|  |
| --- |
| **Pertemuan 11 dan 12 – Stacked-Bidirectional on Neural Network** |
|  |
| **Tujuan pembelajaran** |
| * Mahasiswa mampu memahami konsep timeseries pada cryptocurrency * Mahasiswa mampu memahami konsep stacked-bidirectional pada neural network |

Studi kasus: Model Prediksi Harga BTC-USD Menggunakan

Metode SBi-LSTM-RNN dan SBi-GRU-RNN

|  |
| --- |
| **C01\_data\_collection.py** |
|  |
| # lib data manipulation  import pandas as pd    # function load dataset  def data\_collection(df):      # load dataset    dataset = pd.read\_csv("dataset/"+df, parse\_dates=['Date'])    dataset = dataset[["Date", "Open", "High", "Low", "Close"]]      # return values    return dataset |

|  |
| --- |
| **C02\_visualization.py** |
|  |
| # lib data visualizations  import matplotlib.pyplot as plt  from matplotlib.dates import DateFormatter    # function of lineplot  def timeseries\_matplotlib(df, nm\_labels):        # create lineplot      fig, ax = plt.subplots(figsize = (8,4))      for x in range(len(nm\_labels)):        ax.plot(df.iloc[:, 0:1], df.iloc[:, x+1:x+2], label=nm\_labels[x], linewidth=2.5)        # set label-labels      ax.set\_title("", fontsize=12)      ax.set\_xlabel("", fontsize=10)      ax.set\_ylabel("", fontsize=10)      ax.legend(loc="best")      ax.grid(True)        # show lineplot      return plt.show()    # visualisasi timeseries plot  def lineplot\_matplotlib1(x, y, label, title):      # membuat time series plot    fig, ax = plt.subplots(figsize = (8,4))    ax.plot(x, y, color="tab:blue", label=label, linewidth=2.5)      # membuat label-label    ax.xaxis.set\_major\_formatter(DateFormatter("%Y"))    ax.set\_title(title, fontsize=12)    ax.set\_xlabel("", fontsize=12)    ax.set\_ylabel("", fontsize=12)    ax.legend(loc="best")    ax.grid(True)      # return values    return plt.show()    # visualisasi timeseries plot  def lineplot\_matplotlib2(x1, y1, label1, x2, y2, label2, title):      # membuat time series plot    fig, ax = plt.subplots(figsize = (8,4))    ax.plot(x1, y1, color="tab:blue", label=label1, linewidth=2.5, linestyle="solid")    ax.plot(x2, y2, color="tab:red", label=label2, linewidth=2.5, linestyle="solid")      # membuat label-label    ax.xaxis.set\_major\_formatter(DateFormatter("%Y"))    ax.set\_title(title, fontsize=12)    ax.set\_xlabel("", fontsize=12)    ax.set\_ylabel("", fontsize=12)    ax.legend(loc="best")    ax.grid(True)      # return values    return plt.show()    # visualisasi timeseries plot  def lineplot\_matplotlib3(x1, y1, label1, x2, y2, label2, title):      # membuat time series plot    fig, ax = plt.subplots(figsize = (8,4))    ax.plot(x1, y1, color="tab:blue", label=label1, linewidth=2.5, linestyle="solid")    ax.plot(x2, y2, color="tab:orange", label=label2, linewidth=2.5, linestyle="solid")      # membuat label-label    ax.set\_title(title, fontsize=12)    ax.set\_xlabel("", fontsize=12)    ax.set\_ylabel("", fontsize=12)    ax.legend(loc="best")    ax.grid(True)      # return values    return plt.show() |

|  |
| --- |
| **C03\_preprocessing.py** |
|  |
| # lib manipulation data  import numpy as np  import pandas as pd    # lib data preprocessing  from sklearn.preprocessing import MinMaxScaler  from sklearn.model\_selection import train\_test\_split  # ----------------------------------------------------------------------------------------    # function for supervised learning  def create\_dataset(look\_back, dataset):        # declare variable X and Y      dataX = []      dataY = []        # for loop for create supervised learning      for i in range(look\_back, len(dataset)):          dataX.append(dataset[i-look\_back:i, 0])          dataY.append(dataset[i, 0])        # return value X and Y      return np.array(dataX), np.array(dataY)    # functions of data preprocessing  def preprocessing(dataset):      # 1. set feature    data = dataset.filter(['Close'])    data = data.values      # 2. normalize features    scaler = MinMaxScaler(feature\_range=(0, 1))    scaled = scaler.fit\_transform(np.array(data).reshape(-1,1))      # 3. traing testing    train\_data, test\_data = train\_test\_split(  scaled, train\_size=0.8, test\_size=0.2, shuffle=False)      # 4. supervised learning    x\_train, y\_train = create\_dataset(60, train\_data)    x\_test, y\_test = create\_dataset(60, test\_data)      # 5. reshape input to be [samples, time steps, features]    x\_train = np.reshape(x\_train, (x\_train.shape[0], x\_train.shape[1], 1))    x\_test = np.reshape(x\_test, (x\_test.shape[0], x\_test.shape[1], 1))      # return values    return scaler, scaled, x\_train, y\_train, x\_test, y\_test    # function for inverse normalized  def inverse(scaler, scaled):    return scaler.inverse\_transform(scaled.reshape(-1,1)) |

|  |
| --- |
| **C04\_model\_predictions.py** |
|  |
| # lib neural network algorithms  import tensorflow as tf  from keras.layers import LSTM  from keras.layers import GRU    # func model predictions  def get\_models(algorithm, x\_train, y\_train, x\_test, y\_test):      # 1. The LSTM architecture    if algorithm == "SBi-LSTM-RNN":      tf.keras.backend.clear\_session()      model = tf.keras.Sequential([        tf.keras.layers.Bidirectional(  LSTM(units=50, return\_sequences=True, input\_shape=(x\_train.shape[1], 1))),        tf.keras.layers.Bidirectional(LSTM(units=50, return\_sequences=False)),        tf.keras.layers.Dropout(0.05),        tf.keras.layers.Dense(1)      ])      # 2. The GRU-RNN architecture    if algorithm == "SBi-GRU-RNN":      tf.keras.backend.clear\_session()      model = tf.keras.Sequential([        tf.keras.layers.Bidirectional(  GRU(units=50, return\_sequences=True, input\_shape=(x\_train.shape[1], 1))),        tf.keras.layers.Bidirectional(GRU(units=50, return\_sequences=False)),        tf.keras.layers.Dropout(0.05),        tf.keras.layers.Dense(1)      ])      # 2. compile models    model.compile(optimizer='adamax', loss="mean\_squared\_error")      # 3. fitting models    history = model.fit(      x=x\_train, y=y\_train,      batch\_size=16, epochs=50, verbose="auto",      validation\_data=(x\_test, y\_test),      shuffle=False, use\_multiprocessing=False,    )      # 4. predict models    predictions = model.predict(x\_test, verbose=0)      # return values    return history, predictions |

|  |
| --- |
| **C05\_model\_evaluate.py** |
|  |
| # libs manipulations array  import numpy as np    # lib evaluate models  import scipy.stats as sc  from sklearn.metrics import mean\_absolute\_error  from sklearn.metrics import root\_mean\_squared\_error  from sklearn.metrics import mean\_absolute\_percentage\_error    # func evaluate models  def evaluate\_models(ytrue,ypred):      # calculate mae, rmse, mape    r     = sc.mstats.pearsonr(ytrue,ypred)[0]    p     = sc.mstats.pearsonr(ytrue,ypred)[1]    mae   = mean\_absolute\_error(ytrue,ypred)    rmse  = root\_mean\_squared\_error(ytrue,ypred)    mape  = mean\_absolute\_percentage\_error(ytrue,ypred)      # return values    return np.round(r,4), np.round(p,4), np.round(mae,4), np.round(rmse,4), np.round(mape,4) |

|  |
| --- |
| **K01-Volatilitas.ipynb** |
|  |
| # libs manipulations array  import pandas as pd  import numpy as np    # Muat data BTC-USD  btc\_data = pd.read\_csv('dataset/BTC-USD.csv', parse\_dates=['Date'])    # Hitung pengembalian harian  btc\_data['daily\_return'] = btc\_data['Close'].pct\_change()    # Hapus baris dengan nilai NaN  #btc\_data.dropna(inplace=True)    # Tambahkan kolom tahun  btc\_data['Year'] = btc\_data['Date'].dt.year    # Hitung volatilitas tahunan  annual\_volatility = btc\_data.groupby('Year')['daily\_return'].std() \* np.sqrt(252)  annual\_volatility = annual\_volatility \* 100    # Tampilkan hasil  for year, volatility in annual\_volatility.items():      print(f"Volatilitas untuk tahun {year}: {volatility:.2f}") |
|  |
| Volatilitas untuk tahun 2015: 57.21  Volatilitas untuk tahun 2016: 39.90  Volatilitas untuk tahun 2017: 79.24  Volatilitas untuk tahun 2018: 67.36  Volatilitas untuk tahun 2019: 56.55  Volatilitas untuk tahun 2020: 59.86  Volatilitas untuk tahun 2021: 66.82  Volatilitas untuk tahun 2022: 52.80  Volatilitas untuk tahun 2023: 36.37  Volatilitas untuk tahun 2024: 49.34 |

|  |
| --- |
|  |
|  |
|  |
|  |