CLUSTERING DE UNIVERSIDADES CON MACHINE LEARNING

C. Jesus Alexis Lopez Luque
Optativa 3: Ciencia de Datos
Profesor: Dr. Isabel Dominguez Jimenez

INDICE

1 INTRODUCCIÓN

En este proyecto de Machine Learning, se utilizará el conjunto de datos del QS World University Ranking 2023 para explorar la relación entre las variables que se utilizan para clasificar a las universidades en el ranking. El objetivo es desarrollar un modelo de Machine Learning que pueda agrupar a las universidades en cuatro categorías: **Excelentes, Muy buenas, Regulares y Malas**.

El conjunto de datos contiene información sobre más de 1.500 universidades de todo el mundo. Las variables que se utilizan para clasificar a las universidades incluyen:

- Factores de investigación:
- Factores de enseñanza:
- Factores de empleabilidad:

2

DATASET

EL CONJUNTO DE DATOS "WORLD UNIVERSITY RANKINGS 2022-23" DE KAGGLE CONTIENE INFORMACIÓN SOBRE MÁS DE 1.400 UNIVERSIDADES DE TODO EL MUNDO. EL CONJUNTO DE DATOS SE BASA EN LOS RANKINGS DE UNIVERSIDADES DE QS, QUE UTILIZAN OCHO INDICADORES PARA EVALUAR LA CALIDAD DE LAS UNIVERSIDADES:

- **REPUTACIÓN ACADÉMICA:** LA REPUTACIÓN DE UNA UNIVERSIDAD ENTRE ACADÉMICOS Y EMPLEADORES.
- **REPUTACIÓN DEL EMPLEADOR:** LA REPUTACIÓN DE UNA UNIVERSIDAD ENTRE EMPLEADORES.
- **RELACIÓN FACULTAD/ESTUDIANTE:** LA RELACIÓN ENTRE EL NÚMERO DE PROFESORES Y EL NÚMERO DE ESTUDIANTES.
- **PAPEL DE LA INVESTIGACIÓN:** LA CANTIDAD DE INVESTIGACIÓN QUE SE REALIZA EN UNA UNIVERSIDAD.
- **INFLUENCIA DE LA INVESTIGACIÓN:** LA INFLUENCIA DE LA INVESTIGACIÓN QUE SE REALIZA EN UNA UNIVERSIDAD.
- **CITAS PER CÁPITA:** EL NÚMERO DE CITAS POR ARTÍCULO PUBLICADO EN UNA UNIVERSIDAD.
- **INTERNACIONALIZACIÓN:** LA PROPORCIÓN DE ESTUDIANTES Y PROFESORES INTERNACIONALES EN UNA UNIVERSIDAD.

2023 QS World University Rankings.csv (173.84 kB)

Detail | Compact | Column

10 of 21 columns

Rank	institution	location c...	location	ar score	ar rank	er score
1	Massachusetts Institute of Technology (MIT)	US	United States	100	5	100
2	University of Cambridge	UK	United Kingdom	100	2	100
3	Stanford University	US	United States	100	4	100
4	University of Oxford	UK	United Kingdom	100	3	100
5	Harvard University	US	United States	100	1	100

LIMPIEZA DEL DATASET

Al ser un dataset propio de www.kaggle.com, la mayoría de estos se encuentran previamente limpiados y estructurados, siendo este uno de esos casos, pero dejándonos un poco por hacer...

Primero que nada, a raíz de lo que se verá en la sección de graficación y de la intuición (respaldándola con hechos como la matriz de covarianza), notamos la duplicidad de información en las variables rank y su respectivo score, además de unos detalles de limpieza del dataset, como sería reemplazar los valores nulos de las variables restantes sin afectar su estadística de manera significativa.

```python
#Los NaN en er score, fsr score, cpf score al contar con una menor cantidad de NaNs se reemplazaran como 0.

dfclean['er score'].fillna(0, inplace=True)
dfclean['fsr score'].fillna(0, inplace=True)
dfclean['cpf score'].fillna(0, inplace=True)

#Los Nan con respecto a ifr, isr, irn seran transformados por las media de sus datos, ya que estos se encuentran
#Relacionados con la falta de informacion de la colaboraciones internacionales de las universidades

media_ifr = dfclean['ifr score'].mean()
dfclean['ifr score'].fillna(media_ifr, inplace=True)

media_isr = dfclean['isr score'].mean()
dfclean['isr score'].fillna(media_isr, inplace=True)

media_irn = dfclean['irn score'].mean()
dfclean['irn score'].fillna(media_irn, inplace=True)

media_ger = dfclean['ger score'].mean()
dfclean['ger score'].fillna(media_ger, inplace=True)
```

```
dfclean2 = dfclean1.drop(["Rank"], axis=1)
dfclean2
```

	ar score	er score	fsr score	cpf score	ifr score	isr score	irn score	ger score
0	100.0	100.0	100.0	100.0	100.000000	90.000000	96.1	100.000000
1	100.0	100.0	100.0	92.3	100.000000	96.300000	99.5	100.000000
2	100.0	100.0	100.0	99.9	99.800000	60.300000	96.3	100.000000
3	100.0	100.0	100.0	90.0	98.800000	98.400000	99.9	100.000000
4	100.0	100.0	99.4	100.0	76.900000	66.900000	100.0	100.000000
...	...	...	...	...	...	...	...	...
1417	4.6	5.8	3.6	1.0	1.700000	26.545348	8.1	26.186809
1418	3.0	5.8	2.4	2.2	9.300000	1.300000	8.8	30.900000
1419	2.8	2.6	3.2	3.6	5.200000	2.800000	51.1	26.186809
1420	3.3	1.8	2.7	2.0	31.659517	26.545348	7.5	8.700000
1421	4.0	2.1	3.3	1.5	1.700000	7.900000	16.6	9.700000

1422 rows × 8 columns

4 GRAFICACION

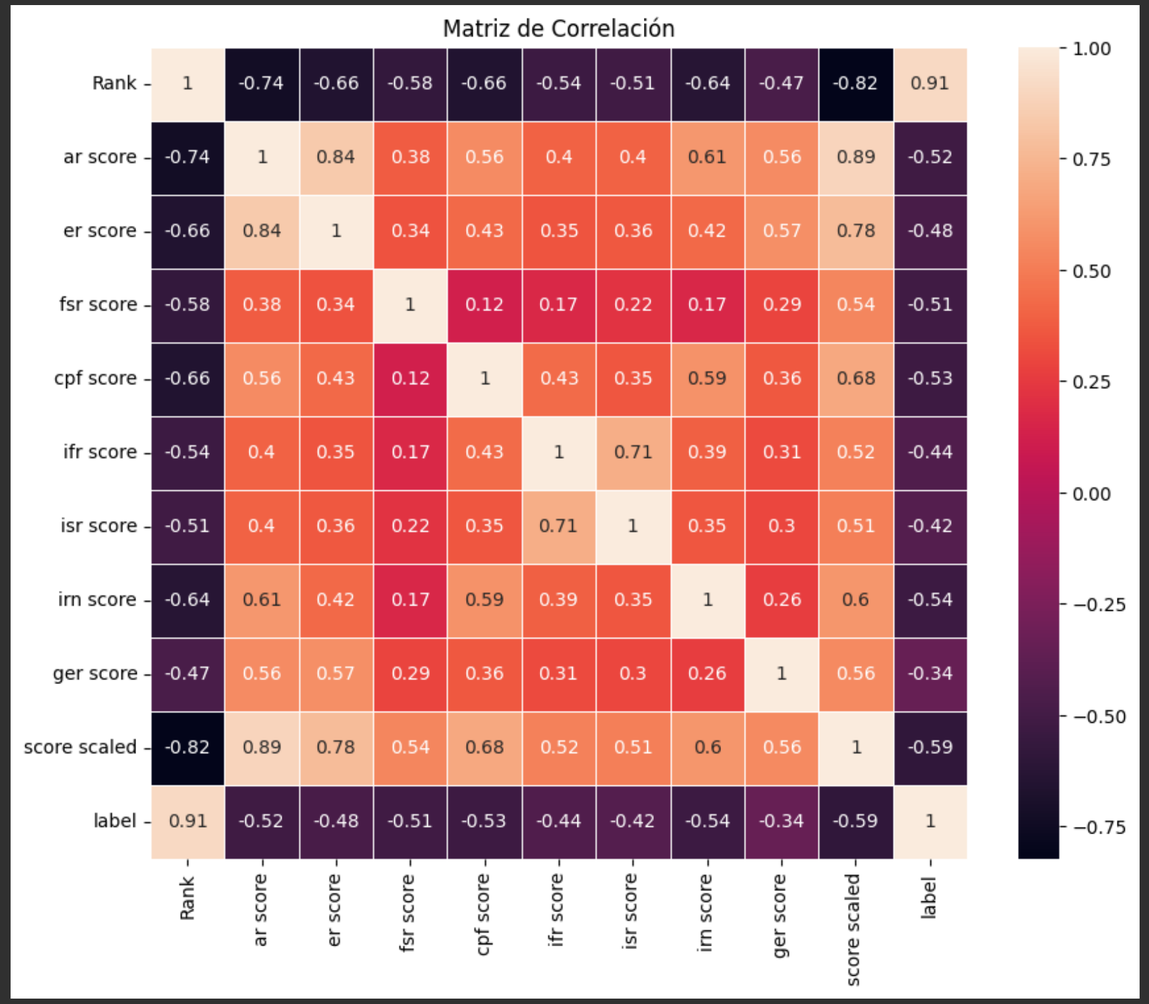


FIG 1. MATRIZ DE CORRELACION PARA DFCLEAN

4 GRAFICACION

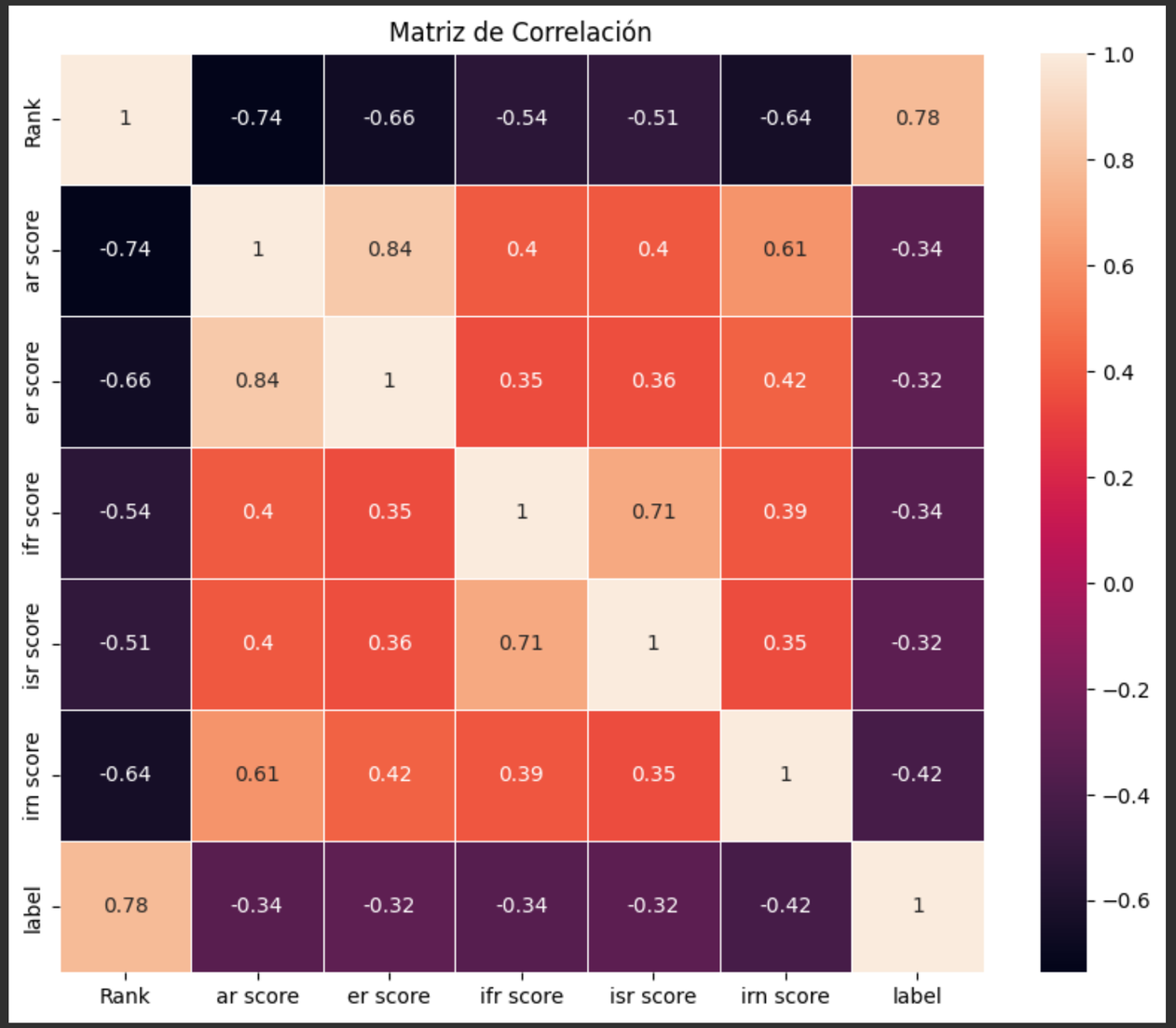


FIG 2. MATRIZ DE CORRELACION PARA PRUEBA

4 GRAFICACION

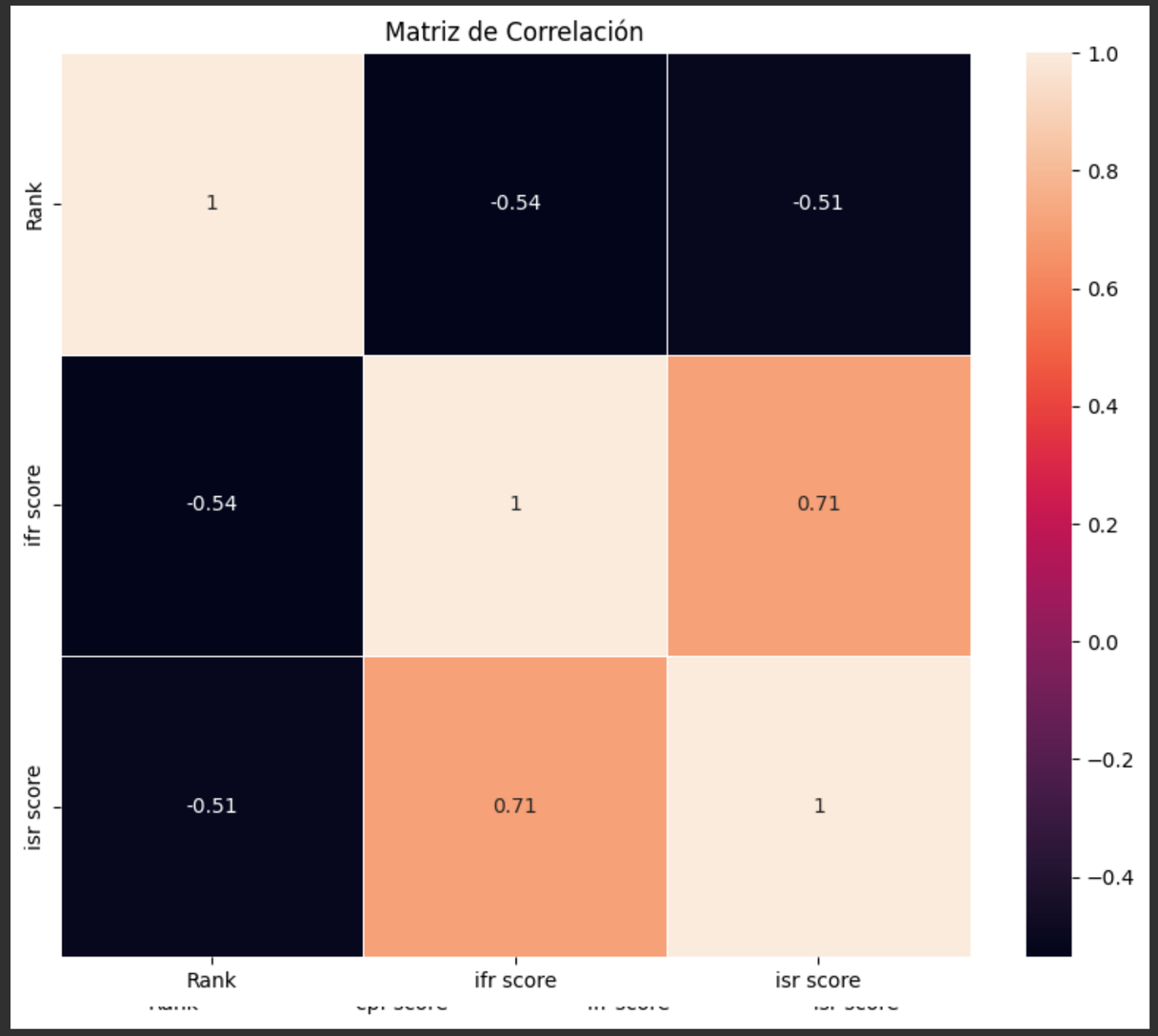


FIG 3. MATRIZ DE CORRELACION PARA DFCLEANP2

3 ARBOLES DE DECISION

Antes de crear el modelo, en esta seccion se hizo las particiones en subsets para el entrenamiento con 2 y 3 variables obtenidos a traves de f_classif y chi2 con scikitlearn y un metodo de wrapper recursivo para validar estas mismas.

Ademas de esto, en esta seccion se llevo acabo la creacion de los subsets de prueba y entrenamiento mediante la herramienta train_test_split de scikitlearn.

Creando despues el modelo de arboles de decision sencillo, utilizando el criterio de entropia, midiendo asi la impureza de los conjuntos de datos.

Se presenta el codigo del modelo:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Cargar los datos y realizar preprocesamiento básico si es necesario
# Suponiendo que tienes un DataFrame df con tus datos
# ...

# Definir intervalos y etiquetas
intervalos = [1, 350, 700, 1050, float('inf')]  # Define tus intervalos según tus criterios
etiquetas = ['Excelentes', 'Buen nivel', 'Mah o meno', 'Regular']  # Define las etiquetas par

# Crear la variable objetivo basada en los intervalos
dfcleanp1['clasificacion'] = pd.cut(dfcleanp1['Rank'], bins=intervalos, labels=etiquetas)

# Codificar la variable objetivo ('clasificacion')
le = LabelEncoder()
dfcleanp1['label'] = le.fit_transform(dfcleanp1['clasificacion'])
X= dfcleanp1[["cpf score", "ifr score", "isr score"]]
y = dfcleanp1['label']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Entrenar el modelo de árbol de decisiones
decision_tree = DecisionTreeClassifier( criterion='entropy')
decision_tree.fit(X_train, y_train)

# Predecir en el conjunto de prueba
y_pred_dt = decision_tree.predict(X_test)
```

3 REGRESION LOGISTICA

Respecto a la Regresion logistica tenemos algunas limitaciones...

Las principales limitaciones de la regresión logística para clasificar más de dos clases son las siguientes:

- Los límites de decisión lineales pueden no ser adecuados para representar los datos.
- La codificación one-hot puede aumentar el número de variables predictoras, lo que puede dificultar el entrenamiento del modelo.
- Los modelos de regresión logística multinomial pueden ser más complejos y difíciles de interpretar.

En general, la regresión logística es una herramienta útil para clasificar dos clases. Sin embargo, para clasificar más de dos clases, es necesario utilizar técnicas adicionales que pueden limitar su rendimiento o complejidad.

Sin embargo, se llevo acabo el entrenamiento para ver como se comportaba el modelo estas circustancias, aclarando que es el modelo simple, sin codificacion one hot o regresion multinomial.

Se presenta el codigo del modelo:

```python
from sklearn.linear_model import LogisticRegression

logisticRegr = LogisticRegression()
logisticRegr.fit(X_train, y_train)
# Predecir en el conjunto de prueba
y_pred_logistic = logisticRegr.predict(X_test)

# Evaluar el modelo
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
print("Precisión:", accuracy_logistic)

# Otras métricas de evaluación
print("Reporte de clasificación:")
print(classification_report(y_test, y_pred_logistic))

print("Matriz de confusión:")
print(confusion_matrix(y_test, y_pred_logistic))
```

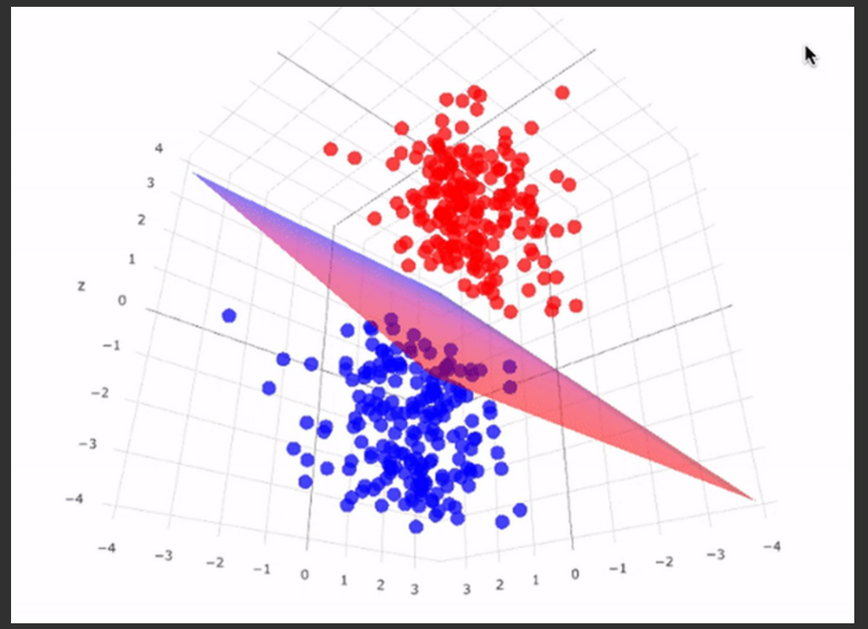
```
Precisión: 0.8245614035087719
Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.77      0.81      0.79        72
           1       0.88      0.94      0.91        67
           2       0.77      0.67      0.71        69
           3       0.88      0.88      0.88        77
           4       0.00      0.00      0.00         0

    accuracy                           0.82       285
   macro avg       0.66      0.66      0.66       285
weighted avg       0.83      0.82      0.82       285

Matriz de confusión:
[[58  8  5  0  1]
 [ 4 63  0  0  0]
 [13  1 46  9  0]
 [ 0  0  9 68  0]
 [ 0  0  0  0  0]]
```

Fig 4. Muestra de esta limitacion en el reporte generado del modelo de regresion logistica y codigo

3

SUPPORT VECTOR MACHINE

Las ***Support Vector Machine*** son uno de los modelos más robustos a la hora de clasificación. De manera abstracta, buscan el hiperplano que separa las clases. Claro que, al aumentar la dimensionalidad, esto aumenta su complejidad de manera increíble. De hecho, es para el entrenamiento: O(n^3) o O(n^2) y para la predicción: O(n)



Se presenta el codigo del modelo:

```python
from sklearn.svm import SVC

# Entrenar el modelo SVM
svm = SVC(kernel='rbf')
svm.fit(X_train, y_train)



# Predecir en el conjunto de prueba
y_pred_svm = svm.predict(X_test)

# Evaluar el modelo
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print("Precisión:", accuracy_svm)

# Otras métricas de evaluación
print("Reporte de clasificación:")
print(classification_report(y_test, y_pred_svm))

print("Matriz de confusión:")
print(confusion_matrix(y_test, y_pred_svm))
```

```
Precisión: 0.8280701754385965
Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.79      0.79      0.79        72
           1       0.88      0.90      0.89        67
           2       0.76      0.74      0.75        69
           3       0.87      0.88      0.88        77

    accuracy                           0.83       285
   macro avg       0.83      0.83      0.83       285
weighted avg       0.83      0.83      0.83       285

Matriz de confusión:
[[57  8  7  0]
 [ 7 60  0  0]
 [ 8  0 51 10]
 [ 0  0  9 68]]
```

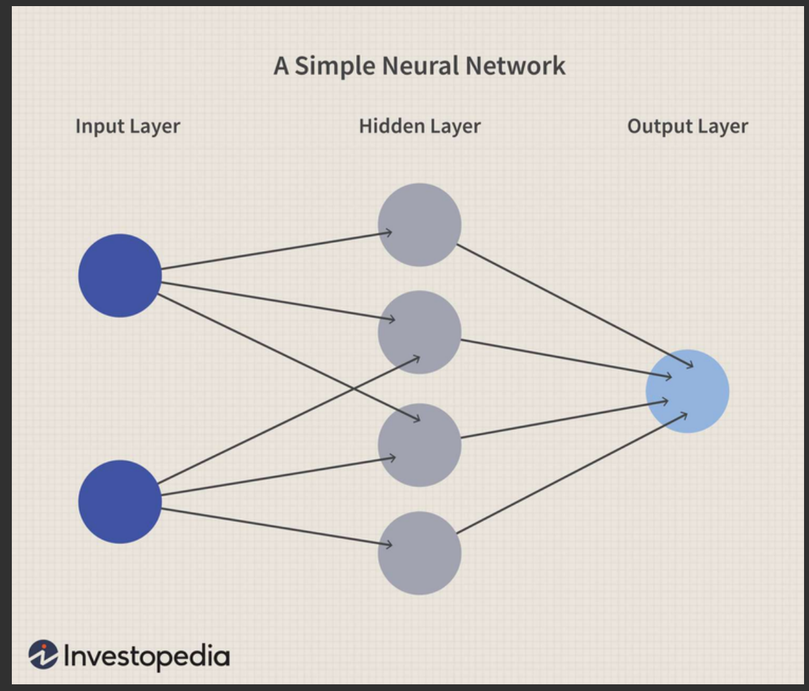
Fig 4. Aqui podriamos tener nuestras sospechas respecto al dataset...

3 REDES NEURONALES

Ahora que hemos explorado diferentes modelos de clasificación, toca sacar el arma pesada: las redes neuronales. Aunque su nivel de abstracción es muy grande, sus resultados son innegables.

La misma red neuronal, por motivos de practicidad, se montó utilizando la librería TensorFlow y su entorno Keras.

Al ser un problema de clasificación, se utilizó una función softmax y se aseguró que la última capa contuviera 4 neuronas (por nuestras 4 clases).



```python
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Definir intervalos y etiquetas
intervalos = [1, 350, 700, 1050, float('inf')]  # Define tus intervalos según tus criterios
etiquetas = ['Excelentes', 'Buen nivel', 'Mah o meno', 'Regular']  # Define las etiquetas para cada intervalo

# Crear la variable objetivo basada en los intervalos
dfclean['clasificacion'] = pd.cut(dfclean['Rank'], bins=intervalos, labels=etiquetas)

# Codificar la variable objetivo ('clasificacion')
le = LabelEncoder()
dfclean['label'] = le.fit_transform(dfclean['clasificacion'])

# Restar 1 a las etiquetas para que estén en el rango correcto [0, 3]
dfclean['label'] = (dfclean['label'] - 1).clip(lower=0)

# El resto del código permanece igual...


Xred = dfclean.drop("institution", axis=1)
Xred = Xred.drop("Rank", axis=1)
Xred = Xred.drop("location code", axis=1)
Xred = Xred.drop("score scaled", axis=1)
Xred = Xred.drop("location", axis=1)
X2= Xred.drop("clasificacion", axis=1)
X2 = X2.drop("label", axis=1)
Y2 = dfclean["label"]


# Paso 1: Preprocesamiento de datos
features = X2
target = Y2

# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
```

```python
# Normalizar los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Construir el modelo de la red neuronal
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax')  # '4' es el número de clases
])

# Compilar el modelo
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Entrenar el modelo
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluar el modelo en el conjunto de prueba
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'\nTest Accuracy: {test_accuracy}')

# Obtener predicciones
predictions = model.predict(X_test)
predicted_labels = tf.argmax(predictions, axis=1)

# Imprimir el informe de clasificación
print('\nClassification Report:')
print(classification_report(y_test, predicted_labels))
```

Fig 5. Codigo de la red neuronal en tensorflow con python

```
Epoch 1/10
36/36 [==============================] - 1s 9ms/step - loss: 1.1688 - accuracy: 0.6209 - val_loss: 0.9465 - va
Epoch 2/10
36/36 [==============================] - 0s 4ms/step - loss: 0.8067 - accuracy: 0.7203 - val_loss: 0.6570 - va
Epoch 3/10
36/36 [==============================] - 0s 3ms/step - loss: 0.6012 - accuracy: 0.7704 - val_loss: 0.5079 - va
Epoch 4/10
36/36 [==============================] - 0s 3ms/step - loss: 0.4957 - accuracy: 0.8188 - val_loss: 0.4291 - va
Epoch 5/10
36/36 [==============================] - 0s 3ms/step - loss: 0.4406 - accuracy: 0.8347 - val_loss: 0.3828 - va
Epoch 6/10
36/36 [==============================] - 0s 3ms/step - loss: 0.4122 - accuracy: 0.8487 - val_loss: 0.3642 - va
Epoch 7/10
36/36 [==============================] - 0s 3ms/step - loss: 0.3942 - accuracy: 0.8417 - val_loss: 0.3579 - va
Epoch 8/10
36/36 [==============================] - 0s 3ms/step - loss: 0.3839 - accuracy: 0.8487 - val_loss: 0.3406 - va
Epoch 9/10
36/36 [==============================] - 0s 3ms/step - loss: 0.3743 - accuracy: 0.8531 - val_loss: 0.3284 - va
Epoch 10/10
36/36 [==============================] - 0s 3ms/step - loss: 0.3662 - accuracy: 0.8575 - val_loss: 0.3295 - va
9/9 [==============================] - 0s 2ms/step - loss: 0.3295 - accuracy: 0.8526

Test Accuracy: 0.8526315689086914
9/9 [==============================] - 0s 2ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.95      0.93       139
           1       0.72      0.64      0.68        69
```

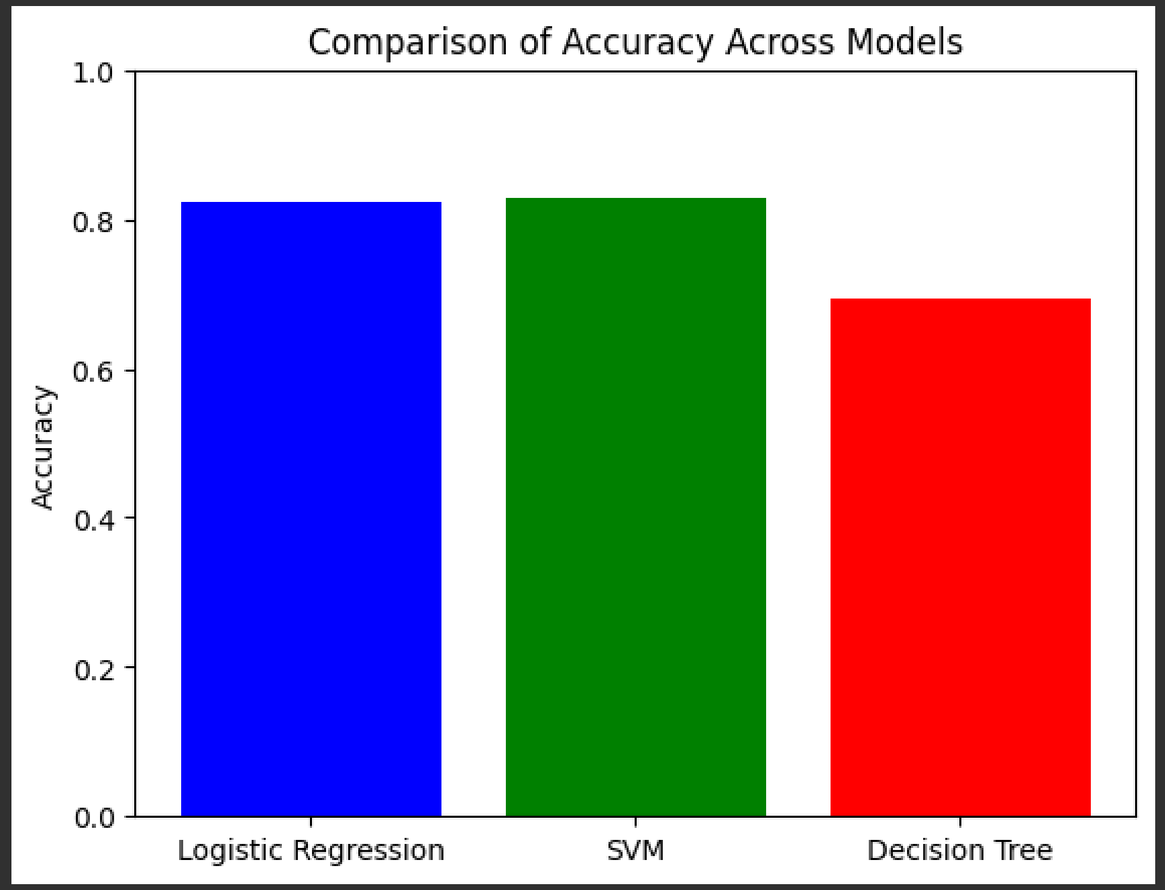
Fig 6. Proceso de entrenamiento

CONCLUSIÓN

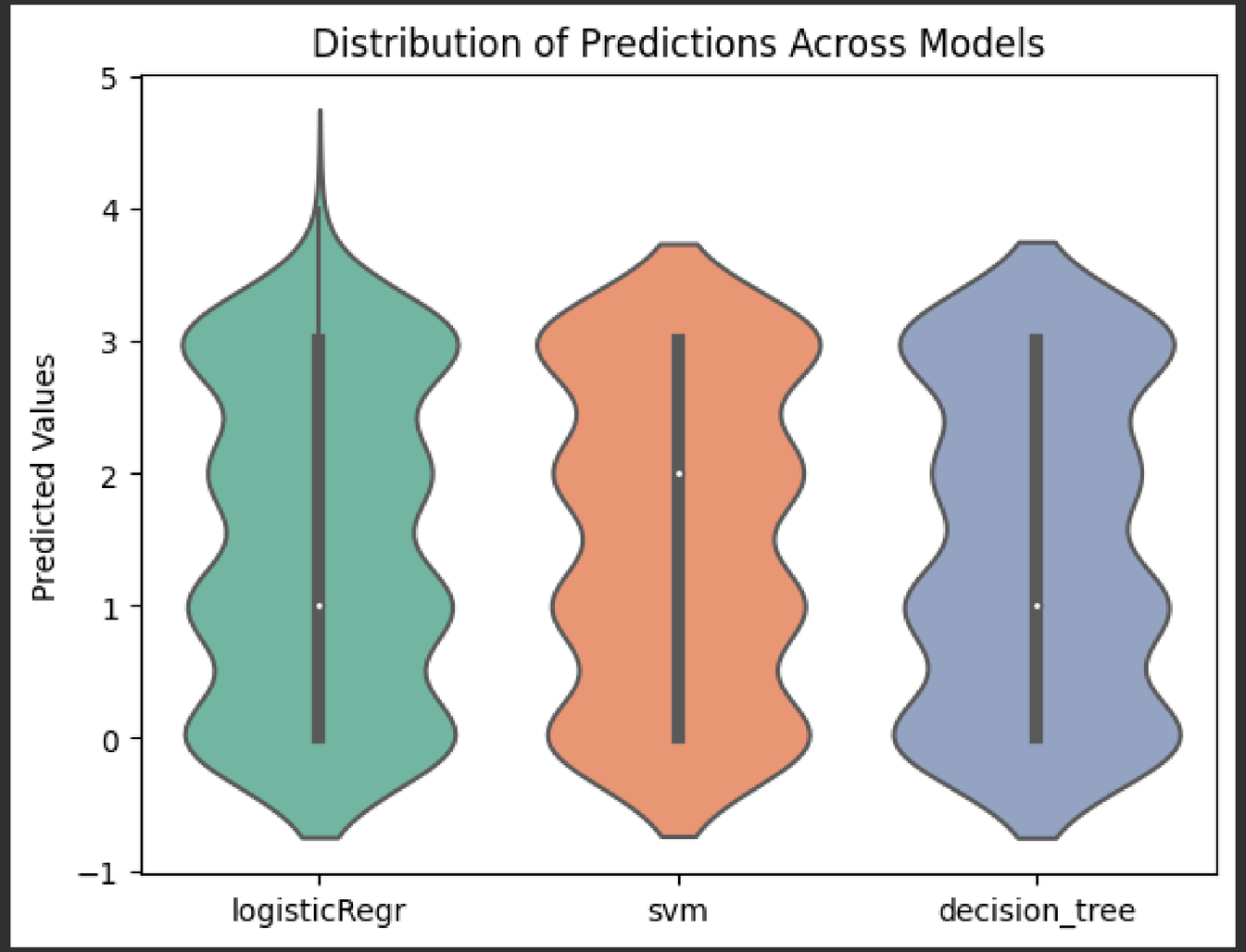
De acuerdo con el analisis estadistico y los resultados de los diferentes modelos de seleccion de variables, una de las primeras conclusion que podemos llevar acabo es : Los datos capturados en este data set no fueron seleccionados con precaucion, debido a que duplican infomacion o bien, cuentan con variables que tienen muy poca aportacion.

Ademas de esto, respecto a los modelos, pudimos hacer notar las carencias de los diferentes modelos en situaciones limte, como lo fue el hecho de la regresion logistica solametne clasifica dos clases de manera eficaz, o el problema de la dimensionalidad y el tiempo de computo que conlleva metodos como SVM o bien metodos recursivos con kernels complicados como arboles de decision.

APENDICE:


Comparison of Accuracy Across Models
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Logistic Regression
SVM
Decision Tree

APENDICE:


Distribution of Predictions Across Models
Predicted Values
5
4
3
2
1
0
−1
logisticRegr
svm
decision_tree