# Olimpiadas

Santiago Gelvez Gonzalez - 2202046
Luis Andres Gonzalez Corzo - 2201493
Daniel Felipe Ballesteros Vesga - 2182786

# Procesamiento de Datos

```
#dataset = pd.read_csv("/content/drive/MyDrive/AI/dataset_olympics.csv", sep=',')
dataset = pd.read_csv("/home/luis/Downloads/dataset_olympics.csv", sep=',')
dataset.head(20)
```

| ID | Name | Sex | Age | Height | Weight | Team | NOC | Games | Year | Season | City | Sport | Event | Medal |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A Dijiang | M | 24.0 | 180.0 | 80.0 | China | CHN | 1992 Summer | 1992 | Summer | Barcelona | Basketball | Basketball Men's Basketball | NaN |
| 2 | A Lamusi | M | 23.0 | 170.0 | 60.0 | China | CHN | 2012 Summer | 2012 | Summer | London | Judo | Judo Men's Extra-Lightweight | NaN |
| 3 | Gunnar Nielsen Aaby | M | 24.0 | NaN | NaN | Denmark | DEN | 1920 Summer | 1920 | Summer | Antwerpen | Football | Football Men's Football | NaN |
| 4 | Edgar Lindenau Aabye | M | 34.0 | NaN | NaN | Denmark/Sweden | DEN | 1900 Summer | 1900 | Summer | Paris | Tug-Of-War | Tug-Of-War Men's Tug-Of-War | Gold |
| 5 | Christine Jacoba Aaftink | F | 21.0 | 185.0 | 82.0 | Netherlands | NED | 1988 Winter | 1988 | Winter | Calgary | Speed Skating | Speed Skating Women's 500 metres | NaN |
| 5 | Christine Jacoba Aaftink | F | 21.0 | 185.0 | 82.0 | Netherlands | NED | 1988 Winter | 1988 | Winter | Calgary | Speed Skating | Speed Skating Women's 1,000 metres | NaN |

```python
dataset = dataset.fillna(0)
dataset.replace({'Sex':{'M': 0, 'F': 1}}, inplace=True)
dataset.replace({'Medal':{'Bronze': 1, 'Silver': 2, 'Gold': 3}}, inplace=True)
dataset.replace({'Season':{'Summer': 0, 'Winter': 1}}, inplace=True)
k = 0
for i in dataset.City.unique():
  dataset.City.replace(i, k, inplace = True)
  k = k+1
k = 0
for i in dataset.Sport.unique():
  dataset.Sport.replace(i, k, inplace = True)
  k = k+1
k = 0
for i in dataset.Games.unique():
  dataset.Games.replace(i, k, inplace = True)
  k = k+1
k = 0
for i in dataset.Event.unique():
  dataset.Event.replace(i, k, inplace = True)
  k = k+1
k = 0
for i in dataset.Team.unique():
  dataset.Team.replace(i, k, inplace = True)
  k = k+1
k = 0
for i in dataset.NOC.unique():
  dataset.NOC.replace(i, k, inplace = True)
  k = k+1
dataset.drop('Name', inplace=True, axis=1)
#dataset.drop('Sport', inplace=True, axis=1)
#dataset.drop('Event', inplace=True, axis=1)
#dataset.drop('NOC', inplace=True, axis=1)
dataset.head(10)
```

# Dataset procesado:

| | ID | Sex | Age | Height | Weight | Team | NOC | Games | Year | Season | City | Sport | Event | Medal |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 24.0 | 180.0 | 80.0 | 0 | 0 | 0 | 1992 | 0 | 0 | 0 | 0 | 0 |
| **1** | 2 | 0 | 23.0 | 170.0 | 60.0 | 0 | 0 | 1 | 2012 | 0 | 1 | 1 | 1 | 0 |
| **2** | 3 | 0 | 24.0 | 0.0 | 0.0 | 1 | 1 | 2 | 1920 | 0 | 2 | 2 | 2 | 0 |
| **3** | 4 | 0 | 34.0 | 0.0 | 0.0 | 2 | 1 | 3 | 1900 | 0 | 3 | 3 | 3 | 3 |
| **4** | 5 | 1 | 21.0 | 185.0 | 82.0 | 3 | 2 | 4 | 1988 | 1 | 4 | 4 | 4 | 0 |
| **5** | 5 | 1 | 21.0 | 185.0 | 82.0 | 3 | 2 | 4 | 1988 | 1 | 4 | 4 | 5 | 0 |
| **6** | 5 | 1 | 25.0 | 185.0 | 82.0 | 3 | 2 | 5 | 1992 | 1 | 5 | 4 | 4 | 0 |
| **7** | 5 | 1 | 25.0 | 185.0 | 82.0 | 3 | 2 | 5 | 1992 | 1 | 5 | 4 | 5 | 0 |
| **8** | 5 | 1 | 27.0 | 185.0 | 82.0 | 3 | 2 | 6 | 1994 | 1 | 6 | 4 | 4 | 0 |
| **9** | 5 | 1 | 27.0 | 185.0 | 82.0 | 3 | 2 | 6 | 1994 | 1 | 6 | 4 | 5 | 0 |

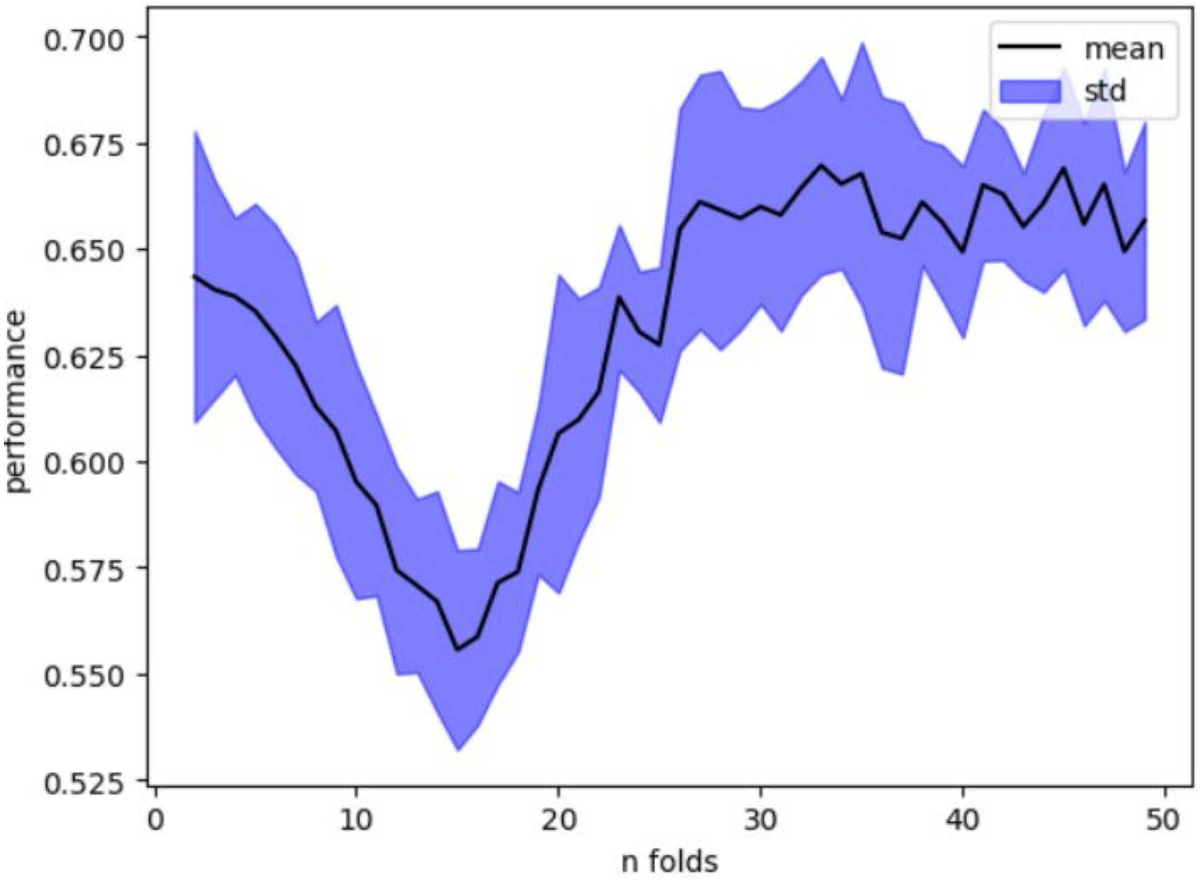

ID 1.00 1.00 0.02 0.01 -0.01 -0.02 0.04 0.01 0.01 -0.03 0.02 0.00 0.02 0.05 0.01
Name 1.00 1.00 0.02 0.01 -0.01 -0.02 0.04 0.01 0.01 -0.03 0.02 0.00 0.02 0.05 0.01
Sex 0.02 0.02 1.00 -0.07 0.13 -0.00 -0.04 0.00 -0.10 0.28 0.03 -0.04 -0.04 0.28 0.01
Age 0.01 0.01 -0.07 1.00 0.08 0.11 0.01 -0.01 -0.07 0.10 0.02 -0.02 0.08 0.07 0.04
Height -0.01 -0.01 0.13 0.08 1.00 0.90 -0.06 0.04 -0.01 0.65 0.03 0.12 -0.02 -0.02 0.00
Weight -0.02 -0.02 -0.00 0.11 0.90 1.00 -0.04 0.05 -0.02 0.63 0.03 0.11 -0.03 -0.06 0.01
Team 0.04 0.04 -0.04 0.01 -0.06 -0.04 1.00 0.67 0.01 -0.04 0.00 0.00 -0.01 0.05 -0.01
NOC 0.01 0.01 0.00 -0.01 0.04 0.05 0.67 1.00 0.00 0.10 -0.01 0.01 -0.06 -0.03 -0.10
Games 0.01 0.01 -0.10 -0.07 -0.01 -0.02 0.01 0.00 1.00 -0.29 0.04 0.84 -0.03 -0.06 0.00
Year -0.03 -0.03 0.28 0.10 0.65 0.63 -0.04 0.10 -0.29 1.00 0.13 -0.09 0.02 0.02 -0.07
Season 0.02 0.02 0.03 0.02 0.03 0.03 0.00 -0.01 0.04 0.13 1.00 0.08 0.03 0.05 -0.03
City 0.00 0.00 -0.04 -0.02 0.12 0.11 0.00 0.01 0.84 -0.09 0.08 1.00 -0.03 -0.08 -0.02
Sport 0.02 0.02 -0.04 0.08 -0.02 -0.03 -0.01 -0.06 -0.03 0.02 0.03 -0.03 1.00 0.30 0.04
Event 0.05 0.05 0.28 0.07 -0.02 -0.06 0.05 -0.03 -0.06 0.02 0.05 -0.08 0.30 1.00 0.03
Medal 0.01 0.01 0.01 0.04 0.00 0.01 -0.01 -0.10 0.00 -0.07 -0.03 -0.02 0.04 0.03 1.00
ID Name Sex Age Height Weight Team NOC Games Year Season City Sport Event Medal
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

# Decision Tree

## Classifier

```
def show_curveDT(criterio, n, var):
    means, stds = [], []
    nfolds_range = range(2,n)
    for nfolds in nfolds_range:
        if(var):
            i = nfolds
        else:
            i = 10
        est = DecisionTreeClassifier(max_depth=nfolds, criterion=criterio)
        s = cross_val_score(est, X, Y, cv=KFold(i, shuffle=True), scoring=make_scorer(mean_squared_error))
        means.append(np.mean(s))
        stds.append(np.std(s))

    means = np.r_[means]
    stds  = np.r_[stds]

    plt.plot(nfolds_range, means, label="mean", color="black")
    plt.fill_between(nfolds_range, means-stds, means+stds, color="blue", alpha=.5, label="std")
    plt.xlabel("n folds")
    plt.ylabel("performance")
    plt.legend()
```

```
show_curveDT('gini', 50, False)
```



# Deep Learning

- 80% de los datos para train
- 20% de los datos para test

7]:

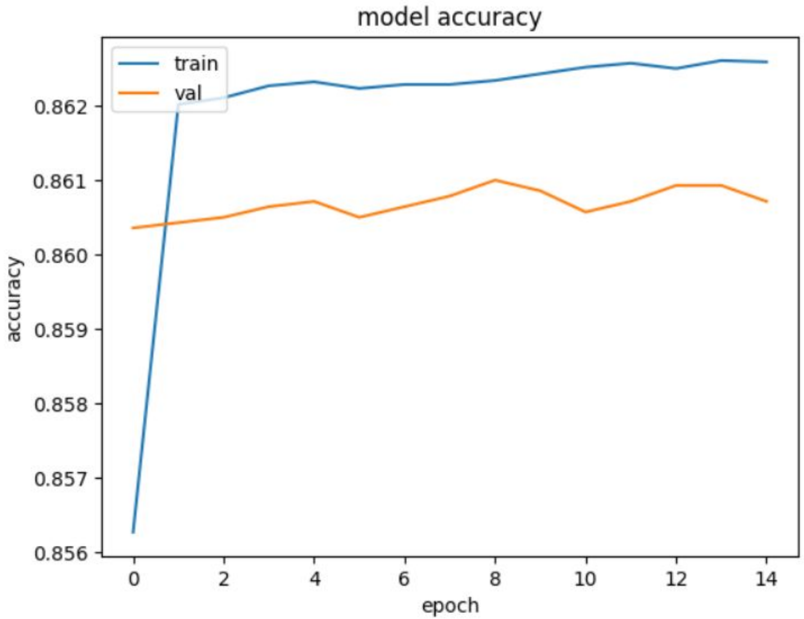
```
X_train, X_test, y_train, y_test = train_test_split(X_normalized, Y, test_size = 0.2, shuffle = True, random_state = 68)

print('Cantidad y dimensión de los datos de: \nEntrenamiento: {} \nTest: {}'.format(X_train.shape, X_test.shape))
```

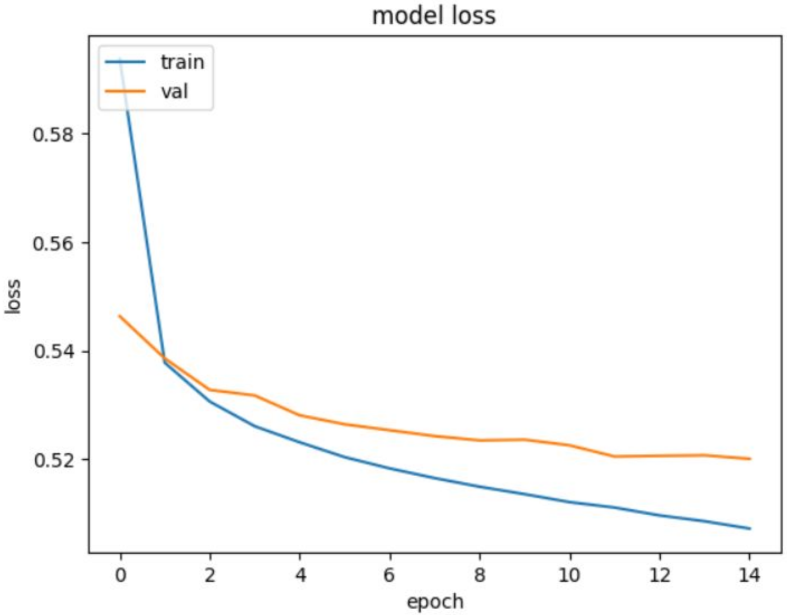
```
Cantidad y dimensión de los datos de:
Entrenamiento: (56000, 13)
Test: (14000, 13)
```

# One-Hot encoding

```
In [8]:  np.unique(y_train).shape[0]

Out[8]:  4

In [9]:  y_train_ohe = tf.keras.utils.to_categorical(y_train, num_classes=4)
         y_test_ohe = tf.keras.utils.to_categorical(y_test, num_classes=4)
         print(y_train_ohe.shape, y_test_ohe.shape)

         (56000, 4) (14000, 4)

In [10]: y_train_ohe[4528]

Out[10]: array([1., 0., 0., 0.], dtype=float32)

In [12]: X_train.shape[1:]

Out[12]: (13,)
```

# Entrenamiento de la red



# Evaluación del modelo

```
test_loss, test_acc = model.evaluate(X_test, y_test_ohe)

probs = model.predict(X_test)
preds = np.argmax(probs, axis=1)
```

```
438/438 [==============================] - 1s 2ms/step - loss: 0.5201 - accuracy: 0.8607
438/438 [==============================] - 1s 1ms/step
```

**Evaluación del Conjunto de Prueba**

- Pérdida en el conjunto de prueba (test loss): 0.5226
- Precisión en el conjunto de prueba (test accuracy): 86.06%

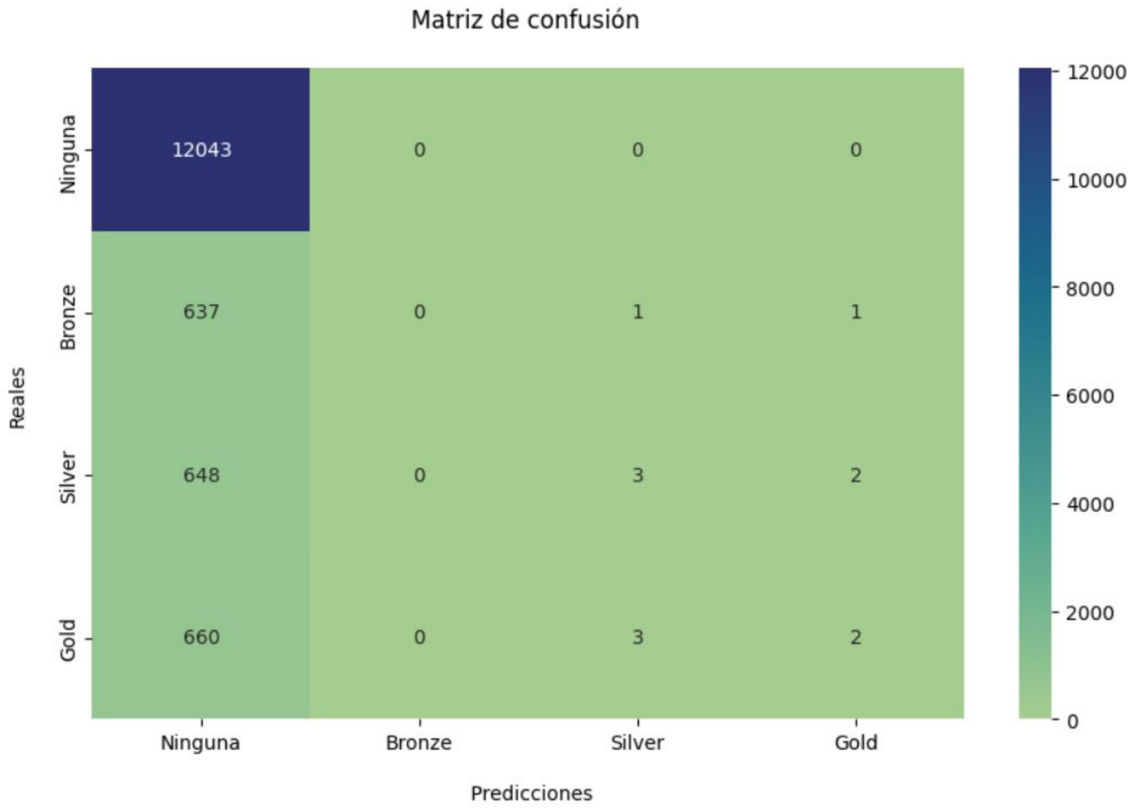
No son resultados tan malos pero pudo haber existido mayor perdida para demostrar que efectivamente hubo un buen entrenamiento.

```
predictions = model.predict(X_test)
print(predictions[2000])
print("valor predicho:",  np.argmax(predictions[523]), "max prob: ", np.max(predictions[523]),
      "ground truth: ", y_test[523] )
```

```
438/438 [==============================] - 1s 3ms/step
[0.9015981  0.04438191 0.03054353 0.02347634]
valor predicho: 0 max prob:  0.84978336 ground truth:  0.0
```

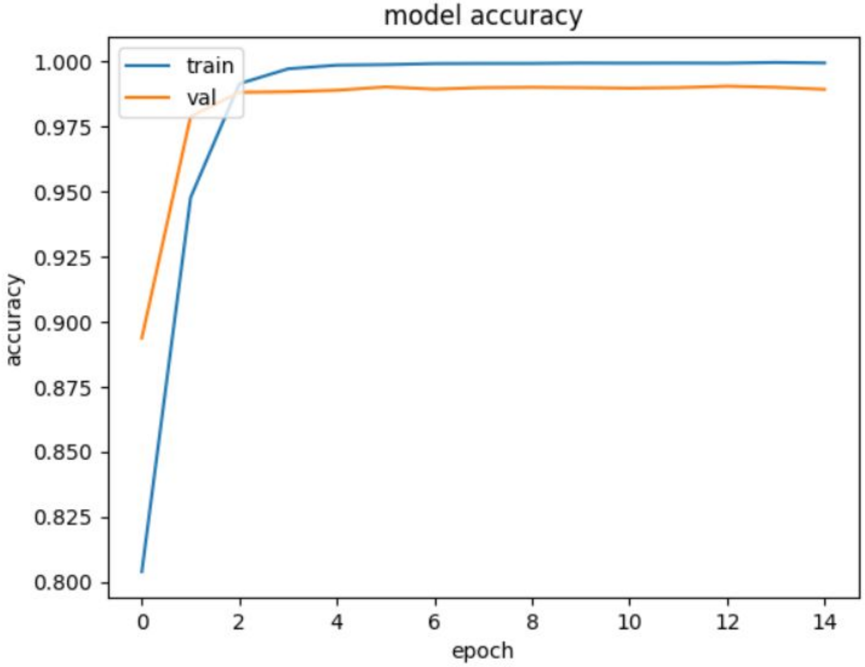
En este caso hizo la predicción que era la clase 0 y efectivamente ese registro de test pertenecía a la clase 0, además lo hizo con mucha seguridad o confianza, y esto lo vemos con más ejemplos de verdaderos positivos para la clase 0. El problema radica cuando prediga un registro como la clase 0 cuando realmente es otra totalmente diferente (falso positivo), esto posiblemente también lo haga con mucha seguridad.

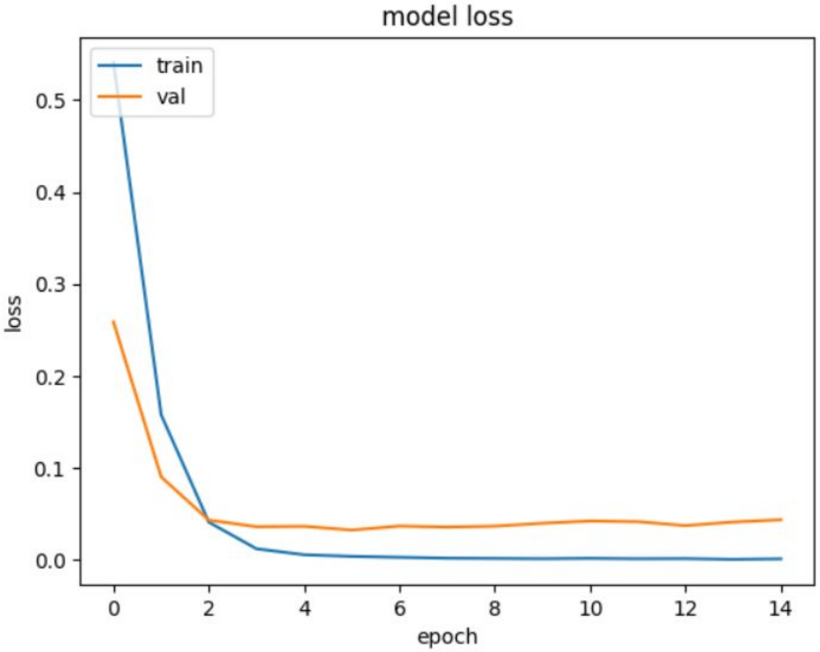
# Matriz de Confusión



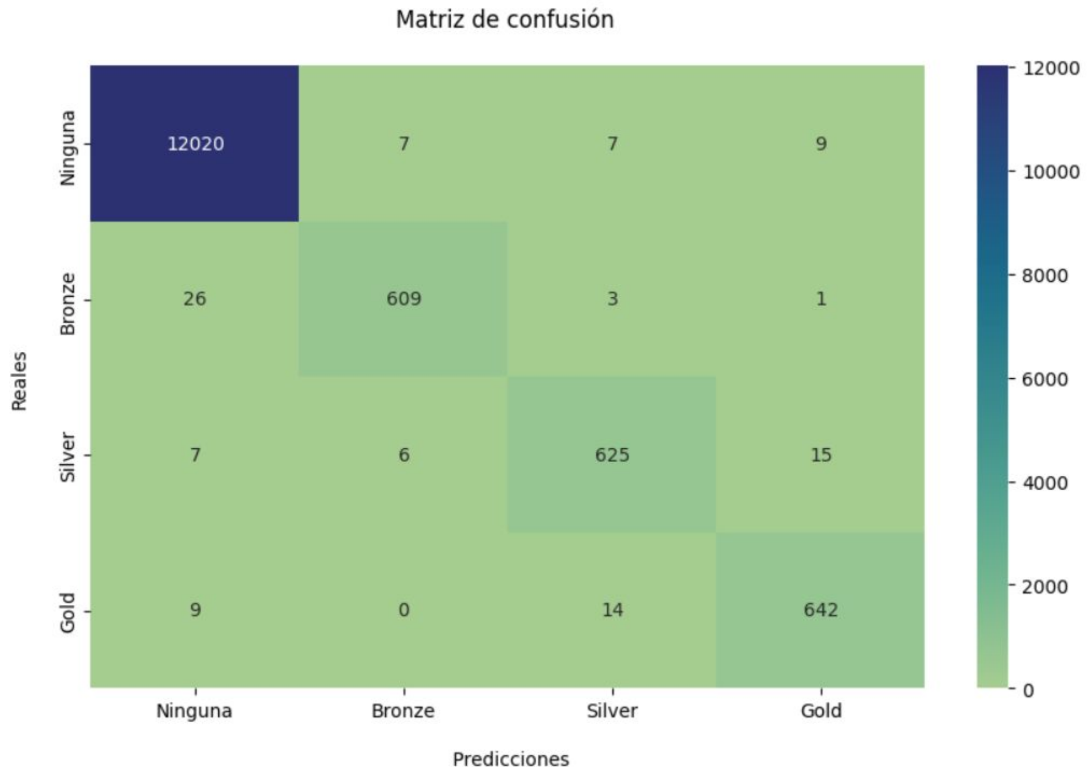
# Aplicando OHE

```
Epoch 1/15
1750/1750 [==============================] - 16s 8ms/step - loss: 0.3934 - accuracy: 0.8791 - val_loss: 0.2363 - val_accurac
y: 0.9067
Epoch 2/15
1750/1750 [==============================] - 13s 7ms/step - loss: 0.1429 - accuracy: 0.9427 - val_loss: 0.1144 - val_accurac
y: 0.9735
Epoch 3/15
1750/1750 [==============================] - 13s 8ms/step - loss: 0.0452 - accuracy: 0.9908 - val_loss: 0.0640 - val_accurac
y: 0.9861
Epoch 4/15
1750/1750 [==============================] - 11s 6ms/step - loss: 0.0146 - accuracy: 0.9971 - val_loss: 0.0476 - val_accurac
y: 0.9900
Epoch 5/15
1750/1750 [==============================] - 12s 7ms/step - loss: 0.0067 - accuracy: 0.9983 - val_loss: 0.0412 - val_accurac
y: 0.9910
Epoch 6/15
1750/1750 [==============================] - 14s 8ms/step - loss: 0.0048 - accuracy: 0.9986 - val_loss: 0.0398 - val_accurac
y: 0.9917
Epoch 7/15
1750/1750 [==============================] - 13s 8ms/step - loss: 0.0031 - accuracy: 0.9991 - val_loss: 0.0364 - val_accurac
y: 0.9920
Epoch 8/15
1750/1750 [==============================] - 11s 7ms/step - loss: 0.0025 - accuracy: 0.9993 - val_loss: 0.0385 - val_accurac
y: 0.9922
Epoch 9/15
1750/1750 [==============================] - 12s 7ms/step - loss: 0.0023 - accuracy: 0.9992 - val_loss: 0.0361 - val_accurac
y: 0.9918
Epoch 10/15
1750/1750 [==============================] - 14s 8ms/step - loss: 0.0020 - accuracy: 0.9993 - val_loss: 0.0386 - val_accurac
y: 0.9914
Epoch 11/15
1750/1750 [==============================] - 14s 8ms/step - loss: 0.0015 - accuracy: 0.9995 - val_loss: 0.0375 - val_accurac
y: 0.9923
```

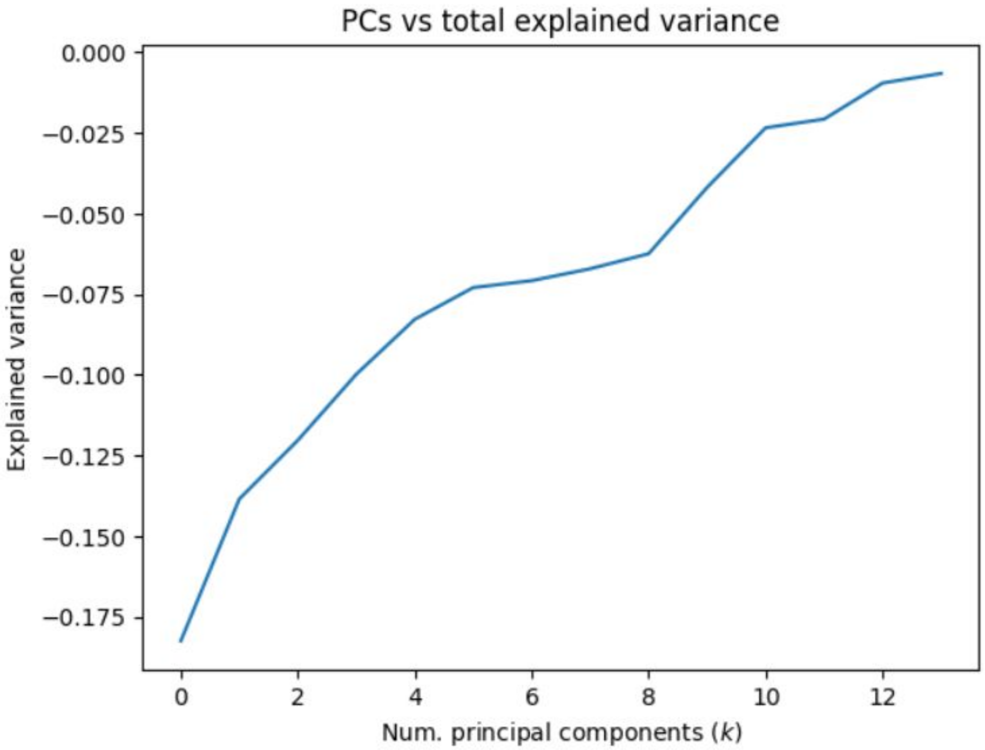

model accuracy
train
val
accuracy
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0 2 4 6 8 10 12 14
epoch


model loss
train
val
loss
0.5
0.4
0.3
0.2
0.1
0.0
0 2 4 6 8 10 12 14
epoch

# Matriz de Confusión



# PCA



PCs vs total explained variance
0.000
−0.025
−0.050
−0.075
−0.100
−0.125
−0.150
−0.175
Explained variance
0
2
4
6
8
10
12
Num. principal components ($k$)