PROYECTO FINAL

OBJETIVO

Este proyecto tiene tiene como enfoque principal tratar de clasificar (mediante distintos metodos de inteligencia artificial) personas expuestas a sufrir de obesidad teniendo en cuenta factores como lo son la alimentación, el ejercicio que realiza, la cantidad de agua que toma por día, etc. En base a estos hábitos, se puede predecir en una etapa temprana cuando una persona puede llegar a sufrir de esta enfermedad y así evitar que pueda llegar a tener complicaciones en el futuro.

CONTENIDO

- Dataset
- Gaussian Naive Bayes
- Decision Tree
- Random forest
- Support Vector Machine (SVC)
- Deep Learning (DL)
- PCA

DATASET

- Datos para la estimación de los niveles de obesidad en personas de los países de México, Perú y Colombia, con edades entre los 14 y 61 años.
- 2111 filas y 17 columnas. 9 columnas con valores categoricos, y 8 con valores numéricos.

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no	Public_Transportation	Normal_Weight
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes	Public_Transportation	Normal_Weight
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently	Public_Transportation	Normal_Weight
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently	Walking	Overweight_Level_I
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	Public_Transportation	Overweight_Level_II
5	Male	29.0	1.62	53.0	no	yes	2.0	3.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	Automobile	Normal_Weight
6	Female	23.0	1.50	55.0	yes	yes	3.0	3.0	Sometimes	no	2.0	no	1.0	0.0	Sometimes	Motorbike	Normal_Weight
7	Male	22.0	1.64	53.0	no	no	2.0	3.0	Sometimes	no	2.0	no	3.0	0.0	Sometimes	Public_Transportation	Normal_Weight
8	Male	24.0	1.78	64.0	yes	yes	3.0	3.0	Sometimes	no	2.0	no	1.0	1.0	Frequently	Public_Transportation	Normal_Weight
9	Male	22.0	1.72	68.0	yes	yes	2.0	3.0	Sometimes	no	2.0	no	1.0	1.0	no	Public_Transportation	Normal_Weight

Tomado de: https://www.kaggle.com/datasets/ankurbajaj9/obesity-levels/data

DATASET

Nuestro ground truth será la columna 17, 'NObeyesdad' cuyos valores son categóricos y sus clases son:

- Insufficient_Weight.
- Normal_Weight.
- Overweight_Level_I.
- Overweight_Level_II.
- Obesity_Type_I.
- Obesity_Type_II.
- Obesity_Type_III.

GAUSSIAN NAIVE BAYES

Accuracy score -> 55%

Gaussian Naive Bayes

```
[ ]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
est = GaussianNB()
est.fit(X_train, y_train)
print(accuracy_score(est.predict(X_test), y_test))
```

```
0.5508274231678487
```

DECISION TREE

Sin tunning de parametros

Decision Tree

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

# Crear el modelo de árbol de decisión
model = DecisionTreeClassifier()

# Ajustar el modelo con los datos de entrenamiento
model.fit(X_train, y_train)

# Predecir las categorías en el conjunto de prueba
y_pred = model.predict(X_test)

# Evaluar la precisión del modelo
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f'Precisión del modelo: {accuracy}')
```

```
Precisión del modelo: 0.7777777777777778
```

Accuracy score -> 77.7%

Tunning de parametros

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05, random_state=21)
model = DecisionTreeClassifier(max_depth = max_depth_value, criterion=max_criterion)
model.fit(X_train,y_train)
print('Accuracy: ',accuracy_score(model.predict(X_test), y_test))
```

```
Accuracy:  0.8301886792452831
```

- criterion = entropy.
- max_depth = 14.
- Train/test = 0.95/0.05.

Accuracy score -> 83.01%

RANDOM FOREST

Sin tunning de parametros

Random forest

```
est = RandomForestClassifier()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21)
est.fit(X_train,y_train)
print(accuracy_score(est.predict(X_test), y_test))
```

```
0.8794326241134752
```

Accuracy score -> 87.94%

Tunning de parametros

- n_estimators = 141.
- criterion = log_loss.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=21)
est = RandomForestClassifier(n_estimators = max_n_estimators, max_depth = max_depth_value, criterion=max_criterion)
est.fit(X_train,y_train)
print('Accuracy: ',accuracy_score(est.predict(X_test), y_test))
```

```
Accuracy:  0.9053627760252366
```

- max_depth = 18.
- Train/test = 0.85/0.15

Accuracy score -> 90.53%

SUPPORT VECTOR MACHINE (SVC)

Sin tunning de parámetros

Support Vector Machine (SVC)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21)
est = SVC()
est.fit(X_train, y_train)
print(accuracy_score(est.predict(X_test), y_test))
```

```
0.4657210401891253
```

Accuracy score -> 46.57%

Tunning de parametros

- C = 36.
- Train/test = 0.65/0.35

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=21)
est = SVC(C=max_C, kernel=max_kernel)
est.fit(X_train,y_train)
print(accuracy_score(est.predict(X_test), y_test))
```

```
0.6698240866035182
```

- kernel = linear.

Accuracy score -> 66.98%

DEEP LEARNING (DL)

One-hot-encoding para la variable categórica 'NObeyesdad' (Ground Truth)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

```
y_train_ohe = tf.keras.utils.to_categorical(y_train, num_classes=7)
y_test_ohe = tf.keras.utils.to_categorical(y_test, num_classes=7)
```

Modelo secuencial para implementar una red neuronal con las siguientes caracteristicas:

- 1 capa con 1024 neuronas, función de activación (tanh).
- 1 capa con 1024 neuronas, función de activación (tanh).
- 1 capa con 1024 neuronas, función de activación (tanh).
- 1 capa con 64 neuronas, función de activación (tanh).
- 1 capa de salida con 7 neuronas (numero de clases), función de activación (softmax).

```
[ ]
model = tf.keras.Sequential([
    tf.keras.layers.Flatten( input_shape=X_train[0].shape),
    tf.keras.layers.Dense(1024, activation=tf.nn.tanh),
    tf.keras.layers.Dense(1024, activation=tf.nn.tanh),
    tf.keras.layers.Dense(1024, activation=tf.nn.tanh),
    tf.keras.layers.Dense(64, activation=tf.nn.tanh),
    tf.keras.layers.Dense(7, activation=tf.nn.softmax)
])
```

DEEP LEARNING (DL)

Se compila el modelo con los siguientes parametros:

- Optimizador de gradiente descendiente (SGD).
- loss = 'categorical_crossentropy'.
- metrica = 'accuracy'

```
model.compile(optimizer=tf.keras.optimizers.SGD(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train, y_train_ohe, epochs=15, batch_size=10)

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

Se entrena el modelo con los siguientes parametros:

- epochs = 15.
- batch_size = 10

```
print('Test accuracy:', test_acc)
```

```
Test accuracy: 0.5744680762290955
```

Accuracy = 57.4%

PCA

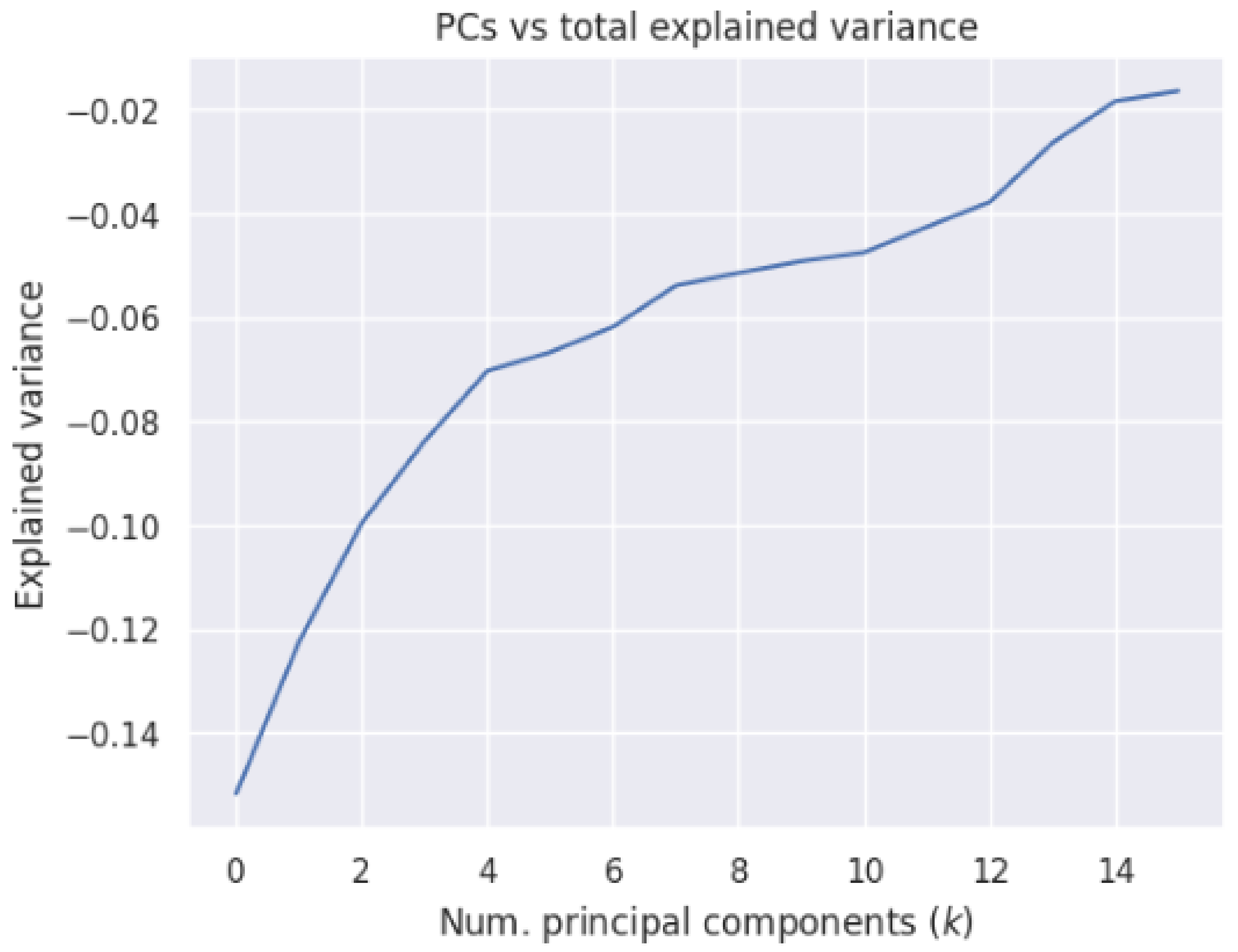
```
total_explained_variance = sum(explained_variance[:k])

print(explained_variance)

print("total_explained_variance (k=14):", total_explained_variance)
```

```
[0.15178865 0.1225576  0.09962691 0.08385851 0.07029266 0.06670767
 0.06183443 0.0538767  0.05143353 0.04918663 0.04747567 0.04254509
 0.03776924 0.02632637 0.018326   0.01639434]
total_explained_variance (k=14): 0.9652796507303126
```

```
plt.plot(-1*explained_variance)
plt.title("PCs vs total explained variance")
plt.xlabel("Num. principal components ($k$)")
plt.ylabel("Explained variance")
plt.show()
```



Se escoge k = 14 como el numero de componentes ideal ya que si bien k = 15 nos genera una varianza explicada mayor, el crecimiento de este valor respecto a la varianza explicada de k = 14 es muy poco.

PCA

```
[ ]  from sklearn.decomposition import PCA

     mypca = PCA(n_components=14)
     X_pca = mypca.fit_transform(standardized_data)
```

```
plt.scatter(X_pca[:,0], X_pca[:,1], cmap="rainbow")
plt.legend()
plt.show()
```

```
<ipython-input-20-9d802de55cd3>:1: UserWarning: No data for colormapping provi
  plt.scatter(X_pca[:,0], X_pca[:,1], cmap="rainbow")
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
```