```
In [1]: # Importación de las bibliotecas
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import tensorflow as tf
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import classification_report, confusion_matrix
        from sklearn import metrics
        from sklearn.model_selection import cross_val_score
        from sklearn.model_selection import GridSearchCV
        import warnings
        warnings.filterwarnings("ignore", category=FutureWarning)

        sns.set()
```

Lectura de los datos desde TensorFlow

```
In [2]: # Construcción de las variables de entrada y salida
        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
In [3]: # Imprimiendo los resultados
        print (x_train.shape)
        print (x_test.shape)
        print (y_train.shape)
        print (y_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

Aplicando las conversiones necesarias para pasar de 3d-array a 2d-array

```
In [4]: # Creando separacion de los datos
        x_train = x_train.reshape(60000, -1)
        x_test = x_test.reshape(10000, -1)

        n_train = 600

        x_train1 = x_train[0:n_train]
        y_train1 = y_train[0:n_train]
```

```
In [5]: # Imprimiendo los resultados obtenidos
        print (x_train.shape)
        print (x_test.shape)
        print (x_train1.shape)
        print (y_train1.shape)
```

```
(60000, 784)
(10000, 784)
(600, 784)
(600,)
```

Creación del primer modelo con n_train = 600

Para este primer ejemplo vamos a utilizar la biblioteca GridSearch, la que nos va a ayudar a determinar cuáles son los mejores parámetros a utilizar en el hipertuneo, así podremos garantizar que nuestro algoritmo utilizará los mejores parámetros posibles.

Al ser un proceso computacional costoso, vamos a hacerlo con solo aquellos paramétros que son más populares como: n_estimators, max_features, max_depth y criterion y con ciertas combinaciones; ya que sino nuestro modelo tardará bastante tiempo ejecutándose.

```
In [6]: # Creando el modelo Random Forest
        rf_model_1 = RandomForestClassifier(random_state=0)
```

```
In [7]: # Creando estimador de parametros
        param_grid = {
            'n_estimators': [300, 500],
            'n_jobs': [2,5],
            'max_features': ['auto', 'sqrt', 'log2'],
            'max_depth' : [3,4,5],
            'criterion' :['gini', 'entropy']
        }
```

```
In [8]: # Realizando Cross Validation con 3 y probando parametros
        CV_rfc = GridSearchCV(estimator=rf_model_1, param_grid=param_grid, cv= 3)
        CV_rfc.fit(x_train1, y_train1)
```

```
Out[8]: GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=0),
                     param_grid={'criterion': ['gini', 'entropy'],
                                 'max_depth': [3, 4, 5],
                                 'max_features': ['auto', 'sqrt', 'log2'],
                                 'n_estimators': [300, 500], 'n_jobs': [2, 5]})
```

```
In [9]: # Comprobando cuales son los mejores parametros a utilizar
        CV_rfc.best_params_
```

```
Out[9]: {'criterion': 'entropy',
         'max_depth': 5,
         'max_features': 'auto',
         'n_estimators': 300,
         'n_jobs': 2}
```

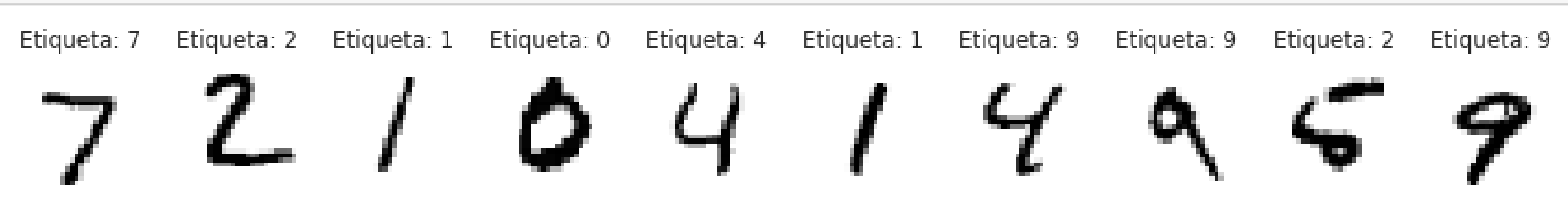
```
In [10]: # Creando modelo con los mejores parametros identificados
         rf_model = RandomForestClassifier(max_depth=5, random_state=0, n_jobs=2, n_estimators=300, max_features
          = 'auto', criterion = 'entropy')
```

```
In [11]: # Ingestando los datos necesarios de training
         rf_model.fit(x_train1, y_train1)
```

```
Out[11]: RandomForestClassifier(criterion='entropy', max_depth=5, n_estimators=300,
                                n_jobs=2, random_state=0)
```

```
In [12]: # Prediciendo el valor del digito en el subset de testing solo con 100 observaciones
         predicted = rf_model.predict(x_test[0:100])
```

```
In [13]: # Creando función que retorne los valores predichos versus los valores reales
         _, axes = plt.subplots(nrows=1, ncols=10, figsize=(15, 15))
         for ax, image, label in zip(axes, x_test[0:100], predicted):
             ax.set_axis_off()
             image = image.reshape(28, 28)
             ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
             ax.set_title('Etiqueta: %i' % label)
```



Análisis: se puede observar como utilizando una cantidad de solo 100 "imágenes" de testing y habiendo utilizado el hipertuneo con GridSearch para entrenar el modelo, con un max_depth=5, random_state=0, n_jobs=2, n_estimators=300, max_features = 'auto', criterion = 'entropy' se obtienen resultados relativamente buenos, ya que se puede apreciar como de 10 predicciones en la imagen el algoritmo logró acertar en 8 de ellas.

```
In [14]: # Imprimiendo resultados para ver el accuracy, precision y recall
         print(
             f"Classification report for classifier {rf_model}:\n"
             f"{metrics.classification_report(y_test[0:100], predicted)}\n"
         )
```

```
Classification report for classifier RandomForestClassifier(criterion='entropy', max_depth=5, n_estim
ators=300,
                       n_jobs=2, random_state=0):
              precision    recall  f1-score   support

           0       0.89      1.00      0.94         8
           1       1.00      1.00      1.00        14
           2       0.55      0.75      0.63         8
           3       0.91      0.91      0.91        11
           4       0.79      0.79      0.79        14
           5       1.00      0.29      0.44         7
           6       0.75      0.60      0.67        10
           7       0.92      0.80      0.86        15
           8       0.50      0.50      0.50         2
           9       0.62      0.91      0.74        11

    accuracy                           0.80       100
   macro avg       0.79      0.75      0.75       100
weighted avg       0.83      0.80      0.79       100
```

Análisis: gracias al resumen que se nos muestra podemos ver que la predicción a nivel de precisión, que recordemos que nos responde la siguiente pregunta: ¿qué porcentaje del dígito que identifiquemos estará realmente correcto? (es decir, mide la calidad del modelo). En este caso podemos ver que el dígito que tiene el score más alto es el dígito 1 y 5, con un 100% de acierto, es decir, el 100% de las veces el algoritmo será capaz de idenficar los números 1y 5, por ende nunca se equivocará, mientras que la más baja es la de 8 con un 50%, es decir, para el algoritmo es tarea fácil identificar el 1 y el 5, pero no así el 8 (tiende a cometer más error tipo I).

A nivel del recall (esta nos informa sobre la cantidad que el modelo es capaz de identificar) que recordemos que este responde a la pregunta: ¿qué porcentaje del dígito correcto somos capaces de identificar? Podemos ver que el mayor es el dígito 0 y 1 con un 100%, es decir, nunca se equivocará, mientras que el más bajo es el dígito 5 con un 29%, es decir para el dígito 5 conlleva una mayor cantidad de FN que el algoritmo predice incorrectamente (error tipo II).

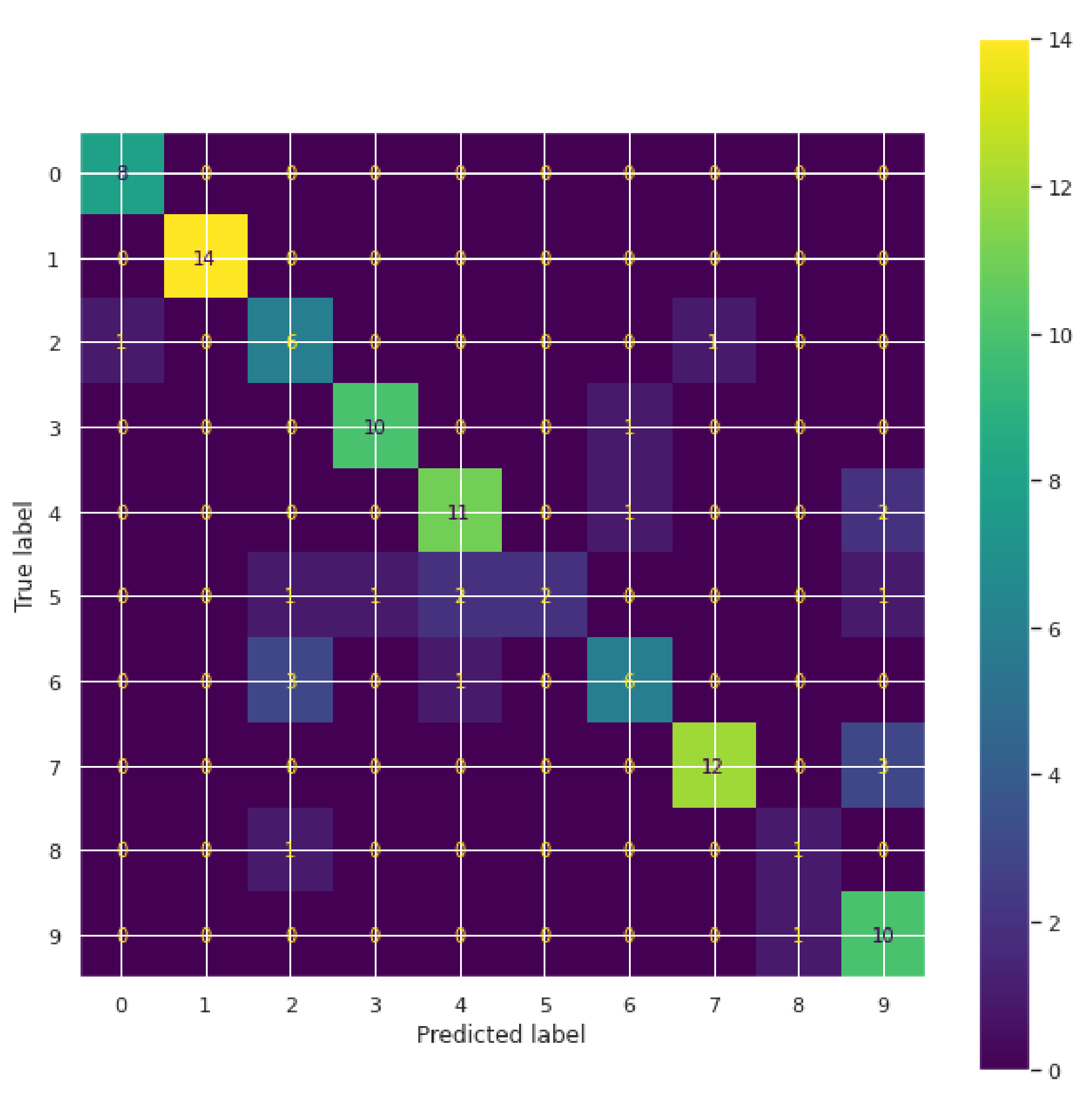
Finalmente podemos ver como el F1 que, recordemos que hace más fácil el poder comparar el rendimiento combinado de la precisión y la exhaustividad entre varias soluciones, obtenemos que el que tiene mayores problemas idenficando es el dígito 5 con un 44%, mientras que el que idenfica de forma bastante aceptable es con el dígito 1 con un 100%, los demás se mantienen en un rango entre 50 y 90%.

```
In [15]: # Creando visualizacion de la matriz de confusion
         print(f"Confusion matrix:\n")
         fig, ax = plt.subplots(figsize=(10, 10))
         metrics.plot_confusion_matrix(rf_model, x_test[0:100], y_test[0:100], cmap=plt.cm.viridis, ax=ax)
         plt.show()

         print("\n")

         print(f"Confusion matrix:\n{confusion_matrix(y_test[0:100], predicted)}")
```

```
Confusion matrix:
```



```
Confusion matrix:
[[ 8  0  0  0  0  0  0  0  0  0]
 [ 0 14  0  0  0  0  0  0  0  0]
 [ 1  0  6  0  0  0  0  1  0  0]
 [ 0  0  0 10  0  0  1  0  0  0]
 [ 0  0  0  0 11  0  1  0  0  2]
 [ 0  0  1  1  2  2  0  0  0  1]
 [ 0  0  3  0  1  0  6  0  0  0]
 [ 0  0  0  0  0  0  0 12  0  3]
 [ 0  0  1  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  1 10]]
```

Análisis: podemos observar de forma rápida que el dígito 1 es en el que suceden la mayor cantidad de aciertos (tal y como indicamos en el punto anterior) seguido del 0, mientras que las predicciones más pobres son realizadas por el número 5 y 9.

Creación del segundo modelo con n_train = 6000

```
In [16]: # Creando separacion de los datos
         x_train = x_train.reshape(60000, -1)
         x_test = x_test.reshape(10000, -1)

         n_train = 6000

         x_train1 = x_train[0:n_train]
         y_train1 = y_train[0:n_train]
```

```
In [17]: # Imprimiendo los resultados obtenidos
         print (x_train.shape)
         print (x_test.shape)
         print (x_train1.shape)
         print (y_train1.shape)
```

```
(60000, 784)
(10000, 784)
(6000, 784)
(6000,)
```

```
In [18]: # Creando el modelo Random Forest
         rf_model_2 = RandomForestClassifier(random_state=0)
```

```
In [19]: # Creando estimador de parametros
         param_grid = {
             'n_estimators': [300, 500],
             'n_jobs': [2,5],
             'max_features': ['auto', 'sqrt', 'log2'],
             'max_depth' : [3,4,5],
             'criterion' :['gini', 'entropy']
         }
```

```
In [20]: # Realizando Cross Validation con 3 y probando parametros
         CV_rfc = GridSearchCV(estimator=rf_model_2, param_grid=param_grid, cv= 3)
         CV_rfc.fit(x_train1, y_train1)
```

```
Out[20]: GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=0),
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'max_depth': [3, 4, 5],
                                  'max_features': ['auto', 'sqrt', 'log2'],
                                  'n_estimators': [300, 500], 'n_jobs': [2, 5]})
```

```
In [21]: # Comprobando cuales son los mejores parametros a utilizar
         CV_rfc.best_params_
```

```
Out[21]: {'criterion': 'gini',
          'max_depth': 5,
          'max_features': 'auto',
          'n_estimators': 500,
          'n_jobs': 2}
```

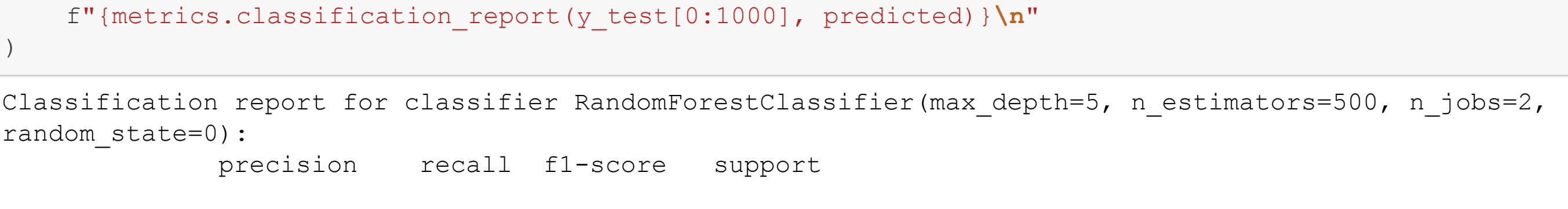
```
In [22]: # Creando modelo con los mejores parametros identificados
         rf_model_sec = RandomForestClassifier(max_depth=5, random_state=0, n_jobs=2, n_estimators=500, criterio
         n = 'gini', max_features= 'auto')
```

```
In [23]: # Ingestando los datos necesarios de training
         rf_model_sec.fit(x_train1, y_train1)
```

```
Out[23]: RandomForestClassifier(max_depth=5, n_estimators=500, n_jobs=2, random_state=0)
```

```
In [24]: # Prediciendo el valor del digito en el subset de testing
         predicted = rf_model_sec.predict(x_test[0:1000])
```

```
In [25]: # Creando función que retorne los valores predichos versus los valores reales
         _, axes = plt.subplots(nrows=1, ncols=10, figsize=(15, 15))
         for ax, image, label in zip(axes, x_test[0:1000], predicted):
             ax.set_axis_off()
             image = image.reshape(28, 28)
             ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
             ax.set_title('Etiqueta: %i' % label)
```



Análisis: se puede observar como utilizando una cantidad de 1000 "imágenes" para entrenar el modelo, max_depth=5, random_state=0, n_jobs=2, n_estimators=500, criterion = 'gini', max_features= 'auto', se obtienen resultados no tan buenos, ya que el algoritmo no fue capaz de identificar correctamente casi todos los dígitos y se equivocó en 3 de ellos (confundió un 2 con un 6, un 9 con un 4 y un 5 con un 9).

```
In [26]: # Imprimiendo resultados para ver el accuracy, precision y recall
         print(
             f"Classification report for classifier {rf_model_sec}:\n"
             f"{metrics.classification_report(y_test[0:1000], predicted)}\n"
         )
```

```
Classification report for classifier RandomForestClassifier(max_depth=5, n_estimators=500, n_jobs=2,
random_state=0):
              precision    recall  f1-score   support

           0       0.85      0.94      0.89        85
           1       0.86      0.99      0.92       126
           2       0.89      0.72      0.80       116
           3       0.74      0.78      0.76       107
           4       0.77      0.84      0.80       110
           5       0.89      0.55      0.68        87
           6       0.86      0.84      0.85        87
           7       0.76      0.82      0.79        99
           8       0.81      0.71      0.75        89
           9       0.71      0.85      0.78        94

    accuracy                           0.81      1000
   macro avg       0.81      0.80      0.80      1000
weighted avg       0.81      0.81      0.81      1000
```

Análisis: a nivel de precisión con un n_train de 6000, podemos ver que los digitos que tienen el score más alto son los dígitos 2 y 5, ambos con un 89% de acierto, es decir, el 89% de las veces el algoritmo será capaz de idenficar los números 2 y 5, por ende se equivocará un 11% de las veces, mientras que la más baja es la de 9 con un 71%, es decir, para el algoritmo es tarea fácil identificar el 2 y 5, pero no así el 9 (tiende a cometer más error tipo I) con los demás tiene un rango aceptable de aciertos.

A nivel del recall podemos ver que el mayor es el dígito 1 con un 99%, es decir, solo se equivocaría en un 1% de las veces, mientras que el más bajo es el dígito 5 con un 55%, es decir para el dígito 5 conlleva una mayor cantidad de FN que el algoritmo predice incorrectamente (error tipo II).

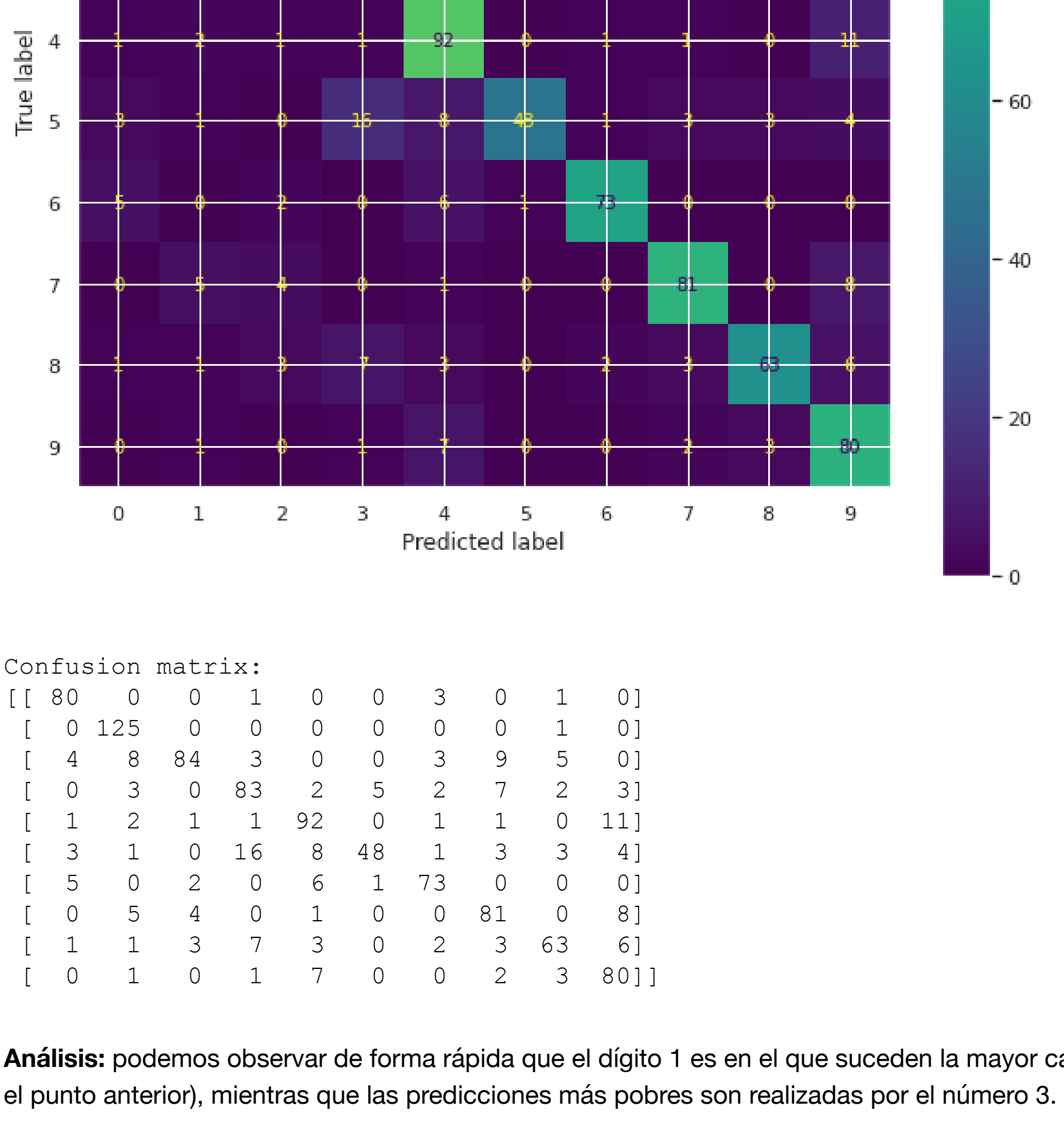
Finalmente podemos ver como el F1 obtenemos que el que tiene mayores problemas idenficando es el dígito 5 con un 68%, mientras que el que idenfica de forma bastante buena es el 1 con un 92% respectivamente, los demás se mantienen en un rango entre 70 y 90%.

```
In [27]: # Creando visualizacion de la matriz de confusion
         print(f"Confusion matrix:\n")
         fig, ax = plt.subplots(figsize=(10, 10))
         metrics.plot_confusion_matrix(rf_model_sec, x_test[0:1000], y_test[0:1000], cmap=plt.cm.viridis, ax=ax)
         plt.show()

         print("\n")

         print(f"Confusion matrix:\n{confusion_matrix(y_test[0:1000], predicted)}")
```

```
Confusion matrix:
```



```
Confusion matrix:
[[ 80   0   0   1   0   0   3   0   1   0]
 [  0 125   0   0   0   0   0   0   1   0]
 [  4   8  84   3   0   0   3   9   5   0]
 [  0   3   0  83   2   5   2   7   2   3]
 [  1   2   1   1  92   0   1   1   0  11]
 [  3   1   0  16   8  48   1   3   3   4]
 [  5   0   2   0   6   1  73   0   0   0]
 [  0   5   4   0   1   0   0  81   0   8]
 [  1   1   3   7   3   0   2   3  63   6]
 [  0   1   0   1   7   0   0   2   3  80]]
```

Análisis: podemos observar de forma rápida que el dígito 1 es en el que suceden la mayor cantidad de aciertos (tal y como indicamos en el punto anterior), mientras que las predicciones más pobres son realizadas por el número 3. Estos resultados no distan de los obtenidos del modelo anterior, ya que no mejoraron considerablemente, pero se nota que en este modelo hay una mayor oportunidad de predicción de forma más correcta, ya que en el pasado se obtenían valores bastante dificientes en para el recall de 5,6 y 8 y en este solo pasa eso con el 5 y sí aumentó el recall, así como los F1 scores.

Conclusiones: se pudo observar como los random forest pueden ser muy potentes para clasificación y en la medida en que se aumentan la cantidad de datos que le ingestamos al algoritmo este obtiene mejores predicciones. Asimismo si subimos la cantidad de max_depths y n_jobs se pueden obtener mejores resultados, sin embargo hay que tener cuidado para no caer en overfitting. Para este caso se elige el segundo modelo, ya que es el que da resultados más equilibrados. También es importante ver cuántas observaciones tenemos de cada una de las etiquetas, ya que podemos tener bastantes 0s y 1s, de ahí que el algoritmo pueda predecir tan bien dichos valores, pero otros como 5, 6 y 8 puede ser que no hayan los suficientes o más bien que en el test no se contemplaron tantos, de ahí la importancia de aleatorizar los sub conjuntos para no caer en sesgos de selección.