CLASIFICADOR DE POSICIONES DE JUGADORES DE LA NBA

Julian gaitan
Santiago Gomez
Erik Vargas

Universidad Industrial de Santander

Objetivo

-Crear un clasificador capaz de identificar la posición en la que juega un jugador de la nba dependiendo de sus estadísticas personales dentro de la cancha.

2022-2023 NBA Player Stats

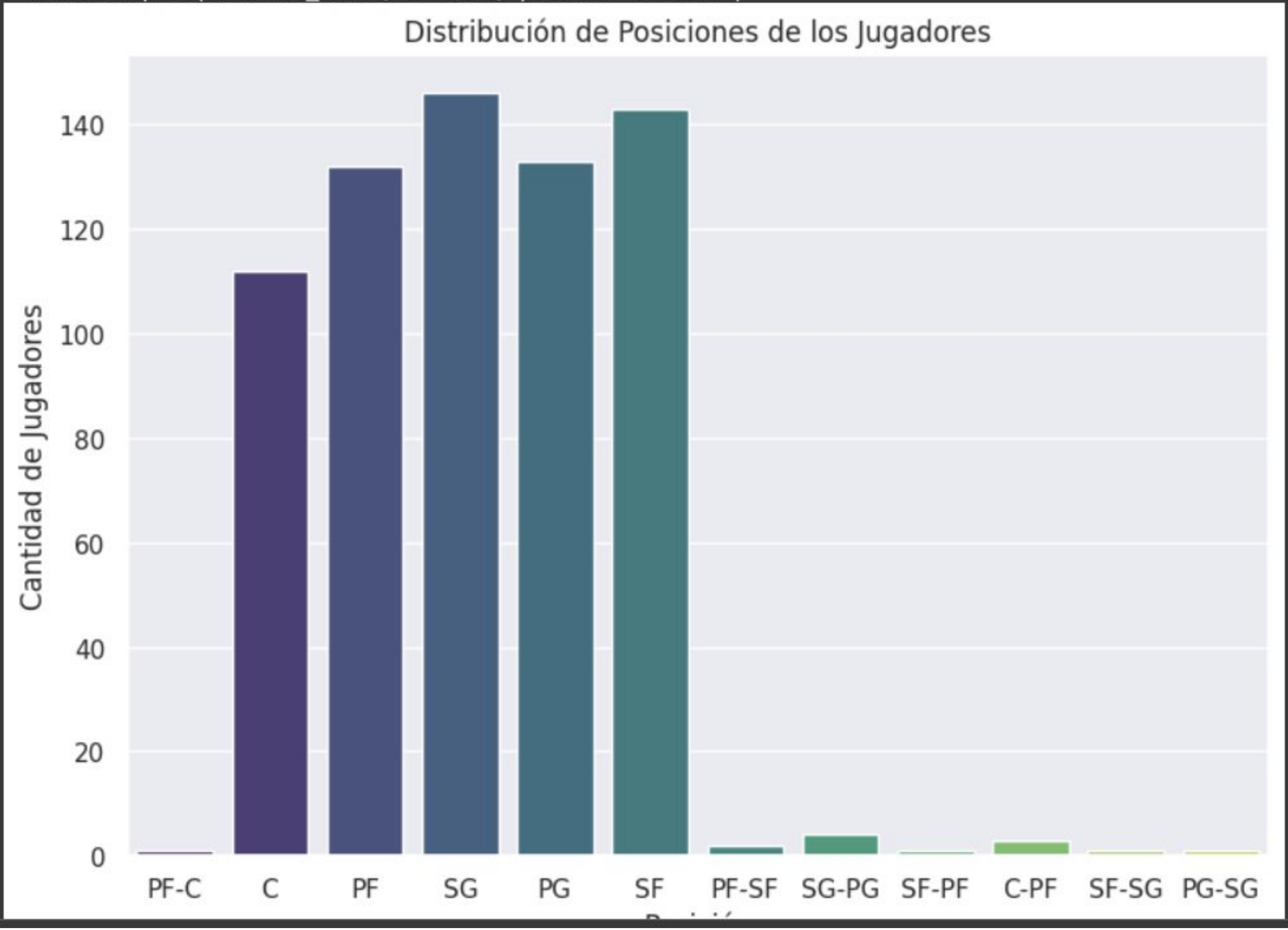
2022-2023 Regular Season NBA Player Stats

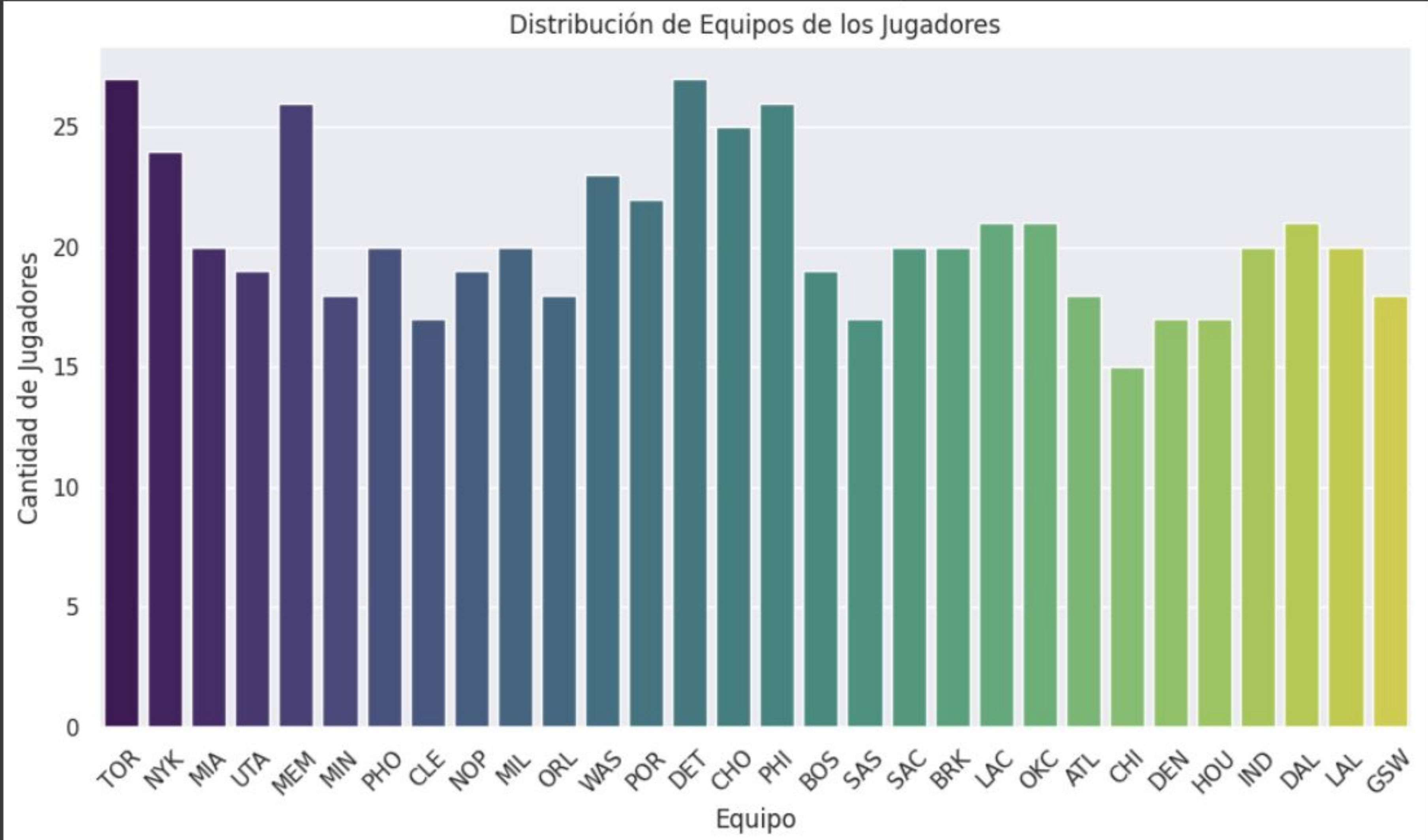
Ver en: Kaggle | kaggle.com

+500 rows and 30 columns. Columns' description are listed below.

- Rk : Rank
- Player : Player's name
- Pos : Position
- Age : Player's age
- Tm : Team
- G : Games played
- GS : Games started
- MP : Minutes played per game
- FG : Field goals per game
- FGA : Field goal attempts per game
- FG% : Field goal percentage
- 3P : 3-point field goals per game
- 3PA : 3-point field goal attempts per game
- 3P% : 3-point field goal percentage
- 2P : 2-point field goals per game
- 2PA : 2-point field goal attempts per game
- 2P% : 2-point field goal percentage
- eFG% : Effective field goal percentage
- FT : Free throws per game
- FTA : Free throw attempts per game
- FT% : Free throw percentage
- ORB : Offensive rebounds per game
- DRB : Defensive rebounds per game
- TRB : Total rebounds per game
- AST : Assists per game
- STL : Steals per game
- BLK : Blocks per game
- TOV : Turnovers per game
- PF : Personal fouls per game
- PTS : Points per game

```
sns.countplot(data=nba_stats, x='Pos', palette='viridis')
```

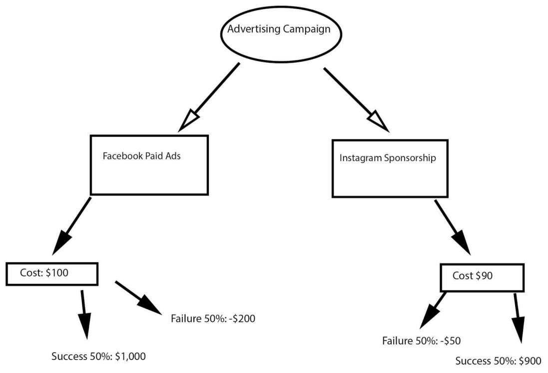




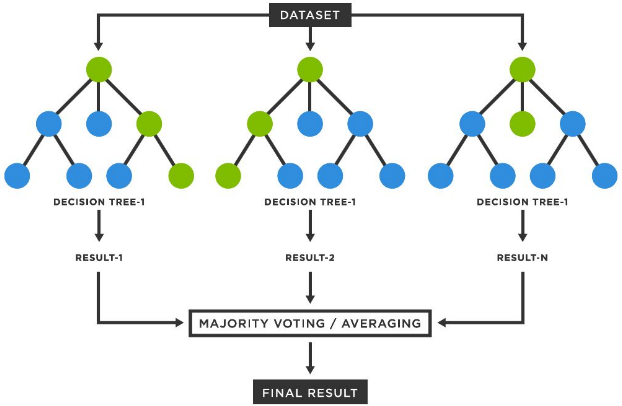
Distribución de Equipos de los Jugadores
Cantidad de Jugadores
0
5
10
15
20
25
TOR
NYK
MIA
UTA
MEM
MIN
PHO
CLE
NOP
MIL
ORL
WAS
POR
DET
CHO
PHI
BOS
SAS
SAC
BRK
LAC
OKC
ATL
CHI
DEN
HOU
IND
DAL
LAL
GSW
Equipo

Modelos utilizados

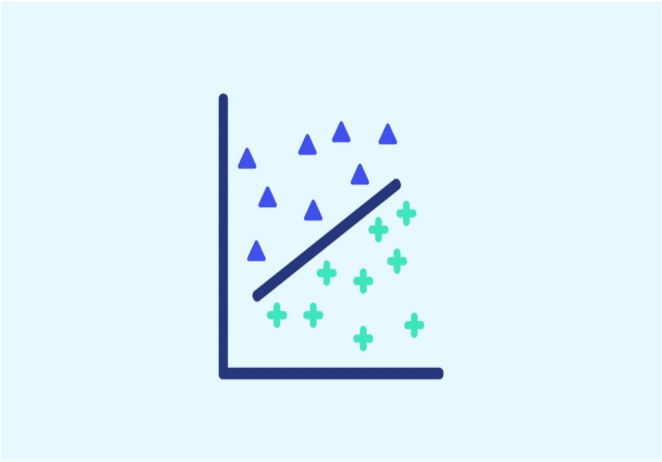
Desicion Tree



Ramdom Forest



Suport Vector Machine



Preprocesamiento

```python
#@title **Preprocesamiento**

# Cargar el dataset
nba_stats = pd.read_csv('data/NBA.csv', sep=';', encoding='latin-1')

# Aplicar OneHotEncoder a la columna categórica 'Tm' y 'Player' usando pd.get_dummies
nba_stats = pd.get_dummies(nba_stats, columns=['Tm', 'Player'])

# Contar el número de ejemplos por clase
class_counts = nba_stats['Pos'].value_counts()

# Definir un umbral para eliminar clases con pocos ejemplos
threshold = 10  # Puedes ajustar este valor según sea necesario

# Eliminar las clases con pocos ejemplos
to_remove = class_counts[class_counts < threshold].index
print(to_remove)
nba_stats = nba_stats[~nba_stats['Pos'].isin(to_remove)]

```

```
Index(['SG-PG', 'C-PF', 'PF-SF', 'PF-C', 'SF-PF', 'SF-SG', 'PG-SG'], dtype='object', name='Pos')
```

Particionado

```python
#@title **Particionado**
# Definir las características (X) y la variable objetivo (y)
X = nba_stats.drop(columns=['Pos'])
y = nba_stats['Pos']

# Dividir el dataset en conjuntos de entrenamiento y prueba (90/10)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Decision Tree

Decision tree: parametros por defecto

```python
# @title **Decision tree: parametros por defecto**
est = DecisionTreeClassifier()
est.fit(X_train,y_train)
accuracy = accuracy_score(est.predict(X_test), y_test)
print(f"Accuracy con parámetros por defecto: {accuracy}")
```

```
Accuracy con parámetros por defecto: 0.5149253731343284
```

Decision tree: tunning de parametros

```python
# @title **Decision tree: tunning de parametros**

param_grid = {
    'max_depth': [None, 2, 5, 10, 20, 30, 40, 50],
    'criterion': ['gini', 'entropy']
}

grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)


best_params = grid_search.best_params_

best_est = DecisionTreeClassifier(**best_params)
best_est.fit(X_train, y_train)


tuned_accuracy = accuracy_score(y_test, best_est.predict(X_test))
print(f"Accuracy con tuning de parámetros: {tuned_accuracy}")
```

```
Accuracy con tuning de parámetros: 0.5447761194029851
```

Decision tree con Crossvalidation

```python
# @title **Decision tree con Crossvalidation**
est_default = DecisionTreeClassifier()
est_default.fit(X_train,y_train)

kf = KFold(n_splits=10, shuffle=True, random_state=42)

cv_scores = cross_val_score(est_default, X_train, y_train, cv=kf, scoring='accuracy')

mean_accuracy = np.mean(cv_scores)
std_accuracy = np.std(cv_scores)

print(f"Accuracy media con CrossValidation: {mean_accuracy}")
print(f"Desviación estándar del accuracy: {std_accuracy}")
```

```
Accuracy media con CrossValidation: 0.40213137665967863
Desviación estándar del accuracy: 0.07893675682476003
```

Ramdom State de mayor accuracy

```python
# @title **Ramdom State de mayor accuracy**
best_accuracy = 0
best_random_state = None

# Probar diferentes random_states
for random_state in range(100):  # Puedes ajustar el rango según tus necesidades
    est = DecisionTreeClassifier(random_state=random_state)
    est.fit(X_train, y_train)
    accuracy = accuracy_score(est.predict(X_test), y_test)

    # Actualizar el mejor accuracy y random_state si el actual es mejor
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_random_state = random_state

# Imprimir el mejor random_state y su accuracy
print(f"Mejor Random State: {best_random_state}")
print(f"Mejor Accuracy: {best_accuracy}")
```

```
Mejor Random State: 55
Mejor Accuracy: 0.5970149253731343
```

Random Forest

Random Forest: parametros por defecto

```python
# @title **Random Forest: parametros por defecto**
est = RandomForestClassifier()
est.fit(X_train,y_train)

print(accuracy_score(est.predict(X_test), y_test))
```

```
0.5597014925373134
```

Random Forest: tunning de parametros

```python
#@title **Random Forest: tunning de parametros**

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'criterion': ['gini', 'entropy']
}

grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_est = RandomForestClassifier(**best_params)
best_est.fit(X_train, y_train)

tuned_accuracy = accuracy_score(y_test, best_est.predict(X_test))
print(f"Accuracy con tuning de parámetros: {tuned_accuracy}")
```

```
Accuracy con tuning de parámetros: 0.6119402985074627
```

Ramdom state de mayor accuracy

```python
# @title **Ramdom state de mayor accuracy**
best_accuracy = 0
best_random_state = None

# Probar diferentes random_states
for random_state in range(100):  # Puedes ajustar el rango según tus necesidades
    est = RandomForestClassifier(random_state=random_state)
    est.fit(X_train, y_train)
    accuracy = accuracy_score(est.predict(X_test), y_test)

    # Actualizar el mejor accuracy y random_state si el actual es mejor
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_random_state = random_state

# Imprimir el mejor random_state y su accuracy
print(f"Mejor Random State: {best_random_state}")
print(f"Mejor Accuracy: {best_accuracy}")
```

```
Mejor Random State: 59
Mejor Accuracy: 0.664179104477612
```

Support Vector Machine

SVM: parametros por defecto

```python
#@title **SVM: parametros por defecto**
svc = SVC()
svc.fit(X_train, y_train)

# Predecir y calcular la precisión
y_pred = svc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy con SVC (parámetros por defecto): {accuracy}")


```

```
Accuracy con SVC (parámetros por defecto): 0.20149253731343283
```

SVM: tunning de parametros

```python
#@title **SVM: tunning de parametros**

# Definir los parámetros a probar en el GridSearchCV
param_grid = {'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}

# Configurar el GridSearchCV con validación cruzada de 5 pliegues
grid_search = GridSearchCV(svc, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Mostrar los resultados de todos los parámetros probados
results = grid_search.cv_results_
for mean, std, params in zip(results['mean_test_score'], results['std_test_score'], results['params']):
    print(f"Mean accuracy: {mean:.4f} (std: {std:.4f}) with parameters: {params}")

# Mejor modelo y su precisión
best_svc = grid_search.best_estimator_
best_accuracy = grid_search.best_score_
print(f"Mejor kernel: {grid_search.best_params_['kernel']}")
print(f"Mejor accuracy con tuning de parámetros: {best_accuracy}")

# Evaluar el mejor modelo en el conjunto de prueba
y_pred = best_svc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy en el conjunto de prueba con el mejor kernel: {accuracy}")
```

```
Mean accuracy: 0.5056 (std: 0.0069) with parameters: {'kernel': 'linear'}
Mean accuracy: 0.2538 (std: 0.0133) with parameters: {'kernel': 'poly'}
Mean accuracy: 0.2557 (std: 0.0130) with parameters: {'kernel': 'rbf'}
Mean accuracy: 0.1673 (std: 0.0262) with parameters: {'kernel': 'sigmoid'}
Mejor kernel: linear
Mejor accuracy con tuning de parámetros: 0.5056427437841651
Accuracy en el conjunto de prueba con el mejor kernel: 0.5447761194029851
```

Redes neuronales

1 capa oculta

```
Epoch 5/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5689 - accuracy: 0.3045
Epoch 6/30
9/9 [==============================] - 0s 4ms/step - loss: 1.5675 - accuracy: 0.3139
Epoch 7/30
9/9 [==============================] - 0s 4ms/step - loss: 1.5663 - accuracy: 0.3177
Epoch 8/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5649 - accuracy: 0.3327
Epoch 9/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5638 - accuracy: 0.3177
Epoch 10/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5626 - accuracy: 0.3289
Epoch 11/30
9/9 [==============================] - 0s 4ms/step - loss: 1.5614 - accuracy: 0.3327
Epoch 12/30
9/9 [==============================] - 0s 5ms/step - loss: 1.5601 - accuracy: 0.3308
Epoch 13/30
9/9 [==============================] - 0s 4ms/step - loss: 1.5587 - accuracy: 0.3440
Epoch 14/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5573 - accuracy: 0.3496
Epoch 15/30
9/9 [==============================] - 0s 4ms/step - loss: 1.5561 - accuracy: 0.3553
Epoch 16/30
9/9 [==============================] - 0s 4ms/step - loss: 1.5549 - accuracy: 0.3421
Epoch 17/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5537 - accuracy: 0.3440
Epoch 18/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5525 - accuracy: 0.3665
Epoch 19/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5512 - accuracy: 0.3665
Epoch 20/30
9/9 [==============================] - 0s 4ms/step - loss: 1.5501 - accuracy: 0.3853
Epoch 21/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5486 - accuracy: 0.3891
Epoch 22/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5474 - accuracy: 0.3816
Epoch 23/30
9/9 [==============================] - 0s 4ms/step - loss: 1.5462 - accuracy: 0.4023
Epoch 24/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5449 - accuracy: 0.3835
Epoch 25/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5436 - accuracy: 0.3816
Epoch 26/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5423 - accuracy: 0.3966
Epoch 27/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5411 - accuracy: 0.3966
Epoch 28/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5398 - accuracy: 0.4004
Epoch 29/30
9/9 [==============================] - 0s 3ms/step - loss: 1.5387 - accuracy: 0.3853
Epoch 30/30
9/9 [==============================] - 0s 4ms/step - loss: 1.5373 - accuracy: 0.3797
5/5 [==============================] - 0s 4ms/step
Accuracy con 30 épocas: 0.2612
```

3 capas ocultas

```
9/9 [==============================] - 0s 5ms/step - loss: 1.5337 - accuracy: 0.3835
Epoch 9/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5314 - accuracy: 0.3778
Epoch 10/30
9/9 [==============================] - 0s 8ms/step - loss: 1.5289 - accuracy: 0.3759
Epoch 11/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5261 - accuracy: 0.3872
Epoch 12/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5237 - accuracy: 0.3891
Epoch 13/30
9/9 [==============================] - 0s 9ms/step - loss: 1.5212 - accuracy: 0.3853
Epoch 14/30
9/9 [==============================] - 0s 9ms/step - loss: 1.5187 - accuracy: 0.3797
Epoch 15/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5163 - accuracy: 0.3929
Epoch 16/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5134 - accuracy: 0.3929
Epoch 17/30
9/9 [==============================] - 0s 7ms/step - loss: 1.5111 - accuracy: 0.3985
Epoch 18/30
9/9 [==============================] - 0s 8ms/step - loss: 1.5081 - accuracy: 0.4004
Epoch 19/30
9/9 [==============================] - 0s 9ms/step - loss: 1.5049 - accuracy: 0.4098
Epoch 20/30
9/9 [==============================] - 0s 7ms/step - loss: 1.5020 - accuracy: 0.4041
Epoch 21/30
9/9 [==============================] - 0s 9ms/step - loss: 1.4990 - accuracy: 0.4079
Epoch 22/30
9/9 [==============================] - 0s 5ms/step - loss: 1.4960 - accuracy: 0.4342
Epoch 23/30
9/9 [==============================] - 0s 6ms/step - loss: 1.4932 - accuracy: 0.4060
Epoch 24/30
9/9 [==============================] - 0s 5ms/step - loss: 1.4901 - accuracy: 0.4286
Epoch 25/30
9/9 [==============================] - 0s 5ms/step - loss: 1.4871 - accuracy: 0.4305
Epoch 26/30
9/9 [==============================] - 0s 5ms/step - loss: 1.4836 - accuracy: 0.4229
Epoch 27/30
9/9 [==============================] - 0s 9ms/step - loss: 1.4806 - accuracy: 0.4342
Epoch 28/30
9/9 [==============================] - 0s 9ms/step - loss: 1.4775 - accuracy: 0.4398
Epoch 29/30
9/9 [==============================] - 0s 9ms/step - loss: 1.4740 - accuracy: 0.4398
Epoch 30/30
9/9 [==============================] - 0s 8ms/step - loss: 1.4708 - accuracy: 0.4605
5/5 [==============================] - 0s 4ms/step
Accuracy con 30 épocas: 0.3731
```

6 capas ocultas

```
Epoch 8/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5973 - accuracy: 0.2425
Epoch 9/30
9/9 [==============================] - 0s 7ms/step - loss: 1.5970 - accuracy: 0.2481
Epoch 10/30
9/9 [==============================] - 0s 8ms/step - loss: 1.5967 - accuracy: 0.2500
Epoch 11/30
9/9 [==============================] - 0s 8ms/step - loss: 1.5964 - accuracy: 0.2444
Epoch 12/30
9/9 [==============================] - 0s 9ms/step - loss: 1.5959 - accuracy: 0.2519
Epoch 13/30
9/9 [==============================] - 0s 9ms/step - loss: 1.5956 - accuracy: 0.2462
Epoch 14/30
9/9 [==============================] - 0s 7ms/step - loss: 1.5952 - accuracy: 0.2519
Epoch 15/30
9/9 [==============================] - 0s 9ms/step - loss: 1.5949 - accuracy: 0.2519
Epoch 16/30
9/9 [==============================] - 0s 9ms/step - loss: 1.5945 - accuracy: 0.2538
Epoch 17/30
9/9 [==============================] - 0s 7ms/step - loss: 1.5942 - accuracy: 0.2519
Epoch 18/30
9/9 [==============================] - 0s 8ms/step - loss: 1.5938 - accuracy: 0.2538
Epoch 19/30
9/9 [==============================] - 0s 7ms/step - loss: 1.5933 - accuracy: 0.2462
Epoch 20/30
9/9 [==============================] - 0s 8ms/step - loss: 1.5930 - accuracy: 0.2538
Epoch 21/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5926 - accuracy: 0.2556
Epoch 22/30
9/9 [==============================] - 0s 7ms/step - loss: 1.5920 - accuracy: 0.2575
Epoch 23/30
9/9 [==============================] - 0s 5ms/step - loss: 1.5916 - accuracy: 0.2538
Epoch 24/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5912 - accuracy: 0.2594
Epoch 25/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5908 - accuracy: 0.2556
Epoch 26/30
9/9 [==============================] - 0s 5ms/step - loss: 1.5905 - accuracy: 0.2519
Epoch 27/30
9/9 [==============================] - 0s 6ms/step - loss: 1.5901 - accuracy: 0.2575
Epoch 28/30
9/9 [==============================] - 0s 9ms/step - loss: 1.5895 - accuracy: 0.2575
Epoch 29/30
9/9 [==============================] - 0s 7ms/step - loss: 1.5891 - accuracy: 0.2575
Epoch 30/30
9/9 [==============================] - 0s 9ms/step - loss: 1.5886 - accuracy: 0.2575
5/5 [==============================] - 0s 8ms/step
Accuracy con 30 épocas: 0.1866
```