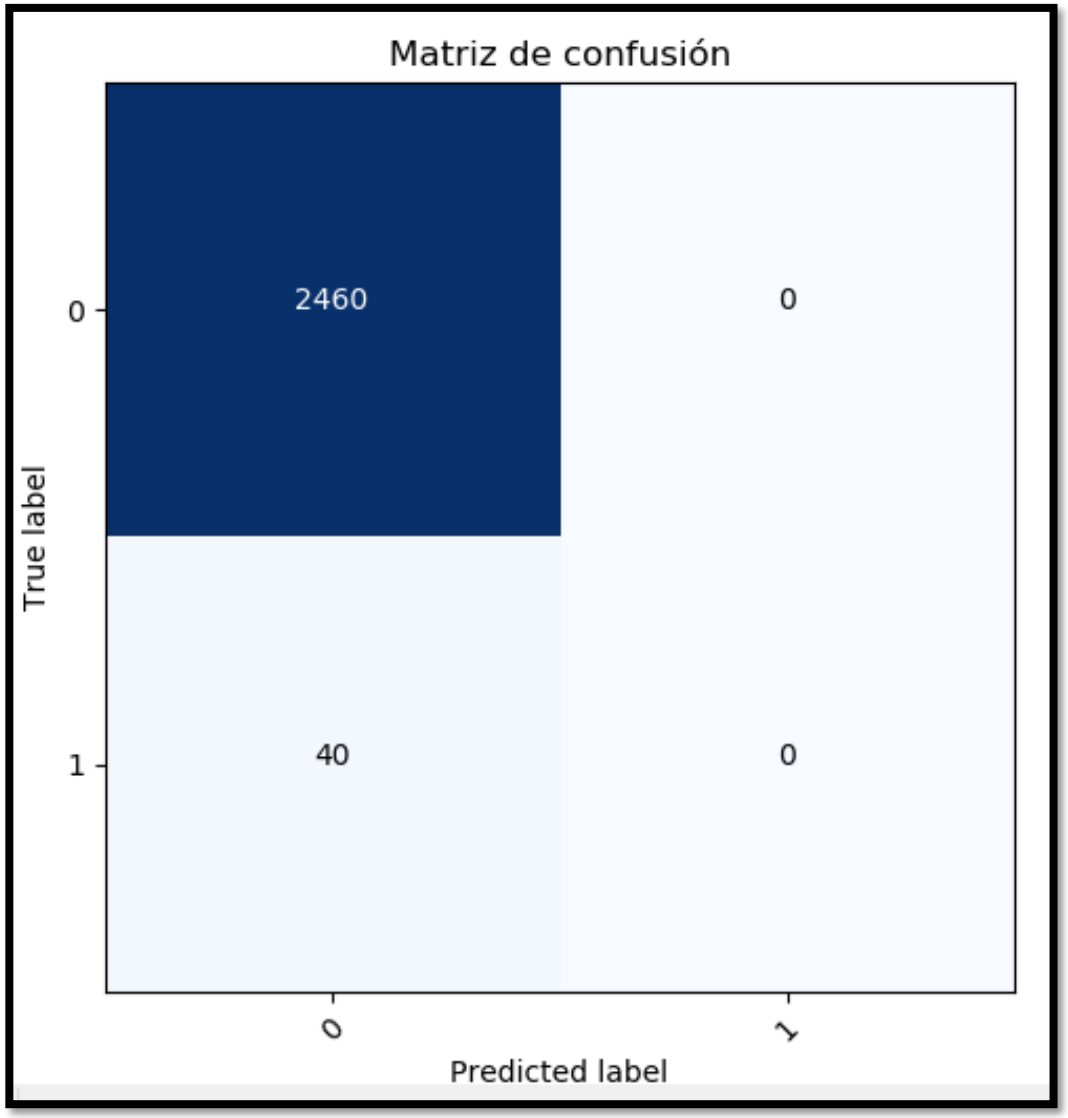
Prec = 0

Rec = 0

F-measure = 0

En este caso el *dataset* está muy desbalanceado (una clase es mayoritaria), lo que ocasiona que el modelo "aprenda" a decir siempre lo mismo.

Recordar que el modelo minimiza el error medio de todos los ejemplos.

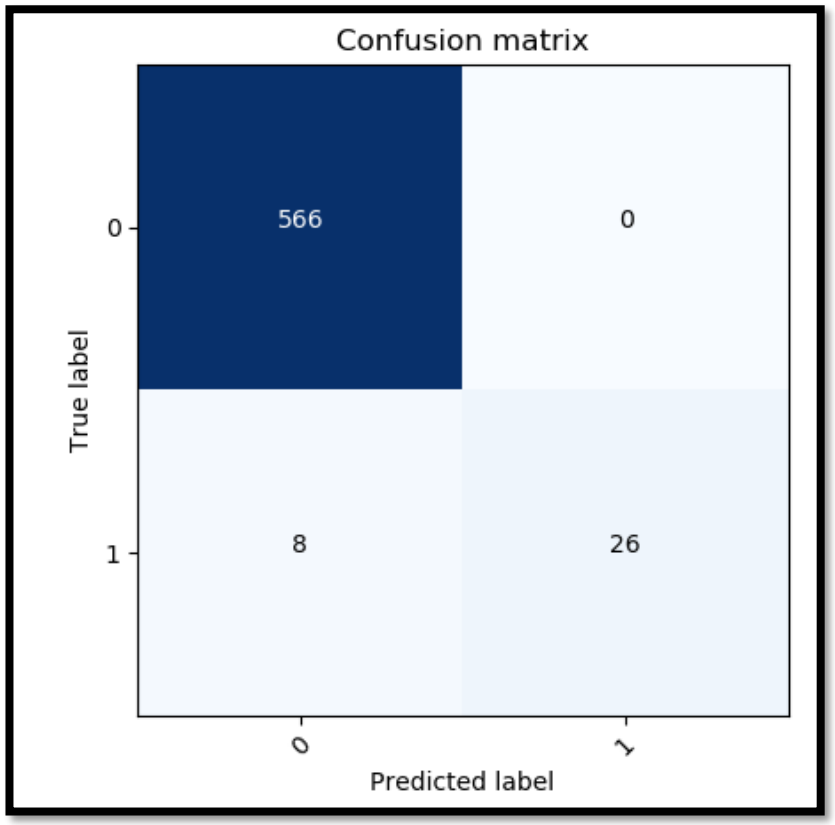


Dataset desbalanceado

Tenemos dos modos de solucionar el problema del desbalance de clases:

- Uno sería implementar nuestra propia función de error para que tenga en cuenta otra métrica.
- Otro es con el parámetro ***class_weight*** al momento de entrenar el modelo. Este parámetro es un diccionario que indica el peso que se le dará a cada clase dentro de la función de error durante el entrenamiento.

```
Test    Accuracy: 0.99
       Precision: 1.00
          Recall: 0.76
       f-measure: 0.87
```

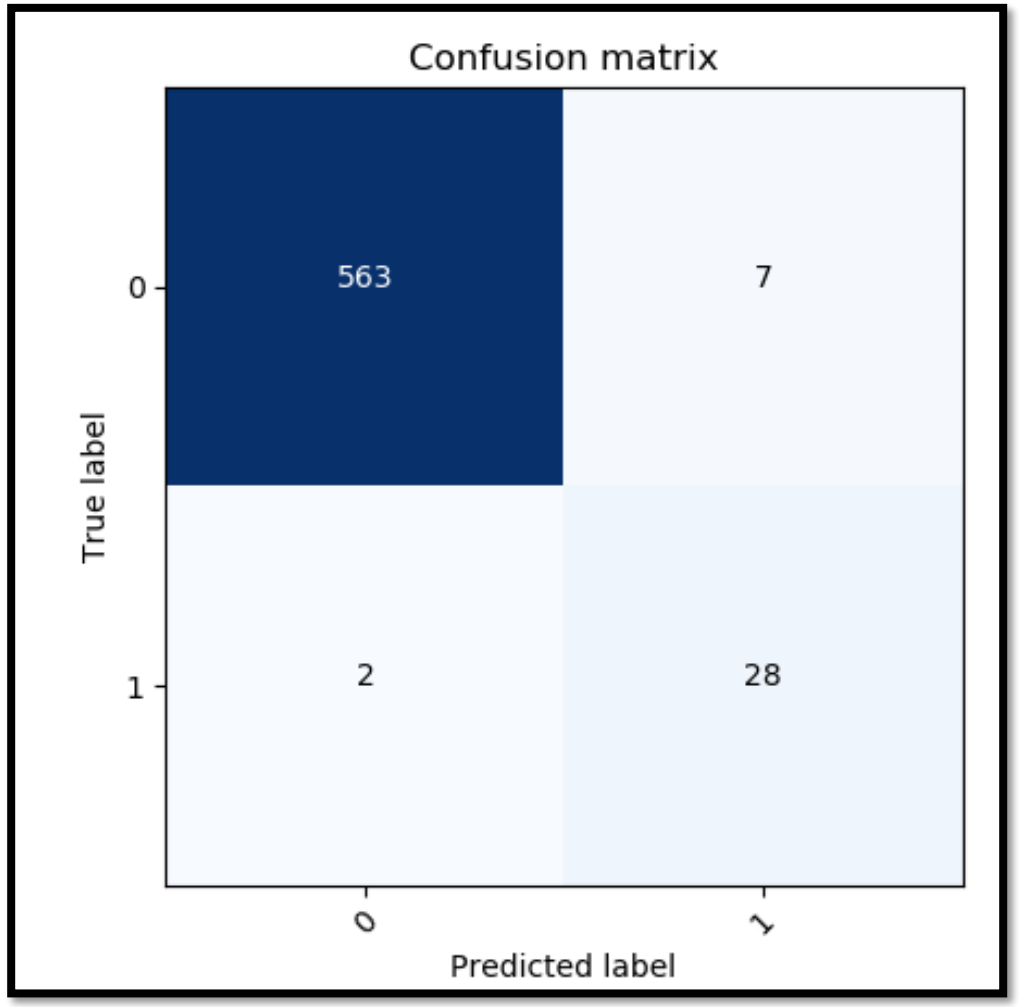


Dataset desbalanceado

```
Test    Accuracy: 0.98
        Precision: 0.80
           Recall: 0.93
         f-measure: 0.86
```

```
model.fit( X_train, Y_train,
           epochs= EPOCAS,
           batch_size= LOTE,
           class_weight= {0:0.3, 1:0.7} )
```

A medida que inclinamos el peso hacia una u otra clase, aumentará el **Recall** y disminuirá el **Precision** o viceversa.



Dataset desbalanceado

```
Test    Accuracy: 0.99
        Precision: 1.00
           Recall: 0.76
         f-measure: 0.87
```

`class_weight= {0:0.5, 1:0.5}`

```
Test    Accuracy: 0.98
        Precision: 0.80
           Recall: 0.93
         f-measure: 0.86
```

`class_weight= {0:0.3, 1:0.7}`

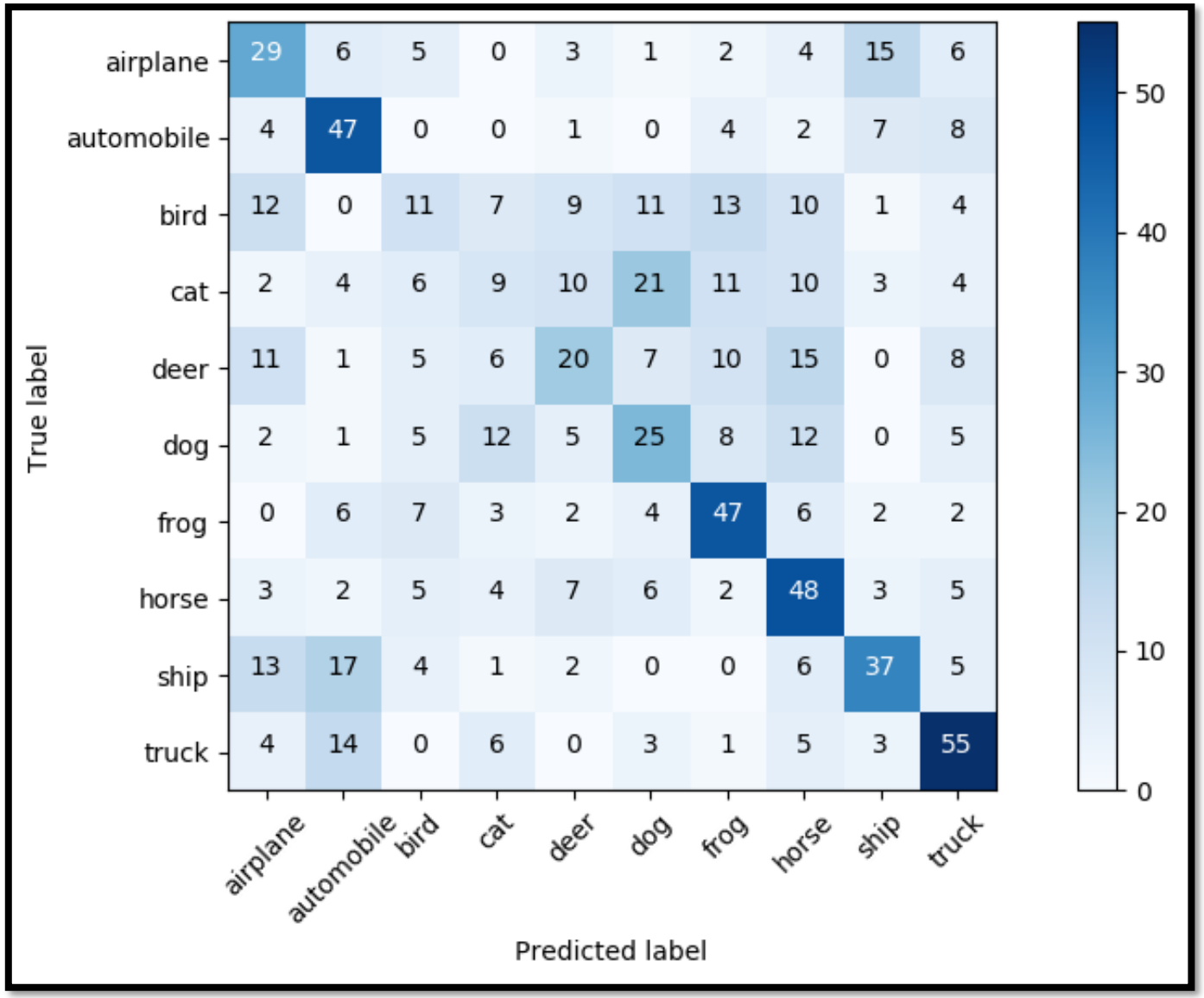
```
Test    Accuracy: 0.95
        Precision: 0.60
           Recall: 0.95
         f-measure: 0.73
```

`class_weight= {0:0.05, 1:0.95}`

Matriz de confusión (clasificación multiclase)

Misma idea para multiclase.

Por ahora seguiremos con Clasificación Binaria.

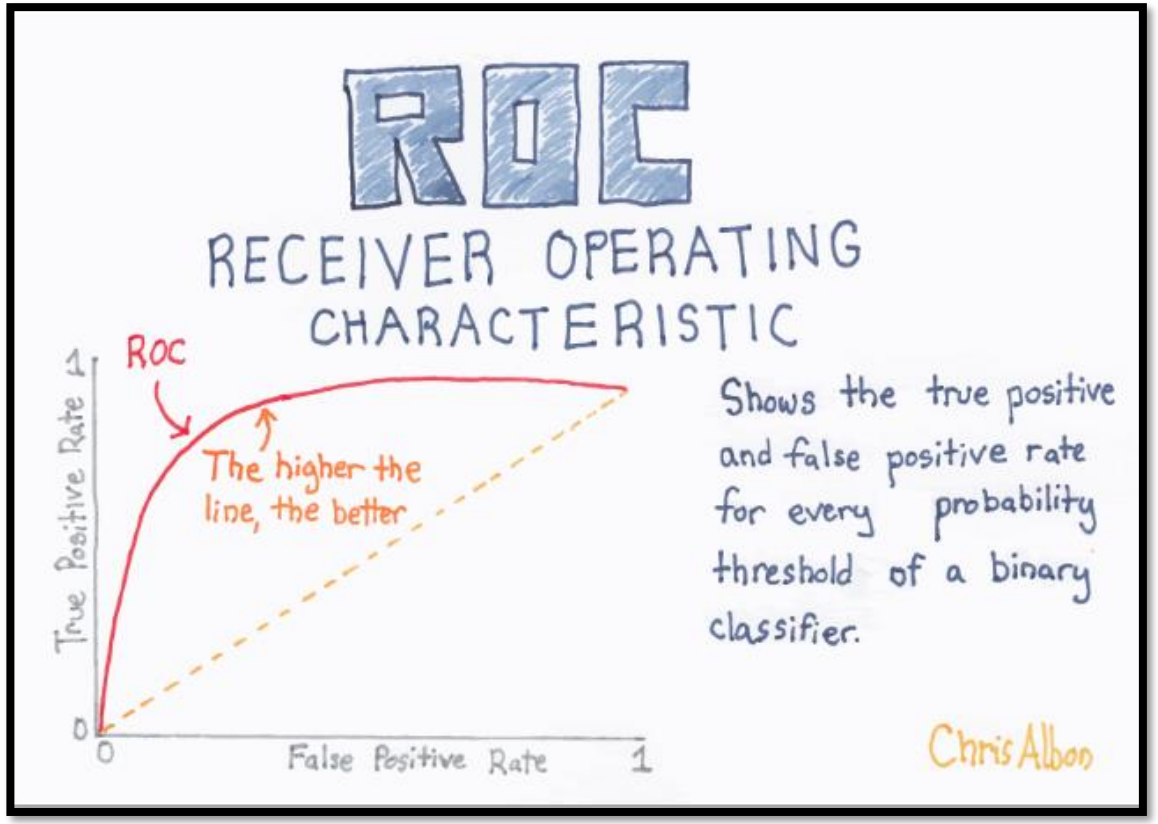


Curvas ROC

Las curvas **ROC** (Receiver Operating Characteristic) nos muestran cómo se comporta un modelo binario al cambiar el umbral de detección.

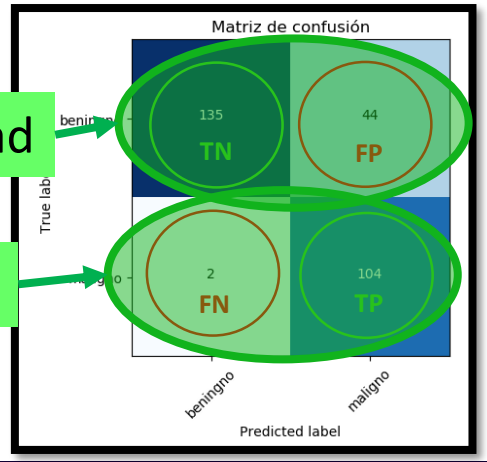
True Positive Rate = Sensitividad = **Recall**= TP/(TP+FN)

False Positive Rate= 1 - Especificidad= 1- TN/(FP+TN)



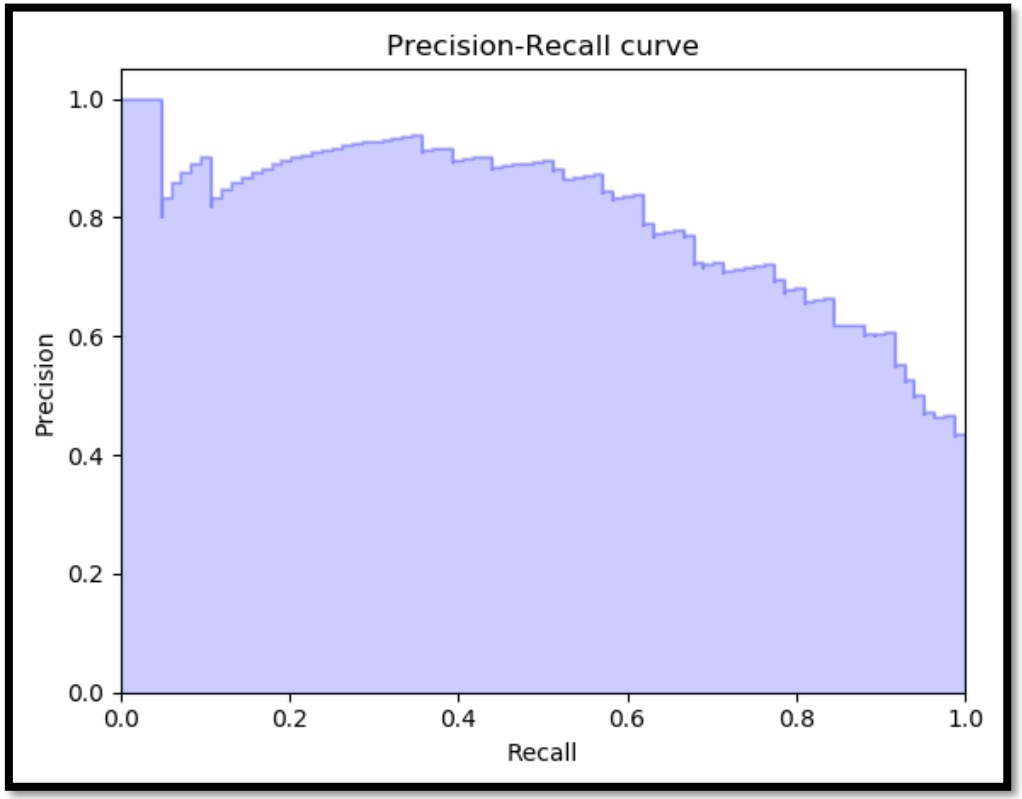
Especificidad

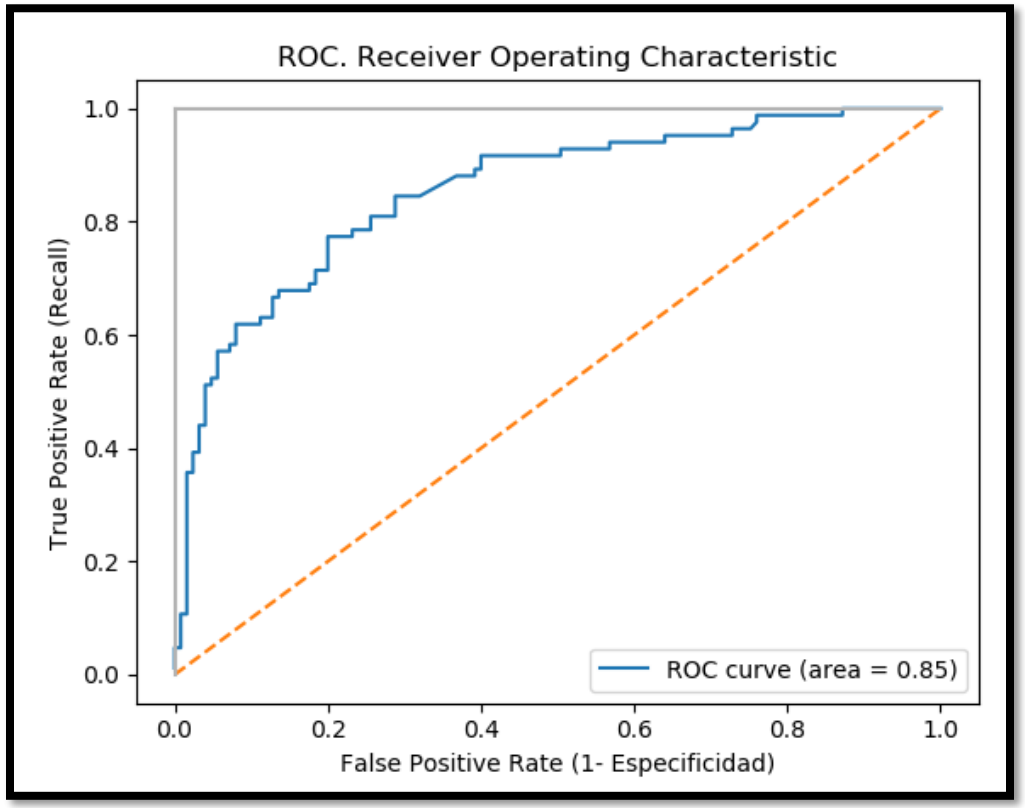
Sensitividad



Curvas Precision-Recall

Las curvas **Precision-Recall** se computan de igual modo que las curvas ROC pero grafican el funcionamiento del modelo para estas métricas.

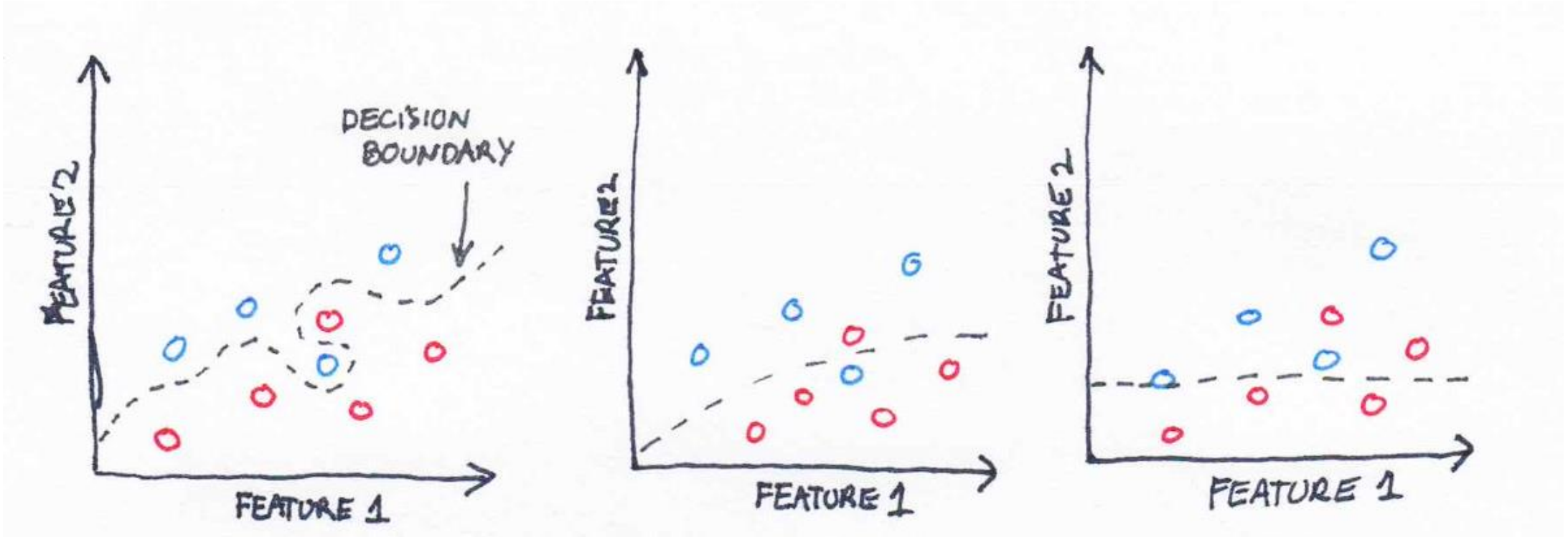





```
AAPutils.plot_ROC_curve(model, x, y)
```

Sobreajuste (*overfitting*)

El *Overfitting* ocurre cuando el modelo es muy complejo e intenta adecuarse perfectamente a los datos de entrenamiento.



Modelo con Overfitting.

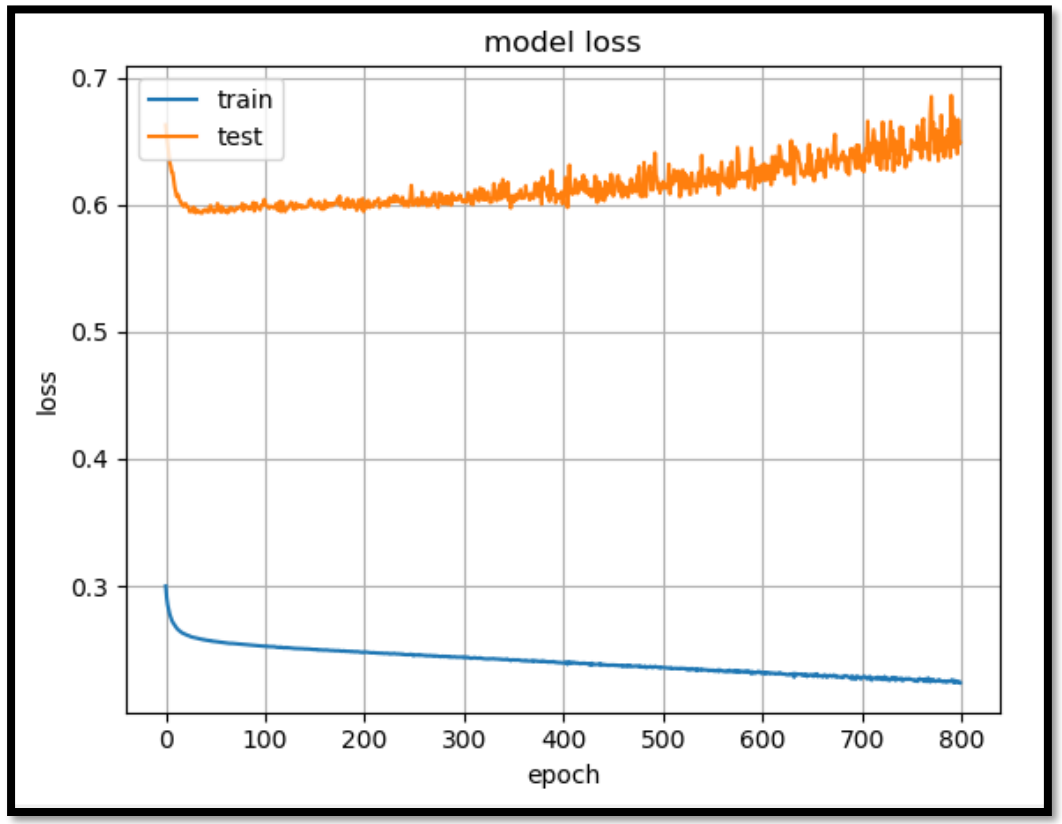
Modelo “correcto”.

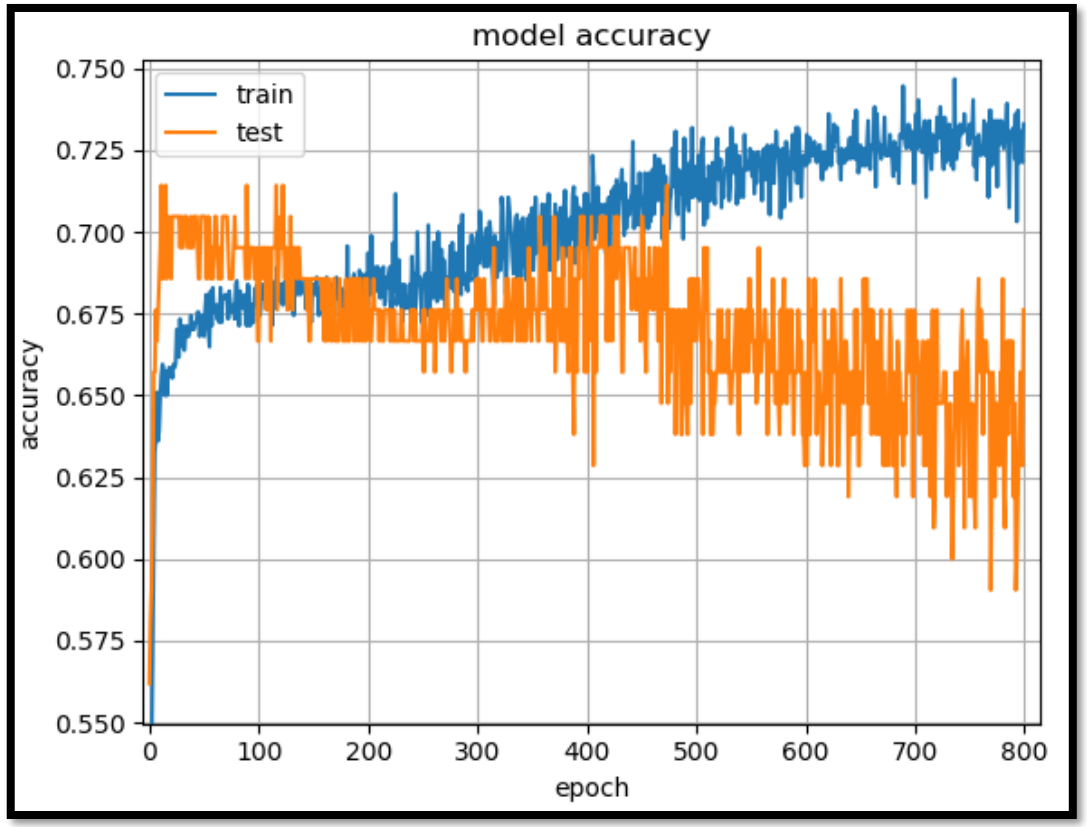
Modelo con UnderFitting.

Overfitting

Podemos verificar el Overfitting si el error en Training es bajo y el de testing es alto.

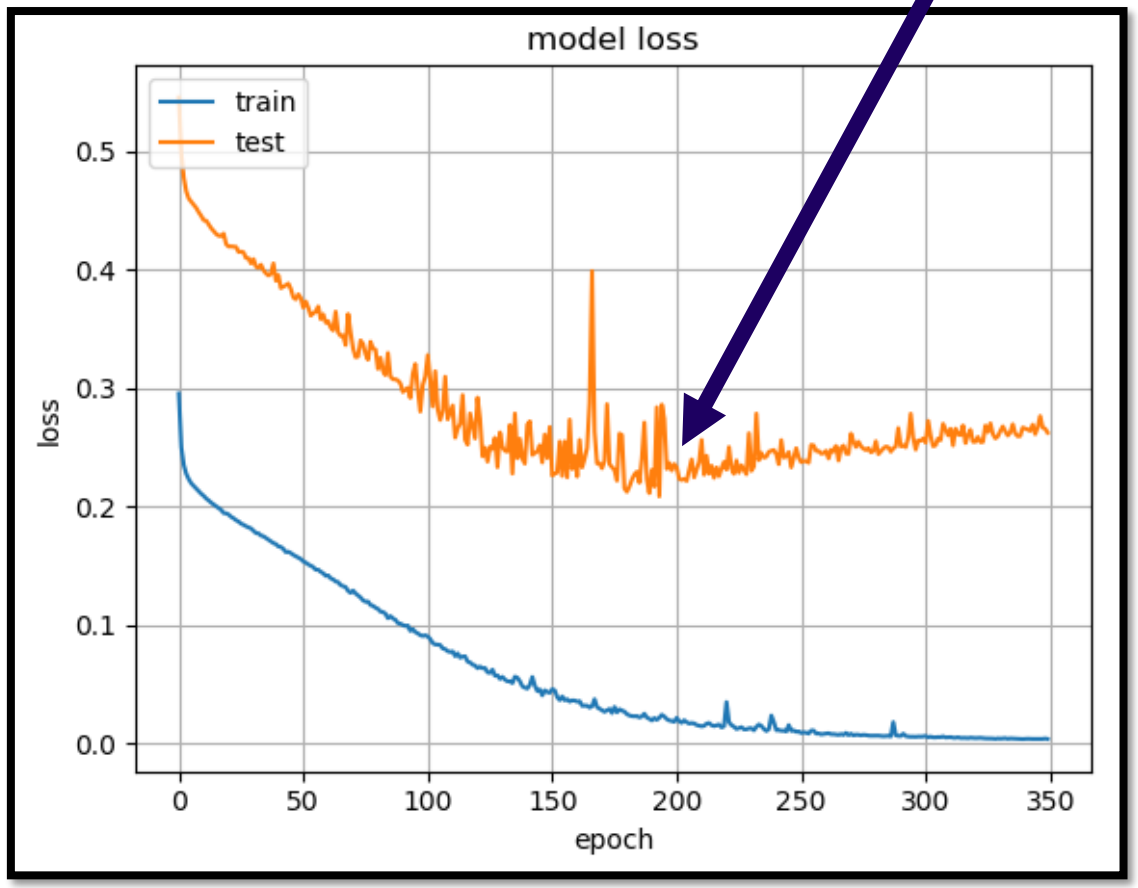
Si observamos las curvas de aprendizaje podemos detectar el momento en que comienza a sobreajustarse el modelo.



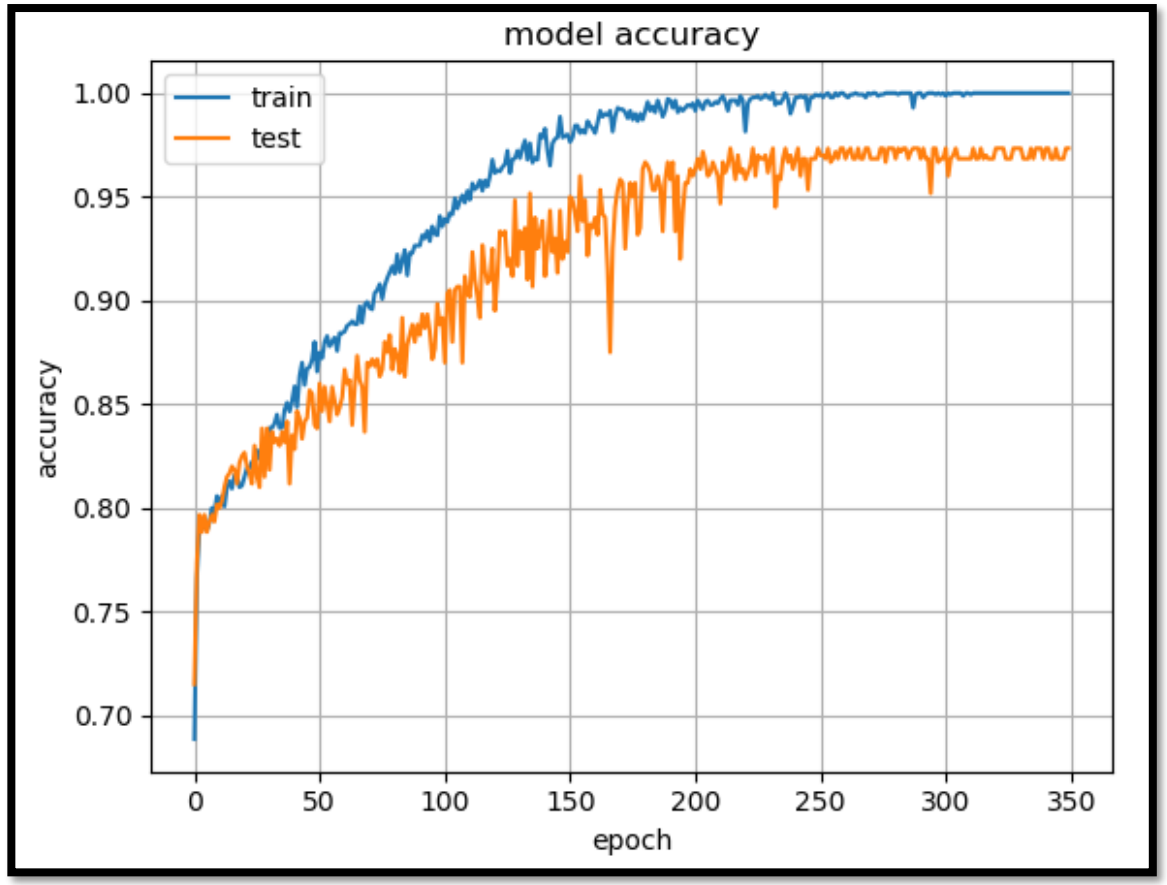


Overfitting

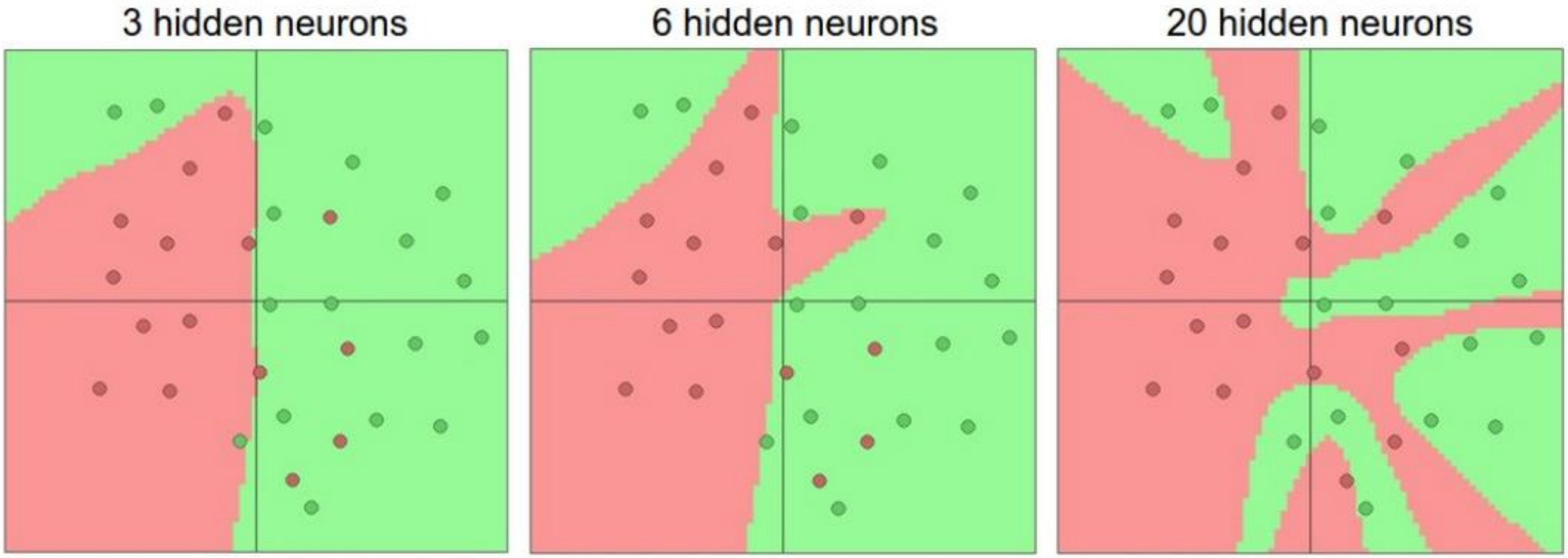
Momento en que el modelo comenzó a Sobreentrenar



En este caso, el accuracy no se vio demasiado afectado.



Overfitting



Hiperparámetros

Los hiperparámetros son los que guían el algoritmo de optimización de nuestro modelo.

No existe una configuración ideal, sino que depende del modelo y del *dataset* específico.

- Cantidad de capas y cant. de neuronas por capa.
- Fx activación
- Learning rate (alfa)
- Batch size
- # Epochs
- Regularización
- …

Regularización

La regularización es un modo de restringir la complejidad de un modelo.

Es útil cuando hay mucha correlación entre features, para filtrar ruido en los datos y también para **evitar el Overfitting**.

Aprendizaje → `Loss + Penalización`

L2 Regularization:

$$\boldsymbol{Loss} = \underbrace{\frac{1}{n}\sum_{i}^{n} E_i}_{\text{Error plano}} + \underbrace{\boldsymbol{\lambda}\sum_{j}^{m} \boldsymbol{w_j^2}}_{\text{Penalización}}$$

Regularización