GUÍA MÉTODOS TENSORFLOW / PANDAS / SCIPY / SEABORN PARA DEEP LEARNING

En esta guía se explica el mapeo de métodos e instrucciones que cambian de Tensorflow 2.0 a Tensorflow 2.2 o posterior. En realidad se pueden ejecutar los métodos de la versión 2.0 si tienes la versión 2.2, pero te pueden aparece warnings durante la ejecución indicando que el método va a quedar desactualizado (aunque siempre puedes instalar versiones previas de la librería).

Para que dispongas de la comparativa de estos cambios por cada proyecto de los realizados durante la formación, aquí aparecen todas las modificaciones de cada instrucción. Para que te sea más sencillo, en los scripts solución dispones de la versión del script en Tensorflow 2.0 pero también en Tensorflow 2.2 o posterior para que ejecutes la apropiada en función de la versión que tengas (si acabas de hacer la instalación, tendrás la versión 2.2 o posterior):

Caso de uso	Método Tensorflow 2.0 / Seaborn	Método Tensorflow 2.2 / Nuevas versiones Seaborn – Pandas - Scipy
Regresión	sns.distplot(df['buy_price'])	sns.histplot(df['buy_price']
Regresión	model.save('modelo_pred_vivienda.h5')	model.save('modelo_pred_vivienda.keras')
Clasificación binaria	Admite número de neuronas que no sea número entero: model.add(Dense(units=np.round(num_neuronas/2),activation='relu'))	No admite que el número de neuronas no sea número entero: model.add(Dense(units=np.round(num_neuronas/2).**astype(int)**,activation='relu'))
Clasificación binaria	predictions = model.predict_classes(X_test)	predictions = (model.predict(X_test) > 0.5).astype("int32")
Clasificación binaria	model.save('modelo_pred_vivienda.h5')	model.save('modelo_pred_heart.keras')
Clasificación binaria / multiclase	df.corr() df.corr()['Length_Employed'].sort_values()	df.corr(numeric_only=True) df.corr(numeric_only = True)['Length_Employed'].sort_values()
Clasificación multiclase	total_acc_inc_avg = df.groupby('Total_Accounts').mean()['Annual_Income']	total_acc_inc_avg = df.groupby('Total_Accounts' ['Annual_Income']).mean()
Clasificación multiclase	df.groupby('Total_Accounts').mean()['Length_Employed']	df.groupby('Total_Accounts')['Length_Employed'].mean()
Clasificación multiclase	total_acc_avg = df.groupby('Total_Accounts').mean()['Length_Employed']	total_acc_avg = df.groupby('Total_Accounts')['Length_Employed'].mean()
Clasificación multiclase	En versiones anteriores se ignoraban columnas no numéricas al hacer mean(): df.groupby("Home_Owner").mean()	Ahora no se ignoran y por tanto hay que indicarlo explícitamente: df.groupby("Home_Owner").mean(numeric_only=True)
Clasificación multiclase	total_acc_inc_avg = df.groupby('Total_Accounts').mean()['Annual_Income']	total_acc_inc_avg = df.groupby('Total_Accounts')['Annual_Income'].mean()
Clasificación multiclase	sns.heatmap(df.corr(),annot=True,cmap='viridis')	sns.heatmap(df.corr(numeric_only = True),annot=True,cmap='viridis')
Clasificación multiclase	df.corr()['Interest_Rate'].sort_values()	df.corr(numeric_only = True)['Interest_Rate'].sort_values()

Clasificación multiclase	df.corr()['Interest_Rate'].sort_values()	df.corr(numeric_only = True)['Interest_Rate'].sort_values()
Clasificación multiclase	Admite número de neuronas que no sea número entero: model.add(Dense(units=np.round(num_neuronas/2),activation='relu'))	No admite que el número de neuronas no sea número entero: model.add(Dense(units=np.round(num_neuronas/2).**astype(int)**,activation='relu'))
Clasificación multiclase	predictions_lab = model.predict_classes(X_test)	predictions = np.argmax(model.predict(X_test), axis=-1) #Si nueva versión de scikit-learn
Clasificación multiclase	df_pred = pd.read_excel('./info_prestamos - predecir.xlsx',sheets="info_prestamos - predecir")	df_pred = pd.read_excel('./info_prestamos - predecir.xlsx',sheet_name="info_prestamos - predecir")
CNN B&N	model.add() se ha sustituido para directamente insertar los distintos tipos de capa en el modelo, ejemplo: model = Sequential() model.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(28, 28, 1), activation='relu')) model.add(MaxPool2D(pool_size=(2, 2))) model.add(Flatten()) model.add(Dense(128, activation='relu')) model.add(Dense(10, activation='softmax'))	model = Sequential([Input(shape=(28, 28, 1)), Conv2D(filters=32, kernel_size=(4, 4), activation='relu'), MaxPool2D(pool_size=(2, 2)), Flatten(), Dense(128, activation='relu'), Dense(10, activation='softmax')])
CNN B&N	predictions = model.predict_classes(x_test)	predictions = np.argmax(model.predict(x_test), axis=-1) #Si nueva versión de scikit-learn
CNN RGB	En Jointplot() ahora hay que especificar los argumentos explícitamente, hay que cambiar esta sentencia: sns.jointplot(dim1_ddl,dim2_ddl)	Por esta: sns.jointplot(x=dim1_ddl,y=dim2_ddl)
CNN RGB	model.add() se ha sustituido para directamente insertar los distintos tipos de capa en el modelo, ejemplo: model = Sequential() model.add(xxxxxx)	model = Sequential([Input(shape=image_shape), Conv2D(filters=32, kernel_size=(3,3), activation='relu'), MaxPooling2D(pool_size=(2, 2)), Conv2D(filters=64, kernel_size=(3,3), activation='relu'), MaxPooling2D(pool_size=(2, 2)), Conv2D(filters=64, kernel_size=(3,3), activation='relu'), MaxPooling2D(pool_size=(2, 2)), Flatten(), Dense(128, activation='relu'), Dropout(0.5), Dense(5, activation='softmax')])
CNN RGB	results = model.fit_generator(train_image_gen, epochs=5, validation_data=test_image_gen, callbacks=[early_stop])	Ya no aplica el método fit_generator en la versión 2.2, directamente fit(): results = model.fit(train_image_gen, epochs=5, validation_data=test_image_gen, callbacks=[early_stop])

CNN RGB	model.save('modelo_CNN_flores.h5')	model.save('modelo_CNN_flores.keras')
CNN RGB	`model.evaluate_generator(test_image_gen)`	model.evaluate(test_image_gen)
CNN RGB	`pred_probabilities = model.predict_generator(test_image_gen)`	pred_probabilities = model.predict(test_image_gen)
CNN RGB	`predictions = model.predict_classes(test_image_gen)`	predictions = np.argmax(model.predict(test_image_gen), axis=-1)
RNN Forecasting	model.add() se ha sustituido para directamente insertar los distintos tipos de capa en el modelo: model = Sequential() model.add(LSTM(150, activation='relu', input_shape=(longitud, n_variables))) # model.add(Dense(n_variables))	Cambiar por: model = Sequential([Input(shape=(longitud, n_variables)), LSTM(150, activation='relu'), Dense(n_variables)])
RNN Forecasting	model.fit_generator(generador,epochs=20, validation_data=val_generador, callbacks=[early_stop])	model.fit(generador,epochs=20, validation_data=val_generador, callbacks=[early_stop])
RNN Forecasting	Predicción nuevos datos: model.fit_generator(generador,epochs=15)	model.fit(generador,epochs=15)
NN No supervisado	En seaborn: sns.distplot(df["Age"]) sns.histplot(df["Annual Income (k$)"],norm_hist=True)	sns.histplot(df["Age"]) sns.histplot(df["Annual Income (k$)"], stat="density", kde=True)
NN No supervisado	sns.heatmap(df.corr(),annot=True,cmap='viridis')	sns.heatmap(df.corr(numeric_only=True),annot=True,cmap='viridis')
NN No supervisado	encoder = Sequential() encoder.add(Dense(units=500,activation='relu',input_shape=[4])) encoder.add(Dense(units=500,activation='relu',input_shape=[500])) encoder.add(Dense(units=2000,activation='relu',input_shape=[500])) encoder.add(Dense(units=10,activation='relu',input_shape=[2000])) decoder = Sequential() decoder.add(Dense(units=2000,activation='relu',input_shape=[10])) decoder.add(Dense(units=500,activation='relu',input_shape=[2000])) decoder.add(Dense(units=500,activation='relu',input_shape=[500])) decoder.add(Dense(units=4,activation='relu',input_shape=[500]))	encoder = Sequential([Input(shape=(4,)), # Definición de entrada Dense(500, activation='relu'), Dense(500, activation='relu'), Dense(2000, activation='relu'), Dense(10, activation='relu')]) decoder = Sequential([Input(shape=(10,)), # salida del encoder Dense(2000, activation='relu'), Dense(500, activation='relu'), Dense(500, activation='relu'), Dense(4, activation='relu') # reconstrucción (mismas dimensiones de entrada)])
NN No supervisado	autoencoder.compile(loss="mse", optimizer=SGD(lr=1.5))	autoencoder.compile(loss="mse", optimizer=SGD(learning_rate=1.5))