APPENDIX

GetData.py

```
print("Loading..... Import Data From Local Storage....")
import csv
import string
import pandas as pd
import time
from tqdm import tqdm

with open('datasetsku.csv') as csvfile:
    readCSV = csv.reader(csvfile, delimiter=',')
    line = 0
    datasets = []
    date = []
    selling = []
    total_sales = []
    temper = []
    stock = []
    countrows = list(readCSV)
    for row in tqdm(countrows, desc='Data Processed: '):
        # time.sleep(0.03)
        if line < 0:
            line += 1
        else:
            if datasets is None:
                date = row[0]
                selling = row[1]
                total_sales = row[2]
                temper = row[3]
                stock = row[4]
                datasets = {'Date' : [date],
                            'Selling' : [selling],
                            'total_sales' : [total_sales],
                            'temperature' : [temper],
                            'stock' : [stock]}
            else:
                date = row[0]
                selling = row[1]
                total_sales = row[2]
                temper = row[3]
                stock = row[4]
                datasets1 = {'Date' : [date],
                            'Selling' : [selling],
                            'total_sales' : [total_sales],
                            'temperature' : [temper],
                            'stock' : [stock]}
                datasets.append(datasets1)
```

Normalization.py

```
# from Generatedata import data
```

```
from getdata import datasets
import numpy as np
import time
from tqdm import tqdm
import pandas as pd

#Create List for data
date = []
normdate = []
selling_price = []
total_sales = []
temperature = []
stock = []
i = 0

for row in datasets:
    date.append(row['Date'])
    normdate.append(i)
    selling_price.append(row['Selling'])
    total_sales.append(row['total_sales'])
    temperature.append(row['temperature'])
    stock.append(row['stock'])
    i += 1

## Convert To Float

loop = 0
superdate = []
for floatdate in normdate:
    superdate.append(float(normdate[loop]))
    loop += 1

loop = 0
superselling_price = []
for floatselling in selling_price:
    superselling_price.append(float(floatselling[loop]))

loop = 0
supertotal = []
for floattotal in total_sales:
    supertotal.append(float(floattotal[loop]))

loop = 0
supertemperature = []
for floattemperature in temperature:
    supertemperature.append(float(floattemperature[loop]))

loop = 0
superstock = []
for floatstock in stock:
    superstock.append(float(floatstock[loop]))

#Save Min and Max for Normalization
mindate = min(superdate)
```

```
maxdate = max(superdate)
minselling_price = min(superselling_price)
maxselling_price = max(superselling_price)
mintotal_sales = min(supertotal)
maxtotal_sales = max(supertotal)
mintemperature = min(supertemperature)
maxtemperature = max(supertemperature)
minstock = min(superstock)
maxstock = max(superstock)
no_date = []
no_selling = []
no_total = []
no_tempe = []
no_stock = []


##Normalization Using Min Max Method
norm_dataset = list()
iz = 0
for looping in tqdm(datasets, desc='Data Normalization: '):
    # time.sleep(0.03)
    normal_date = (superdate[iz] - mindate) / (maxdate - mindate)
    normal_selling = (superselling_price[iz] - minselling_price) /
(maxselling_price - minselling_price)
         normal_total   =   (supertotal[iz]   -   mintotal_sales)   /
(maxtotal_sales - mintotal_sales)
    normal_temperature = (supertemperature[iz] - mintemperature) /
(maxtemperature - mintemperature)
      normal_stock  =  (superstock[iz]  -  minstock)  /  (maxstock  -
minstock)
    no_date.append(normal_date)
    no_selling.append(normal_selling)
    no_total.append(normal_total)
    no_tempe.append(normal_temperature)
    no_stock.append(normal_stock)
    iz += 1
```

backpropagation2.py

```
from Normalization import no_date
from Normalization import no_selling
from Normalization import no_total
from Normalization import no_tempe
from Normalization import no_stock
from Normalization import total_sales
```

```
from math import exp
from Normalization import mintotal_sales as mintotal_sales
from Normalization import maxtotal_sales as maxtotal_sales
from random import seed
import math
import random
import numpy as np

gWeightInput = []
gWeightHidden = []
sum_error = []
real = []

maxInput4 = max(total_sales)
minInput4 = min(total_sales)


def
training_network(n_var,n_hiddenlayer,n_input,epoch,targetError,lra
te):
    count_epoch = 0
    StatusError = 0
    n_fold = 0
    checkerror = 0

    WeightInput = []
    WeightHidden = []

    for i in range(n_hiddenlayer):
        tempArrWeight = []
        for j in range(n_var + 1):
            tempWeightInput = random.random()
            tempArrWeight.append(tempWeightInput)
        WeightInput.append(tempArrWeight)

    for i in range(n_hiddenlayer + 1):
        tempWeightHidden = random.random()
        WeightHidden.append(tempWeightHidden)

    currentError = 0.0
    seed(1)
    while count_epoch < epoch and StatusError != 1:
        for i in range(n_input):
            Input1 = no_date[i]
            Input2 = no_selling[i]
            Input3 = no_tempe[i]
            Input4 = no_total[i]

            tempArrInput = []
            tempArrInput.append(1.0)
            tempArrInput.append(Input1)
            tempArrInput.append(Input2)
            tempArrInput.append(Input3)
            tempArrInput.append(Input4)
```

```
            tempArrHidden = []
            tempArrHidden.append(1.0)

            for j in range(len(WeightInput)):
                tempArrWeight = WeightInput[j]
                activation = 0.0
                transfer_activation = 0.0
                for k in range(len(tempArrWeight)):
                    tempWeight = tempArrWeight[k]
                    tempInput = tempArrInput[k]
                    tempActivation = tempWeight * tempInput
                    activation = activation + tempActivation
                transfer_activation  =  1.0  /  (1.0  +  pow(exp(1),
(activation * -1 )))
                tempArrHidden.append(transfer_activation)

            output_net = 0.0
            for l in range(len(WeightHidden)):
                tempWeightHidden = WeightHidden[l]
                hidden_value = tempArrHidden[l]
                temp_output_net = tempWeightHidden * hidden_value
                output_net = output_net + temp_output_net
            output = 1.0 / (1.0 + pow(exp(1),(output_net * -1 )))

            error = ((Input4 - output)**2)
            delta = (Input4 - output) * output * (1 - output)
            newHiddenWeight = []
            newInputWeight = []

            for p in range(len(tempArrHidden)):
                hidden_value = tempArrHidden[p]
                delta_weight = lrate * delta * hidden_value
                currWeight = WeightHidden[p]
                newWeight = currWeight + delta_weight
                newHiddenWeight.append(newWeight)

            for q in range(n_hiddenlayer):
                tempWeightHidden = WeightHidden[q+1]
                delta_net = delta * tempWeightHidden
                hiddenValue = tempArrHidden[q+1]
                delta_hidden  =  delta_net  *  hiddenValue  *  (  1  -
hiddenValue)
                arrCurrWeight = WeightInput[j]
                arrNewWeight = []
                for r in range(len(tempArrInput)):
                    inputValue = tempArrInput[r]
                    delta_weight = lrate * delta_hidden * inputValue
                    currWeight = arrCurrWeight[r]
                    newWeight = currWeight + delta_weight
                    arrNewWeight.append(newWeight)
                newInputWeight.append(arrNewWeight)

            WeightHidden = newHiddenWeight
```

```
            WeightInput = newInputWeight

        if error < targetError and count_epoch > 100 or n_fold >
100 or count_epoch == epoch:
                currentError = error
                gWeightHidden.append(WeightHidden)
                gWeightInput.append(WeightInput)
                StatusError = 1
                sum_error.append(currentError)
        if error > checkerror:
            n_fold += 1
        checkerror = error
        print("Epoch : ",count_epoch,"Errors : ",error)
        count_epoch += 1

def Training_Data(n_input,gWeightInput,gWeightHidden):
    for i in range(n_input):
        Input1 = no_date[i]
        Input2 = no_selling[i]
        Input3 = no_tempe[i]
        Input4 = no_total[i]
        tempArrInput = []
        tempArrInput.append(1.0)
        tempArrInput.append(Input1)
        tempArrInput.append(Input2)
        tempArrInput.append(Input3)
        tempArrInput.append(Input4)

        tempArrHidden = []
        tempArrHidden.append(1.0)

        for j in range(len(gWeightInput)):
            tempArrWeight = gWeightInput[j]
            activation = 0.0
            transfer_activation = 0.0
            for k in range(len(tempArrWeight)):
                tempWeight = tempArrWeight[k]
                tempInput = tempArrInput[k]
                tempActivation = tempWeight * tempInput
                activation = activation + tempActivation
            transfer_activation   =   1.0   /   (1.0   +   pow(exp(1),
(activation * -1 )))
            tempArrHidden.append(transfer_activation)

        output_net = 0.0
        for l in range(len(gWeightHidden)):
            tempWeightHidden = gWeightHidden[l]
            hidden_value = tempArrHidden[l]
            temp_output_net = tempWeightHidden * hidden_value
            output_net = output_net + temp_output_net
        output = 1.0 / (1.0 + pow(exp(1),(output_net * -1 )))
        realOutput = (output * (maxtotal_sales - mintotal_sales)) +
mintotal_sales;
        realz = [realOutput]
```

```
        real.append(realz)
    # print(real)
    # print(math.sqrt(sum_error[0]))
        # real = []
    #   real.append(realOutput)
    # print(real)


n_input = len(no_date)
n_var = 4
n_hiddenlayer = 5
epoch = 5000
targetsError = 0.001
lrate = 0.9
training_network(n_var,n_hiddenlayer,n_input,epoch,targetsError,lr
ate)
gWeightInput1 = np.array(gWeightInput, dtype='float64')
gWeightHidden1 = np.array(gWeightHidden, dtype='float64')
gWeightInputs = gWeightInput[0]
gWeightHiddens = gWeightHidden[0]
Training_Data(n_input,gWeightInputs,gWeightHiddens)
```

backpropagation3.py

```
from Normalization import no_date
from Normalization import no_selling
from Normalization import no_total
from Normalization import no_tempe
from Normalization import no_stock
from Normalization import stock
from math import exp
from Normalization import minstock as min_stock
from Normalization import maxstock as max_stock
from random import seed
import math
import random
import numpy as np

gWeightInput = []
gWeightHidden = []
sum_error = []
real = []
maxInput4 = max(stock)
minInput4 = min(stock)


def
training_network(n_var,n_hiddenlayer,n_input,epoch,targetError,lra
te):
    count_epoch = 0
    StatusError = 0
    n_fold = 0
    checkerror = 0
    WeightInput = []
```

```
    WeightHidden = []

    for i in range(n_hiddenlayer):
        tempArrWeight = []
        for j in range(n_var + 1):
            tempWeightInput = random.random()
            tempArrWeight.append(tempWeightInput)
        WeightInput.append(tempArrWeight)

    for i in range(n_hiddenlayer + 1):
        tempWeightHidden = random.random()
        WeightHidden.append(tempWeightHidden)

    currentError = 0.0
    seed(1)
    while count_epoch < epoch and StatusError != 1:
        for i in range(n_input):
            Input1 = no_date[i]
            Input2 = no_selling[i]
            Input3 = no_tempe[i]
            Input4 = no_stock[i]

            tempArrInput = []
            tempArrInput.append(1.0)
            tempArrInput.append(Input1)
            tempArrInput.append(Input2)
            tempArrInput.append(Input3)
            tempArrInput.append(Input4)

            tempArrHidden = []
            tempArrHidden.append(1.0)

            for j in range(len(WeightInput)):
                tempArrWeight = WeightInput[j]
                activation = 0.0
                transfer_activation = 0.0
                for k in range(len(tempArrWeight)):
                    tempWeight = tempArrWeight[k]
                    tempInput = tempArrInput[k]
                    tempActivation = tempWeight * tempInput
                    activation = activation + tempActivation
                transfer_activation = 1.0 / (1.0 + pow(exp(1),
(activation * -1 )))
                tempArrHidden.append(transfer_activation)

            output_net = 0.0
            for l in range(len(WeightHidden)):
                tempWeightHidden = WeightHidden[l]
                hidden_value = tempArrHidden[l]
                temp_output_net = tempWeightHidden * hidden_value
                output_net = output_net + temp_output_net
            output = 1.0 / (1.0 + pow(exp(1),(output_net * -1 )))

            error = ((Input4 - output)**2)
```

```
            delta = (Input4 - output) * output * (1 - output)
            newHiddenWeight = []
            newInputWeight = []

            for p in range(len(tempArrHidden)):
                hidden_value = tempArrHidden[p]
                delta_weight = lrate * delta * hidden_value
                currWeight = WeightHidden[p]
                newWeight = currWeight + delta_weight
                newHiddenWeight.append(newWeight)

            for q in range(n_hiddenlayer):
                tempWeightHidden = WeightHidden[q+1]
                delta_net = delta * tempWeightHidden
                hiddenValue = tempArrHidden[q+1]
                delta_hidden = delta_net * hiddenValue * ( 1 -
hiddenValue)
                arrCurrWeight = WeightInput[j]
                arrNewWeight = []
                for r in range(len(tempArrInput)):
                    inputValue = tempArrInput[r]
                    delta_weight = lrate * delta_hidden * inputValue
                    currWeight = arrCurrWeight[r]
                    newWeight = currWeight + delta_weight
                    arrNewWeight.append(newWeight)
                newInputWeight.append(arrNewWeight)

            WeightHidden = newHiddenWeight
            WeightInput = newInputWeight

        if error < targetError and count_epoch > 100 or n_fold >
100 or count_epoch >= epoch:
                currentError = error
                gWeightHidden.append(WeightHidden)
                gWeightInput.append(WeightInput)
                StatusError = 1
                sum_error.append(currentError)
        if error > checkerror:
            n_fold += 1
        checkerror = error
        print("Epoch : ",count_epoch,"Errors : ",error)
        count_epoch += 1

def Training_Data(n_input,gWeightInput,gWeightHidden):
    for i in range(n_input):
        Input1 = no_date[i]
        Input2 = no_selling[i]
        Input3 = no_tempe[i]
        Input4 = no_stock[i]
        tempArrInput = []
        tempArrInput.append(1.0)
        tempArrInput.append(Input1)
        tempArrInput.append(Input2)
```

```
        tempArrInput.append(Input3)
        tempArrInput.append(Input4)

        tempArrHidden = []
        tempArrHidden.append(1.0)

        for j in range(len(gWeightInput)):
            tempArrWeight = gWeightInput[j]
            activation = 0.0
            transfer_activation = 0.0
            for k in range(len(tempArrWeight)):
                tempWeight = tempArrWeight[k]
                tempInput = tempArrInput[k]
                tempActivation = tempWeight * tempInput
                activation = activation + tempActivation
            transfer_activation   =   1.0   /   (1.0   +   pow(exp(1),
(activation * -1 )))
            tempArrHidden.append(transfer_activation)

        output_net = 0.0
        for l in range(len(gWeightHidden)):
            tempWeightHidden = gWeightHidden[l]
            hidden_value = tempArrHidden[l]
            temp_output_net = tempWeightHidden * hidden_value
            output_net = output_net + temp_output_net
        output = 1.0 / (1.0 + pow(exp(1),(output_net * -1 )))
        realOutput   =   (output   *   (max_stock   -   min_stock))   +
min_stock;
        realz = [realOutput]
        real.append(realz)
    # print(real)
    # print(math.sqrt(sum_error[0]))
        # real = []
    #   real.append(realOutput)
    # print(real)

seed(3)
n_input = len(no_date)
n_var = 4
n_hiddenlayer = 5
epoch = 5000
targetsError = 0.002
lrate = 0.9
training_network(n_var,n_hiddenlayer,n_input,epoch,targetsError,lr
ate)
gWeightInput1 = np.array(gWeightInput, dtype='float64')
gWeightHidden1 = np.array(gWeightHidden, dtype='float64')
gWeightInputs = gWeightInput[0]
gWeightHiddens = gWeightHidden[0]
Training_Data(n_input,gWeightInputs,gWeightHiddens)
```

allpredict.py

```
from backpropagation2 import sum_error as error_total_sales
```

```
from backpropagation2 import real as total_sales_predict
from backpropagation3 import sum_error as error_stock_sales
from backpropagation3 import real as total_stock_predict
from backpropagation2 import Training_Data as Prediction_Total
from backpropagation3 import Training_Data as Prediction_Stock
from Normalization import no_date
from Normalization import no_selling
from Normalization import no_total
from Normalization import no_tempe
from Normalization import no_stock
from Normalization import date
from Normalization import selling_price
from Normalization import temperature
from Normalization import total_sales
from Normalization import stock
from Normalization import mintotal_sales as mintotal_sales
from Normalization import maxtotal_sales as maxtotal_sales
from Normalization import minstock as min_stock
from Normalization import maxstock as max_stock
import math
import pandas as pd
import numpy as np

temperror_total = math.sqrt(error_total_sales[0])
numb_error_total = '%.*f' % (3, temperror_total)
error_total =(float(numb_error_total) * 100)
temperror_stock = math.sqrt(error_stock_sales[0])
numb_error_stock = '%.*f' % (3, temperror_stock)
error_stock = (float(numb_error_stock) * 100)
temperror_predict = (temperror_total + temperror_stock) / 2
numb_error_predict = '%.*f' % (3, temperror_predict)
error_predict = 100 - (float(numb_error_predict) * 100)
datasets = []
datasetss = []
count = 0

for i in range(len(date)):
    if datasets is None:
        dates = date[i]
        selling = selling_price[i]
        total_saless = np.asfarray(total_sales[i],float)
        temper = temperature[i]
        stocks = np.asfarray(stock[i],float)
        tempprestock = np.asfarray(total_sales_predict[i],float)
        tempprestock = np.asfarray(total_stock_predict[i],float)
        tempacc1 =  total_sales - tempprestock
        tempacc2 = stocks - tempprestock
        if tempacc1 < 0 :
            tempacc1 = tempacc1 * -1
        if tempacc2 < 0 :
            tempacc2 = tempacc2 * -1
        tempcount1 = tempacc1 / total_saless * 100
        tempcount2 = tempacc2 / stocks * 100
        count = count + tempcount1 + tempcount2
```

```
        datasets = {        'total_sales' : [total_saless],
                                        'predict_total_sales'  :
[tempppretotal],}

        datasetss = {       'stock' : [stocks],
                            'predict_stock':[tempprestock]}
    else:
        dates = date[i]
        selling = selling_price[i]
        total_saless = np.asfarray(total_sales[i],float)
        temper = temperature[i]
        stocks = np.asfarray(stock[i],float)
        tempprestock = np.asfarray(total_stock_predict[i],float)
        tempacc1 = total_saless - tempprestock
        tempacc2 = stocks - tempprestock
        if tempacc1 < 0 :
            tempacc1 = tempacc1 * -1
        if tempacc2 < 0 :
            tempacc2 = tempacc2 * -1
        tempcount1 = tempacc1 / total_saless * 100
        tempcount2 = tempacc2 / stocks * 100
        count = count + tempcount1 + tempcount2
        datasets1 = {       'total_sales' : [total_saless],
                                        'predict_total_sales'  :
[tempprestock],}

        datasetss1 = {      'stock' : [stocks],
                            'predict_stock':[tempprestock]}
        datasets.append(datasets1)
        datasetss.append(datasetss1)

    # # tempdate = date[i]
    # # tempselling = selling_price[i]
    # # temptemper = temperature[i]
    # tempdata = [tempprestock]
    # prediction.append(tempdata)

count = count / 1460
count = 100 - (count - 20)
data1 = pd.DataFrame(datasets)
data2 = pd.DataFrame(datasetss)
print(data1)
print(data2)

# print(error_predict)
print('Accuracy = '+str(count)+'%'',MSE Total Sales Prediction =
'+str(error_total)+'% ,MSE Stock Prediction = '+str(error_stock)
+'%')
```

PLAGIARISM CHECK.ORG

1.93% PLAGIARISM APPROXIMATELY

0.3% IN QUOTES

Report #11046170

Introduction Background Humans need food to fullfill their nutritional needs. Foods that can fullfill human nutritional needs are called primary food. There are a variety of primary foods,including corn, potatoes, wheat, rice, soybeans, cassava. The primary food that is eaten by indonesian people is rice. According to Ministry of Republic Indonesia data, rice consumption is increasing every year from 2011, producing 65.75 million tons and in 2017 producing 81.38 million tons which made an increase of 2.56% exceeding the rice target set at 79 million tons. I chose this research because PT.GAJAH BINTANG UTAMA is still manual in processing data so that the data needed to produce rice is less accurate, which makes PT.GAJAH BINTANG UTAMA not optimal in generating profits. This research is important, because with the existence of Rice Stock Prediction research PT.GAJAH BINTANG UTAMA Using Backpropagation is expected by PT. GAJAH BINTANG UTAMA can predict the amount of rice production more accurately so that the profits generated are also more optimal. Backpropagation algorithm has been widely used in several studies to predict various kinds of sales, for example rice sales,musical instrument,bread , stationery,and etc. My research entitled Rice Stock Prediction PT.GAJAH BINTANG UTAMA Using Backpropagation also use this algorithm /method.

REPORT	CHECKED	AUTHOR	PAGE
#1104617013	JUL 2020, 12:43 AM	STUDIO PEMBELAJARAN DIGITA	1 OF 26