task3

May 15, 2024

1 Task 3

2 Correlation

Correlation is a statistical measure that describes the extent to which two or more variables fluctuate together. A positive correlation indicates that as one variable increases, the other variable tends to increase, while a negative correlation indicates that as one variable increases, the other tends to decrease. Correlation is quantified by the correlation coefficient, a value between -1 and 1. A correlation coefficient of 1 implies a perfect positive correlation, -1 implies a perfect negative correlation, and 0 implies no correlation. Understanding correlation is crucial in data analysis as it helps in identifying relationships between variables, which can inform decision-making, feature selection in machine learning, and identifying patterns in datasets. However, it is important to note that correlation does not imply causation; it simply indicates a relationship between variables without providing evidence that changes in one variable cause changes in another.

3 Practical Example in Cybersecurity

In cybersecurity, correlation can be used to analyze the relationship between different features of network traffic data to identify patterns indicative of normal or malicious behavior. For instance, we can analyze how features such as packet size, duration, and number of packets relate to each other and to the occurrence of network attacks.

[24]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# For demonstration, we create a random dataset
np.random.seed(42)
data_size = 1000
feature_size = 20
data = pd.DataFrame(np.random.rand(data_size, feature_size),␣
 ↪columns=[f'feature_{i}' for i in range(feature_size)])
data['label'] = np.random.randint(0, 2, size=data_size)
```

```
# Output 10 rows from the generated data
print("Sample data (10 rows):")
print(data.head(10))

# Reshape data for CNN input (e.g., treat features as 1D spatial data)
X = data.drop('label', axis=1).values.reshape(-1, feature_size, 1)
y = data['label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train.reshape(-1, feature_size)).reshape(-1,␣
 ↪feature_size, 1)
X_test = scaler.transform(X_test.reshape(-1, feature_size)).reshape(-1,␣
 ↪feature_size, 1)

# Define the CNN model
model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(64, kernel_size=3, activation='relu',␣
 ↪input_shape=(feature_size, 1)),
    tf.keras.layers.MaxPooling1D(pool_size=2),
    tf.keras.layers.Conv1D(128, kernel_size=3, activation='relu'),
    tf.keras.layers.MaxPooling1D(pool_size=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),  # Adding dropout for regularization
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32,␣
 ↪validation_split=0.2)

# Evaluate the model
y_pred = (model.predict(X_test) > 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred)
print(f'CNN Accuracy: {accuracy * 100:.2f}%')
```

```
Sample data (10 rows):
   feature_0  feature_1  feature_2  feature_3  feature_4  feature_5  \
0   0.374540   0.950714   0.731994   0.598658   0.156019   0.155995
1   0.611853   0.139494   0.292145   0.366362   0.456070   0.785176
2   0.122038   0.495177   0.034389   0.909320   0.258780   0.662522
3   0.388677   0.271349   0.828738   0.356753   0.280935   0.542696
4   0.863103   0.623298   0.330898   0.063558   0.310982   0.325183
5   0.031429   0.636410   0.314356   0.508571   0.907566   0.249292
6   0.807440   0.896091   0.318003   0.110052   0.227935   0.427108
7   0.962447   0.251782   0.497249   0.300878   0.284840   0.036887
8   0.367783   0.632306   0.633530   0.535775   0.090290   0.835302
9   0.341066   0.113474   0.924694   0.877339   0.257942   0.659984

   feature_6  feature_7  feature_8  feature_9  …  feature_11  feature_12  \
0   0.058084   0.866176   0.601115   0.708073  …    0.969910    0.832443
1   0.199674   0.514234   0.592415   0.046450  …    0.170524    0.065052
2   0.311711   0.520068   0.546710   0.184854  …    0.775133    0.939499
3   0.140924   0.802197   0.074551   0.986887  …    0.198716    0.005522
4   0.729606   0.637557   0.887213   0.472215  …    0.713245    0.760785
5   0.410383   0.755551   0.228798   0.076980  …    0.161221    0.929698
6   0.818015   0.860731   0.006952   0.510747  …    0.222108    0.119865
7   0.609564   0.502679   0.051479   0.278646  …    0.239562    0.144895
8   0.320780   0.186519   0.040775   0.590893  …    0.016588    0.512093
9   0.817222   0.555201   0.529651   0.241852  …    0.897216    0.900418

   feature_13  feature_14  feature_15  feature_16  feature_17  feature_18  \
0    0.212339    0.181825    0.183405    0.304242    0.524756    0.431945
1    0.948886    0.965632    0.808397    0.304614    0.097672    0.684233
2    0.894827    0.597900    0.921874    0.088493    0.195983    0.045227
3    0.815461    0.706857    0.729007    0.771270    0.074045    0.358466
4    0.561277    0.770967    0.493796    0.522733    0.427541    0.025419
5    0.808120    0.633404    0.871461    0.803672    0.186570    0.892559
6    0.337615    0.942910    0.323203    0.518791    0.703019    0.363630
7    0.489453    0.985650    0.242055    0.672136    0.761620    0.237638
8    0.226496    0.645173    0.174366    0.690938    0.386735    0.936730
9    0.633101    0.339030    0.349210    0.725956    0.897110    0.887086

   feature_19  label
0    0.291229      1
1    0.440152      1
2    0.325330      1
3    0.115869      0
4    0.107891      0
5    0.539342      0
6    0.971782      1
7    0.728216      0
8    0.137521      1
9    0.779876      0

[10 rows x 21 columns]
```

```
Epoch 1/50
```

```
/opt/anaconda3/lib/python3.11/site-
packages/keras/src/layers/convolutional/base_conv.py:99: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(
```

```
18/18            0s 4ms/step -
accuracy: 0.5295 - loss: 0.7055 - val_accuracy: 0.5071 - val_loss: 0.6916
Epoch 2/50
18/18            0s 1ms/step -
accuracy: 0.5460 - loss: 0.6892 - val_accuracy: 0.4929 - val_loss: 0.6900
Epoch 3/50
18/18            0s 1ms/step -
accuracy: 0.5416 - loss: 0.6876 - val_accuracy: 0.5071 - val_loss: 0.6914
Epoch 4/50
18/18            0s 1ms/step -
accuracy: 0.5933 - loss: 0.6690 - val_accuracy: 0.5714 - val_loss: 0.6859
Epoch 5/50
18/18            0s 1ms/step -
accuracy: 0.5673 - loss: 0.6688 - val_accuracy: 0.5071 - val_loss: 0.7038
Epoch 6/50
18/18            0s 1ms/step -
accuracy: 0.5787 - loss: 0.6727 - val_accuracy: 0.5714 - val_loss: 0.6829
Epoch 7/50
18/18            0s 1ms/step -
accuracy: 0.6264 - loss: 0.6515 - val_accuracy: 0.5000 - val_loss: 0.6994
Epoch 8/50
18/18            0s 2ms/step -
accuracy: 0.6128 - loss: 0.6596 - val_accuracy: 0.5786 - val_loss: 0.6905
Epoch 9/50
18/18            0s 1ms/step -
accuracy: 0.6146 - loss: 0.6421 - val_accuracy: 0.5357 - val_loss: 0.6865
Epoch 10/50
18/18            0s 2ms/step -
accuracy: 0.7303 - loss: 0.6018 - val_accuracy: 0.5429 - val_loss: 0.7010
Epoch 11/50
18/18            0s 2ms/step -
accuracy: 0.7102 - loss: 0.6086 - val_accuracy: 0.5286 - val_loss: 0.7168
Epoch 12/50
18/18            0s 2ms/step -
accuracy: 0.6933 - loss: 0.5962 - val_accuracy: 0.5071 - val_loss: 0.7633
Epoch 13/50
18/18            0s 2ms/step -
accuracy: 0.6554 - loss: 0.6260 - val_accuracy: 0.5357 - val_loss: 0.7286
Epoch 14/50
18/18            0s 2ms/step -
```

```
accuracy: 0.7506 - loss: 0.5546 - val_accuracy: 0.5143 - val_loss: 0.7177
Epoch 15/50
18/18             0s 1ms/step -
accuracy: 0.7470 - loss: 0.5491 - val_accuracy: 0.5143 - val_loss: 0.7344
Epoch 16/50
18/18             0s 2ms/step -
accuracy: 0.7545 - loss: 0.5223 - val_accuracy: 0.5071 - val_loss: 0.7288
Epoch 17/50
18/18             0s 1ms/step -
accuracy: 0.7584 - loss: 0.5196 - val_accuracy: 0.5429 - val_loss: 0.7246
Epoch 18/50
18/18             0s 2ms/step -
accuracy: 0.7780 - loss: 0.4787 - val_accuracy: 0.5500 - val_loss: 0.7461
Epoch 19/50
18/18             0s 2ms/step -
accuracy: 0.7942 - loss: 0.4632 - val_accuracy: 0.5286 - val_loss: 0.7459
Epoch 20/50
18/18             0s 2ms/step -
accuracy: 0.8385 - loss: 0.4147 - val_accuracy: 0.5929 - val_loss: 0.7634
Epoch 21/50
18/18             0s 2ms/step -
accuracy: 0.8500 - loss: 0.4099 - val_accuracy: 0.5071 - val_loss: 0.7962
Epoch 22/50
18/18             0s 2ms/step -
accuracy: 0.8690 - loss: 0.3479 - val_accuracy: 0.5429 - val_loss: 0.7892
Epoch 23/50
18/18             0s 2ms/step -
accuracy: 0.8892 - loss: 0.3301 - val_accuracy: 0.5643 - val_loss: 0.7828
Epoch 24/50
18/18             0s 2ms/step -
accuracy: 0.8690 - loss: 0.3320 - val_accuracy: 0.5429 - val_loss: 0.8291
Epoch 25/50
18/18             0s 2ms/step -
accuracy: 0.9216 - loss: 0.2854 - val_accuracy: 0.5500 - val_loss: 0.8453
Epoch 26/50
18/18             0s 2ms/step -
accuracy: 0.9216 - loss: 0.2639 - val_accuracy: 0.5571 - val_loss: 0.9511
Epoch 27/50
18/18             0s 2ms/step -
accuracy: 0.9165 - loss: 0.2399 - val_accuracy: 0.5714 - val_loss: 0.9014
Epoch 28/50
18/18             0s 1ms/step -
accuracy: 0.9359 - loss: 0.2123 - val_accuracy: 0.5286 - val_loss: 0.9667
Epoch 29/50
18/18             0s 1ms/step -
accuracy: 0.9238 - loss: 0.2021 - val_accuracy: 0.5357 - val_loss: 0.9857
Epoch 30/50
18/18             0s 1ms/step -
```

```
accuracy: 0.9413 - loss: 0.1862 - val_accuracy: 0.5429 - val_loss: 1.0113
Epoch 31/50
18/18            0s 2ms/step -
accuracy: 0.9602 - loss: 0.1604 - val_accuracy: 0.5643 - val_loss: 0.9915
Epoch 32/50
18/18            0s 2ms/step -
accuracy: 0.9755 - loss: 0.1293 - val_accuracy: 0.5643 - val_loss: 1.0308
Epoch 33/50
18/18            0s 2ms/step -
accuracy: 0.9616 - loss: 0.1371 - val_accuracy: 0.5214 - val_loss: 1.2062
Epoch 34/50
18/18            0s 2ms/step -
accuracy: 0.9489 - loss: 0.1498 - val_accuracy: 0.5714 - val_loss: 1.0420
Epoch 35/50
18/18            0s 2ms/step -
accuracy: 0.9558 - loss: 0.1388 - val_accuracy: 0.5643 - val_loss: 1.1678
Epoch 36/50
18/18            0s 2ms/step -
accuracy: 0.9733 - loss: 0.1030 - val_accuracy: 0.5643 - val_loss: 1.1317
Epoch 37/50
18/18            0s 2ms/step -
accuracy: 0.9796 - loss: 0.0937 - val_accuracy: 0.5429 - val_loss: 1.2757
Epoch 38/50
18/18            0s 2ms/step -
accuracy: 0.9797 - loss: 0.0915 - val_accuracy: 0.5286 - val_loss: 1.1883
Epoch 39/50
18/18            0s 2ms/step -
accuracy: 0.9865 - loss: 0.0657 - val_accuracy: 0.5786 - val_loss: 1.1994
Epoch 40/50
18/18            0s 1ms/step -
accuracy: 0.9931 - loss: 0.0588 - val_accuracy: 0.5643 - val_loss: 1.3314
Epoch 41/50
18/18            0s 2ms/step -
accuracy: 0.9937 - loss: 0.0503 - val_accuracy: 0.5571 - val_loss: 1.2587
Epoch 42/50
18/18            0s 2ms/step -
accuracy: 0.9897 - loss: 0.0556 - val_accuracy: 0.5357 - val_loss: 1.4767
Epoch 43/50
18/18            0s 2ms/step -
accuracy: 0.9979 - loss: 0.0508 - val_accuracy: 0.5643 - val_loss: 1.3490
Epoch 44/50
18/18            0s 2ms/step -
accuracy: 0.9990 - loss: 0.0320 - val_accuracy: 0.5786 - val_loss: 1.3933
Epoch 45/50
18/18            0s 2ms/step -
accuracy: 0.9975 - loss: 0.0348 - val_accuracy: 0.5929 - val_loss: 1.4001
Epoch 46/50
18/18            0s 2ms/step -
```

```
accuracy: 0.9984 - loss: 0.0365 - val_accuracy: 0.5500 - val_loss: 1.3672
Epoch 47/50
18/18            0s 2ms/step -
accuracy: 0.9988 - loss: 0.0335 - val_accuracy: 0.5714 - val_loss: 1.5075
Epoch 48/50
18/18            0s 2ms/step -
accuracy: 0.9976 - loss: 0.0271 - val_accuracy: 0.5643 - val_loss: 1.3962
Epoch 49/50
18/18            0s 2ms/step -
accuracy: 0.9972 - loss: 0.0363 - val_accuracy: 0.5500 - val_loss: 1.4632
Epoch 50/50
18/18            0s 2ms/step -
accuracy: 0.9961 - loss: 0.0322 - val_accuracy: 0.5714 - val_loss: 1.4989
10/10            0s 2ms/step
CNN Accuracy: 46.33%
```

Please use the "Kernel>Restart & Run All" command in Jupyter Notebook and check your results before submitting your homework. Note that I rerun all boxes on my side before grading.