**Mudah Belajar Otodidak Data Science**

**(Praktek Menggunakan Python3)**

**Edisi 2 Tahun 2023**

**Disusun Oleh:**

**Aryajaya Alasmsyah, S.Kom., M.Kom., MTA.**

**Materi Praktikum Data Science**

1. Pengantar Bahasa Python
2. Visualisasi Data Menggunakan Matplotlib dan Seaborn
3. Visualisasi Data Menggunakan Framework Streamlit
4. Teknik-Teknik Praproses Data – Data Tabular, Time Series, dan Spasial
5. Algoritma Klasifikasi Data Bagian 1
6. Algoritma Klastering Data Bagian 2
7. Ujian Tengah Semester (UTS)
8. Algoritma Klastering Data Bagian 1
9. Algoritma Klastering Data Bagian 2
10. Algoritma Regresi Linier
11. Algoritma Neural Network: SBi-LSTM dan SBi-GRU
12. Algoritma Neural Network: SBi-LSTM-XGBoost dan SBi-GRU-XGBoostost
13. Presentasi Projek
14. Ujian Akhir Semester (UAS)

|  |
| --- |
| **Pertemuan 11 – Stacked-Bidirectional on Neural Network** |

|  |
| --- |
| **Tujuan pembelajaran**   * Mahasiswa mampu memahami konsep timeseries. * Mahasiswa mampu memahami konsep Stacked-Bidirectional pada Neural Network. * Mahasiswa mampu menerapkan atau membuat model prediksi timeseries menggunakan metode Stacked-Bidirectional pada Neural Network. |

**Studi kasus: Prediksi bitcoin menggunakan metode SBi-LSTM-RNN dan SBi-GRU-RNN.**

**C01\_visualization.py**

|  |
| --- |
| # Declaration library  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    plt.show()  # ----------------------------------------------------------------------------------------    # func timeseries plot  def lineplot\_matplotlib1(x1, y1, label1, title):      # create lineplot    fig, ax = plt.subplots(figsize = (8,4))    ax.plot(x1, y1, color="tab:blue", label=label1, linewidth=2.5)      # set label-labels    ax.set\_title(title, fontsize=12)    ax.set\_xlabel("", fontsize=10)    ax.set\_ylabel("", fontsize=10)    ax.legend(loc="best")    ax.grid(True)      # show lineplot    plt.show()  # ----------------------------------------------------------------------------------------    # func timeseries plot  def lineplot\_matplotlib2(x1, y1, label1, x2, y2, label2, title):      # create lineplot    fig, ax = plt.subplots(figsize = (8,4))    ax.plot(x1, y1, color="tab:blue", label=label1, linewidth=2.5)    ax.plot(x2, y2, color="tab:red", label=label2, linewidth=2.5)      # set label-labels    ax.set\_title(title, fontsize=12)    ax.set\_xlabel("", fontsize=10)    ax.set\_ylabel("", fontsize=10)    ax.legend(loc="best")    ax.grid(True)      # show lineplot    plt.show()  # ----------------------------------------------------------------------------------------    # func timeseries plot  def lineplot\_matplotlib3(x1, y1, label1, x2, y2, label2, title):      # create lineplot    fig, ax = plt.subplots(figsize = (8,4))    ax.plot(x1, y1, color="tab:blue", label=label1, linewidth=2.5)    ax.plot(x2, y2, color="tab:red", label=label2, linewidth=2.5)      # set label-labels    ax.xaxis.set\_major\_formatter(DateFormatter("%Y"))    ax.set\_title(title, fontsize=12)    ax.set\_xlabel("", fontsize=10)    ax.set\_ylabel("", fontsize=10)    ax.legend(loc="best")    ax.grid(True)      # show lineplot    plt.show()  # ---------------------------------------------------------------------------------------- |

**C02\_model\_predictions.py**

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

**C03\_model\_evaluate.py**

|  |
| --- |
| # libs manipulations array  import numpy as np    # lib evaluate models  from math import sqrt  import scipy.stats as sc  from sklearn.metrics import mean\_absolute\_error  from sklearn.metrics import mean\_squared\_error  from sklearn.metrics import mean\_absolute\_percentage\_error    # func evaluate models  def evaluate\_models(y\_test, predictions):      # calculate mae, rmse, mape    r = sc.mstats.pearsonr(y\_test, predictions)[0]    p = sc.mstats.pearsonr(y\_test, predictions)[1]    mae = mean\_absolute\_error(y\_test, predictions)    rmse = sqrt(mean\_squared\_error(y\_test, predictions))    mape = mean\_absolute\_percentage\_error(y\_test, predictions)      # return values    return np.round(r,4), np.round(p,4), np.round(mae,4), np.round(rmse,4), np.round(mape,4) |

**Stacked-Bidirectional-NN.ipynb**

|  |
| --- |
| # load all functions  from C01\_visualization import \*  from C02\_model\_predictions import \*  from C03\_model\_evaluate import \*    # lib manipulation data  import numpy as np  import pandas as pd    # lib data visualizations  import seaborn as sns  import matplotlib.pyplot as plt    # lib data preprocessing  from sklearn.preprocessing import MinMaxScaler  from sklearn.model\_selection import train\_test\_split |

|  |
| --- |
| # set random number  import random as rm  rm.seed(1234)    # set random number  import numpy as np  np.random.seed(1234)    # set random number  import tensorflow as tf  tf.random.set\_seed(1234) |

1. **Akuisisi Data**

|  |
| --- |
| # load dataset  dataset = pd.read\_csv("../dataset/Cryptocurrency-BTC-USD-2024-04.csv", parse\_dates=['Date'])  dataset = dataset[["Date", "Open", "High", "Low", "Close"]]  dataset.info() |
|  |
| <class 'pandas.core.frame.DataFrame'>  RangeIndex: 3408 entries, 0 to 3407  Data columns (total 5 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----  0 Date 3408 non-null datetime64[ns]  1 Open 3408 non-null float64  2 High 3408 non-null float64  3 Low 3408 non-null float64  4 Close 3408 non-null float64  dtypes: datetime64[ns](1), float64(4)  memory usage: 133.3 KB |
|  |
| # show dataset  print(dataset.tail()) |
|  |
| Date Open High Low Close  3403 2024-04-26 64485.371094 64789.656250 63322.398438 63755.320313  3404 2024-04-27 63750.988281 63898.363281 62424.718750 63419.140625  3405 2024-04-28 63423.515625 64321.484375 62793.597656 63113.230469  3406 2024-04-29 63106.363281 64174.878906 61795.457031 63841.121094  3407 2024-04-30 63839.417969 64703.332031 59120.066406 60636.855469 |

1. **Eksplorasi Data Analysis**

|  |
| --- |
| # visualisasi data  timeseries\_matplotlib(dataset, ["Open", "High", "Low", "Close"]) |
|  |
|  |
| Gambar x. Output program |

1. **Praproses Data**

|  |
| --- |
| # 1. process feature selection  data = dataset.filter(['Close'])  data = data.values |
|  |
| # 1. results feature selection  np.round(data[:5],6) |
|  |
| array([[314.248993],  [315.032013],  [281.082001],  [264.195007],  [274.473999]]) |
|  |
| # 2. process normalize features  scaler = MinMaxScaler(feature\_range=(0, 1))  scaled = scaler.fit\_transform(np.array(data).reshape(-1,1)) |
|  |
| # 2. results normalize features  np.round(scaled[:5],6) |
|  |
| array([[0.001867],  [0.001878],  [0.001413],  [0.001181],  [0.001322]]) |
|  |
| # 2. results normalize features  lineplot\_matplotlib1(    x1=dataset["Date"], y1=scaled, label1="Close Price",    title="Results of Normalize Data"  ) |
|  |
|  |
| Gambar x. Output program |
|  |
| # 3. process inverse normalize features  def inverse(scaler, scaled):    hasil = scaler.inverse\_transform(scaled.reshape(-1,1))    return hasil |
|  |
| # 4. process splitting data  train\_data, test\_data = train\_test\_split(scaled, train\_size=0.80, test\_size=0.20, shuffle=False) |
|  |
| # 4. results splitting data  lineplot\_matplotlib2(    x1=dataset.iloc[0:len(train\_data),0], y1=train\_data, label1="Training data",    x2=dataset.iloc[len(train\_data):len(dataset),0], y2=test\_data, label2="Testing data",    title="Results of Splitting Data"  ) |
|  |
|  |
| Gambar x. Output program |

1. **Supervised Learning**

|  |
| --- |
| # 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)  # ---------------------------------------------------------------------------------------- |
|  |
| # process supervised learning  x\_train, y\_train = create\_dataset(60, train\_data)  x\_test, y\_test = create\_dataset(60, test\_data) |
|  |
| # reshape input to be [samples, time steps, features]  x\_train = np.reshape(x\_train, (x\_train.shape[0], x\_train.shape[1], 1))  print(x\_train.shape) |
|  |
| (2666, 60, 1) |
|  |
| # reshape input to be [samples, time steps, features]  x\_test = np.reshape(x\_test, (x\_test.shape[0], x\_test.shape[1], 1))  print(x\_test.shape) |
|  |
| (622, 60, 1) |

1. **Model SBi-LSTM-RNN**

|  |
| --- |
| # config algorithms  algorithms="SBi-LSTM-RNN" |
|  |
| # process predict with LSTM  lstm\_model = get\_models(algorithms=algorithms, timestep=x\_train.shape[1]) |
|  |
| # process predict with LSTM  lstm\_history, lstm\_predictions = get\_predictions(    model=lstm\_model, x\_train=x\_train, y\_train=y\_train, x\_test=x\_test, y\_test=y\_test,  ) |
|  |
| # results training and validation  lineplot\_matplotlib1(    x1=lstm\_history.epoch, y1=lstm\_history.history["loss"], label1="loss func on training",    title="Results training and validation on "+algorithms  ) |
|  |
|  |
|  |