DAFTAR PUSTAKA

[1] V. Prakruthi, D. Sindhu and D. S. Anupama Kumar, "Real Time Sentiment Analysis Of Twitter Posts," *2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS)*, 2018, pp. 29-34, doi: 10.1109/CSITSS.2018.8768774.

[2] Kusrini and M. Mashuri, "Sentiment Analysis In Twitter Using Lexicon Based and Polarity Multiplication," *2019 International Conference of Artificial Intelligence and Information Technology (ICAIIT)*, 2019, pp. 365-368, doi: 10.1109/ICAIIT.2019.8834477.

[3] L. Mandloi and R. Patel, "Twitter Sentiments Analysis Using Machine Learning Methods," *2020 International Conference for Emerging Technology (INCET)*, 2020, pp. 1-5, doi: 10.1109/INCET49848.2020.9154183.

[4] F. A. Wenando, R. Hayami, Bakaruddin and A. Y. Novermahakim, "Tweet Sentiment Analysis for 2019 Indonesia Presidential Election Results using Various Classification Algorithms," *2020 1st International Conference on Information Technology, Advanced Mechanical and Electrical Engineering (ICITAMEE)*, 2020, pp. 279-282, doi: 10.1109/ICITAMEE50454.2020.9398513.

[5] D. A. Damaratih, "Sentiment Analysis Of Online Lecture Opinions On Twitter Social Media Using Naive Bayes Classifier," *2021 International Conference on Computer Science, Information Technology, and Electrical Engineering (ICOMITEE)*, 2021, pp. 24-28, doi: 10.1109/ICOMITEE53461.2021.9650135.

[6] M. T. H. K. Tusar and M. T. Islam, "A Comparative Study of Sentiment Analysis Using NLP and Different Machine Learning Techniques on US Airline Twitter Data," *2021 International Conference on Electronics, Communications and Information Technology (ICECIT)*, 2021, pp. 1-4, doi: 10.1109/ICECIT54077.2021.9641336.

[7] P. S. Mung and S. Phyu, "Effective Analytics on Healthcare *Big data* Using Ensemble Learning," *2020 IEEE Conference on Computer Applications(ICCA)*, 2020, pp. 1-4, doi: 10.1109/ICCA49400.2020.9022853.

[8] J. L. Fernández-Alemán, J. M. Carrillo-de-Gea, M. Hosni, A. Idri and G. García-Mateos, "Homogeneous and heterogeneous ensemble classification methods in diabetes disease: a review," *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2019, pp. 3956-3959, doi: 10.1109/EMBC.2019.8856341.

[9] Y. Feng, X. Wang and J. Zhang, "A Heterogeneous Ensemble Learning Method For Neuroblastoma Survival Prediction," in *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 4, pp. 1472-1483, April 2022, doi: 10.1109/JBHI.2021.3073056.

[10] X. LI, T. SHI, P. LI and W. ZHOU, "Application of Bagging Ensemble Classifier based on Genetic Algorithm in the Text Classification of Railway Fault Hazards," *2019 2nd International Conference on Artificial Intelligence and Big Data (ICAIBD)*, 2019, pp. 286-290, doi: 10.1109/ICAIBD.2019.8836988.

[11] S. Alelyani, "Stable bagging feature selection on medical data", *Journal of Big Data*, vol. 8, no. 11, pp. 1-18, Jan. 2021, doi: https://doi.org/10.1186/s40537-020-00385-8.

[12] L. Li *et al*, "Cluster-based bagging of constrained mixed-effects models for high spatiotemporal resolution nitrogen oxides prediction over large regions", *Environment International*, vol. 128, pp. 310-323, July 2019, doi: https://doi.org/10.1016/j.envint.2019.04.057.

[13] A. Tareq and N. Hewahi, "Sentiment Analysis of Tweets During COVID-19 Pandemic Using BLSTM," *2021 International Conference on Data Analytics for Business and Industry (ICDABI), 2021, pp. 245-249, doi: 10.1109/ICDABI53623.2021.9655932.*

[14] A. Poornima and K. S. Priya, "A Comparative Sentiment Analysis Of Sentence Embedding Using Machine Learning Techniques," *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2020, pp. 493-496, doi: 10.1109/ICACCS48705.2020.9074312.

[15] S. A. El Rahman, F. A. AlOtaibi and W. A. AlShehri, "Sentiment Analysis of Twitter Data," *2019 International Conference on Computer and Information Sciences (ICCIS)*, 2019, pp. 1-4, doi: 10.1109/ICCISci.2019.8716464.

[16] R. Katarya, G. A. Nath, D. Singhal and A. Shukla, "Analysing the Twitter Sentiments in COVID-19 Using Machine Learning Algorithms," *2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, 2022, pp. 1-8, doi: 10.1109/ACCAI53970.2022.9752511.

[17] R. Latifah, R. Baddalwan, P. Meilina, A. D. Saputra and Y. Adharani, "Sentiment Analysis of COVID-19 Vaccines from Indonesian Tweets and ws Headlines using Various Machine Learning Techniques," *2021 ernational Conference on Informatics, Multimedia, Cyber and Information tem (ICIMCIS)*, 2021, pp. 69-73, doi: 1109/ICIMCIS53775.2021.9699187.

[18] K. Kurnianingsih, N. F. Al Faridi Hadi, E. Dwi Wardihani, N. Kubota and W. H. Chin, "Ensemble Learning Based on Soft Voting for Detecting Methamphetamine in Urine," *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2020, pp. 1-6, doi: 10.1109/FUZZ48607.2020.9177622.

[19] A. J. Nair, V. G and A. Vinayak, "Comparative study of Twitter Sentiment On COVID - 19 Tweets," *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, 2021, pp. 1773-1778, doi: 10.1109/ICCMC51019.2021.9418320.

[20] U. N. Wisesty, R. Rismala, W. Munggana and A. Purwarianti, "Comparative Study of Covid-19 Tweets Sentiment Classification Methods," *2021 9th International Conference on Information and Communication Technology (ICoICT)*, 2021, pp. 588-593, doi: 10.1109/ICoICT52021.2021.9527533.

[21] P. Puvar et al., "Heart Disease Detection using Ensemble Learning Approach", *International Research Journal of Engineering and Technology (IRJET)*, vol. 8, pp. 1414-1418, May 2021.

[22] S. Mehta, P. Rana, S. Singh, A. Sharma and P. Agarwal, "Ensemble Learning Approach for Enhanced Stock Prediction," *2019 Twelfth International Conference on Contemporary Computing (IC3)*, 2019, pp. 1-5, doi: 10.1109/IC3.2019.8844891.

[23] H. R. Sanabila and W. Jatmiko, "Ensemble Learning on Large Scale Financial Imbalanced Data," *2018 International Workshop on Big Data and Information Security (IWBIS)*, 2018, pp. 93-98, doi: 10.1109/IWBIS.2018.8471702.

[24] J. Tie, X. Lei and Y. Pan, "Metabolite-disease association prediction algorithm combining DeepWalk and random forest," *in Tsinghua Science and Technology*, vol. 27, no. 1, pp. 58-67, Feb. 2022, doi: 10.26599/TST.2021.9010003.

[25] C. Zhao, Y. Xin, X. Li, Y. Yang, and Y. Chen, "A Heterogeneous Ensemble Learning Framework for Spam Detection in Social Networks with Imbalanced Data," *Applied Sciences*, vol. 10, no. 3, p. 936, Jan. 2020, doi: 10.3390/app10030936.

[26] Murni, T. Handhika, A. Fahrurozi, I. Sari, D. P. Lestari and R. I. M. Zen, "Hybrid Method for Sentiment Analysis Using Homogeneous Ensemble Classifier," *2019 2nd International Conference of Computer and Informatics Engineering (IC2IE)*, 2019, pp. 232-236, doi: 1109/IC2IE47452.2019.8940896.

[27] K. S. Gan, K. O. Chin, P. Anthony and S. V. Chang, "Homogeneous Ensemble FeedForward Neural Network in CIMB Stock Price Forecasting," *2018 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, 2018, pp. 1-6, doi: 10.1109/IICAIET.2018.8638452.

[28] Imamah and F. H. Rachman, "Twitter Sentiment Analysis of Covid-19 Using Term Weighting TF-IDF And Logistic Regresion," *2020 6th Information Technology International Seminar (ITIS)*, 2020, pp. 238-242, doi: 10.1109/ITIS50118.2020.9320958.

[29] S. Singh, K. Kumar and B. Kumar, "Sentiment Analysis of Twitter Data Using TF-IDF and Machine Learning Techniques," *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON)*, 2022, pp. 252-255, doi: 10.1109/COM-IT-CON54601.2022.9850477.

[30] R. Man and K. Lin, "Sentiment Analysis Algorithm Based on BERT and Convolutional Neural Network," *2021 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*, 2021, pp. 769-772, doi: 10.1109/IPEC51340.2021.9421110.

[31] A. A. Marouf, J. G. Rokne, and R. Alhajj, "Detecting and Understanding Sentiment Trends and Emotion Patterns of Twitter Users—A Study on the Demise of a Bollywood Celebrity," *Big Data and Cognitive Computing*, vol. 6, no. 4, p. 129, Oct. 2022, doi: 10.3390/bdcc6040129.

[32] H. Fang, G. Xu, Y. Long, and W. Tang, "An Effective ELECTRA-Based Pipeline for Sentiment Analysis of Tourist Attraction Reviews," *Applied Sciences*, vol. 12, no. 21, p. 10881, Oct. 2022, doi: 10.3390/app122110881.

[33] S. S. Aljameel et al., "A Sentiment Analysis Approach to Predict an Individual's Awareness of the Precautionary Procedures to Prevent COVID-19 Outbreaks in Saudi Arabia," *International Journal of Environmental Research and Public Health*, vol. 18, no. 1, p. 218, Dec. 2020, doi: 10.3390/ijerph18010218.

[34] M. Reusens, M. Reusens, M. Callens, S. vanden Broucke, and B. Baesens, "Comparison of Different Modeling Techniques for Flemish Twitter Sentiment Analysis," *Analytics*, vol. 1, no. 2, pp. 117–134, Oct. 2022, doi: 10.3390/analytics1020009.

[35] J. J. E. Macrohon, C. N. Villavicencio, X. A. Inbaraj, and J.-H. Jeng, "A Semi-)ervised Approach to Sentiment Analysis of Tweets during the 2022 lippine Presidential Election," *Information*, vol. 13, no. 10, p. 484, Oct. 2, doi: 10.3390/info13100484.

[36] A. Jenasamanta and S. Mohapatra, “An automated system for the assessment and grading of adolescent delinquency using a machine learning-based soft voting framework,” *Humanities and Social Sciences Communications*, vol. 9, no. 1, Oct. 2022, doi: 10.1057/s41599-022-01407-x.

[37] A. Taha, “Intelligent Ensemble Learning Approach for Phishing Website Detection Based on Weighted Soft Voting,” *Mathematics*, vol. 9, no. 21, p. 2799, Nov. 2021, doi: 10.3390/math9212799.

[38] R. Wardoyo, A. Musdholifah, G. Angga Pradipta and I. N. Hariyasa Sanjaya, “Weighted Majority Voting by Statistical Performance Analysis on Ensemble Multiclassifier,” *2020 Fifth International Conference on Informatics and Computing (ICIC)*, Gorontalo, Indonesia, 2020, pp. 1-8, doi: 10.1109/ICIC50835.2020.9288552.

[39] A. Dogan and D. Birant, “A Weighted Majority Voting Ensemble Approach for Classification,” *2019 4th International Conference on Computer Science and Engineering (UBMK)*, Samsun, Turkey, 2019, pp. 1-6, doi: 10.1109/UBMK.2019.8907028.

[40] R. Islam, M. I. Sayed, S. Saha, M. J. Hossain, and M. A. Masud, “Android malware classification using Optimum Feature Selection and Ensemble Machine Learning,” *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 100–111, 2023, doi:10.1016/j.iotcps.2023.03.001.

[41] P. Agrawal, H. F. Abutarboush, T. Ganesh and A. W. Mohamed, “Metaheuristic Algorithms on Feature Selection: A Survey of One Decade of Research (2009-2019),” *in IEEE Access*, vol. 9, pp. 26766-26791, 2021, doi: 10.1109/ACCESS.2021.3056407.

[42] T. M. Shami, A. A. El-Saleh, M. Alswaitti, Q. Al-Tashi, M. A. Summakieh and S. Mirjalili, “Particle Swarm Optimization: A Comprehensive Survey,” *in IEEE Access*, vol. 10, pp. 10031-10061, 2022, doi: 10.1109/ACCESS.2022.3142859.

[43] M. Grandini, Enrico Bagli, and Giorgio Visani, “Metrics for Multi-Class Classification: an Overview,” *arXiv (Cornell University)*, Aug. 2020, doi: https://doi.org/10.48550/arxiv.2008.05756.

LAMPIRAN

Lampiran 1. *Script* Program untuk Prapemrosesan Data

```python
import pandas as pd
import re
import nltk
from sklearn.pipeline import Pipeline
from textblob import TextBlob
nltk.download('stopwords')
from nltk.corpus import stopwords
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

def clean(data):

    def casefolding(tweet):
        tweet = tweet.lower()
        #tweet = tweet.strip(" ")
        tweet = re.sub(r'[\r\n]+', ' ',tweet)
        tweet = re.sub(r'https?:\/\/\S+', ' ',tweet)
        tweet = re.sub(r'@[\w]+', ' ',tweet)
        # tweet = re.sub(r'[^A-Za-z0-9\s]+| ', ' ',tweet)
        tweet = re.sub(r'[^A-Za-z\s]+|_', ' ',tweet)
        tweet = re.sub(r'[\W]+', ' ',tweet)
        return tweet
    data['tweet'] = data['tweet'].apply(casefolding)

    #Tokenizing
    def token(tweet):
        nstr = tweet.split(' ')
        dat= []
        for hu in nstr:
            if (hu != ''):
                dat.append(hu)
        return dat
    data['tweet'] = data['tweet'].apply(token)

    dictonary = { "knp": "kenapa", "bru": "baru", "lg": "lagi", "dr": "dari", "sdh": "sudah",
            "dg": "dengan", "tp": "tapi", "sblm": "sebelum", "jg": "juga", "sy": "saya",
            "gk": "gak", "bs": "bisa", "gpp": "gapapa", "tpi": "tapi", "sm": "sama",
            "bgt": "banget", "gmn": "bagaimana", "dgn": "dengan", "bsa": "bisa",
            "jd": "jadi", "tdk": "tidak", "spt": "seperti", "sdng": "sedang",
            "gini": "begini", "kmu": "kamu", "krn": "karena", "dlm": "dalam",
            "bukn": "bukan", "mnrt": "menurut", "trs": "terus", "sdikit": "sedikit", "sdkt": "sedikit",
            "brp": "berapa", "gtu": "begitu", "w": "aku", "kmn": "kemana", "lm": "lama",
            "smg": "semoga", "bnyk": "banyak", "mn": "mana", "slm": "selama", "gk": "gak",
            "btl": "betul", "bnr": "benar", "pd": "pada", "yg": "yang", "lw": "kalau",
            "mlm": "malam", "bsr": "besar", "org": "orang", "ntr": "nanti", "dlu": "dulu",
            "bwh": "bawah", "tmn": "teman", "syg": "sayang", "trus": "terus", "tp": "tapi",
            "thn": "tahun", "cma": "cuma", "smpe": "sampai", "pdhl": "padahal", "lh": "lah",
            "ny": "nya", "bbrp": "beberapa", "lmyn": "lumayan", "n": "dan",
            "pa": "apa", "tar": "sebentar", "bca": "baca", "lm": "lama", "ps": "pas",
            "lgsg": "langsung", "msh": "masih", "bnyk": "banyak", "mnrut": "menurut",
            "hnya": "hanya", "mslh": "masalah", "sih": "sih", "hri": "hari",
            "dri": "dari", "dkt": "dekat", "skr": "sekarang", "jm": "jam", "y": "iya",
            "k": "oke", "byk": "banyak", "ntar": "nanti", "msk": "masuk", "ktnya": "katanya",
            "mksh": "makasih", "br": "baru", "mgkn": "mungkin", "nanya": "bertanya",
            "mau": "ingin", "u": "kamu", "tngh": "tengah", "blm": "belum",
            "q": "aku", "nya": "dia", "s": "sudah", "btw": "by the way", "g": "ga",
            "w": "gw", "tw": "tahu", "kt": "kita", "bgs": "bagus", "tpi": "tapi", "sm": "sama",
            "jg": "juga", "pd": "pada", "sdh": "sudah", "dn": "dan", "bw": "bawa",
            "sndr": "sendiri", "jln": "jalan", "ngr": "negara", "gt": "gitu", "bhs": "bahasa",
            "klo": "kalau", "lh": "lah", "bgt": "banget", "sdikit": "sedikit",
            "tlg": "tolong", "plg": "pulang", "dl": "dulu", "sms": "pesan",
            "mksd": "maksud", "ngmng": "ngomong", "mlu": "malu", "thnks": "thanks",
            "bgtu": "begitu", "ktny": "katanya", "mksdny": "maksudnya",
            "mkasih": "makasih", "blg": "bilang", "tmn": "teman", "kbr": "kabar",
            "ksh": "kasih", "kmrn": "kemarin", "kbtulan": "kebetulan",
            "mnding": "mending", "brp": "berapa", "kluar": "keluar", "stlh": "setelah",
            "shbt": "sahabat", "pny": "punya", "mgkin": "mungkin", "mgk": "mungkin",
            "jwb": "jawab", "ntn": "nonton", "sblum": "sebelum", "byr": "bayar"
            }
    def ubah_singkatan(tweet):
        ubah = []
        for i in tweet:
            p = 0
            for j in dictonary:
                if i == j:
                    p=p+1
                    ubah.append(dictonary[j])
                    break
            if p == 0:
                ubah.append(i)
        return ubah
    data['tweet'] = data['tweet'].apply(ubah_singkatan)
```

```
#Filtering/Stopwords Removal
def stopword_removal(tweet):
    pkg_stopwords = stopwords.words('indonesian','english')
    more_stopwords = [
        'yg', 'utk', 'cuman', 'deh', 'btw', 'gua', 'gue', 'gw', 'lo', 'lu',
        'kalo', 'trs', 'jd', 'nih', 'ntar', 'nya', 'lg', 'gk', 'ecusli', 'dpt',
        'dr', 'kpn', 'kok', 'kyk', 'donk', 'dong', 'yah', 'u', 'ya', 'km', 'eh',
        'sih', 'eh', 'bang', 'br', 'rp', 'jt', 'kan', 'gpp', 'sm', 'usah',
        'mas', 'sob', 'thx', 'ato', 'jg', 'wkwk', 'mak', 'haha',' ly', 'k',
        'tp', 'hahaha', 'dg', 'dri', 'duh', 'ye', 'wkwkwk', 'syg', 'btw',
        'gaes', 'guys', 'moga', 'kmrn', 'nemu', 'yuk', 'yukk', 'yukkk',
        'klas', 'iw', 'ew', 'lho', 'loh', 'sbnry', 'org', 'gtu', 'bwt',
        'klrga', 'clau', 'lbh', 'cpet', 'ku', 'wke', 'mba', 'mas', 'sdh', 'kmrn',
        'oi', 'spt', 'dlm', 'bs', 'krn', 'sapa', 'spt', 'sh', 'wakakaka', 'wakakak',
        'sihh', 'sihhh', 'hehe', 'hehehe', 'ih', 'dgn', 'la', 'kl', 'ttg', 'mana', 'kmna', 'kmn',
        'tuh', 'dah', 'kek', 'ko', 'pls', 'bbrp', 'pd', 'mah', 'dhhh', 'dhh', 'dh',
        'kpd', 'tuh', 'kzl', 'sbl', 'byr', 'byar', 'sii', 'si', 'cm', 'cmn', 'sy', 'hahahaha', 'weh',
        'dlu', 'tuh', 'nah', 'naahhh', 'nahh'
    ]
    filtering = more_stopwords + pkg_stopwords
    x = []
    data = []
    def myFunc(x):
        if x in filtering:
            return False
        else:
            return True
    fit = filter(myFunc, tweet)
    for x in fit:
        data.append(x)
    return data
data['tweet'] = data['tweet'].apply(stopword_removal)

#Stemming
def stemming(tweet):
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()
    do=[]

    for w in tweet:
        dt = stemmer.stem(w)
        do.append(dt)

    d_clean=[]
    d_clean= " ".join(do)
    print(d_clean)
    return d_clean

data['tweet'] = data['tweet'].apply(stemming)

return data
```

Lampiran 2. *Script* Program untuk Vektorisasi Data menggunakan TF-IDF

```python
import math
import pandas as pd
import numpy as np

def tf(d_list, d_semua):
    data_dict = {}
    data_tf = []
    for kalimat in d_list:
        for kata in d_semua:
            data_dict[kata] = kalimat.split().count(kata)/len(kalimat.split())
        data_tf.append(data_dict)
        data_dict = {}
    return data_tf

def df(d_list, d_semua):
    df_dict = {}
    for kalimat in d_list:
        for term in d_semua:
            if term in kalimat.split():
                if term in df_dict:
                    df_dict[term] += 1
                else:
                    df_dict[term] = 1
            else:
                if term not in df_dict:
                    df_dict[term] = 0
    return df_dict

def idf(df_dict, d_list):
    data_idf = {}
    for kata in df_dict:
        hitung = math.log(len(d_list)/(df_dict[kata]+1))
        data_idf[kata] = hitung
    return data_idf

def tfidf(data_tf, data_idf):
    hasil = []
    data_sementara = {}
    for index in range(len(data_tf)):
        for i in data_tf[index]:
            kalkulasi = data_tf[index][i] * data_idf[i]
            data_sementara[i] = round(kalkulasi, 4)
        hasil.append(data_sementara)
        data_sementara = {}
    return hasil

def tfidf_proses(data, d_semua=[], idf_train={}):

    n=1
    if d_semua == []:
        n = 0
    d_list = []
    for i in data['tweet']:
        d_list.append(i)

    def token(tweet):
        nstr = tweet.split(' ')
        dat= []
        for hu in nstr:
            if (hu != ''):
                dat.append(hu)
        return dat
    data['tweet'] = data['tweet'].apply(token)

    if n == 0:
        d_semua = []
        for i in data['tweet']:
            for j in i:
                if j not in d_semua:
                    d_semua.append(j)

    #TF
    data_tf = tf(d_list, d_semua)

    #DF
    df_dict = df(d_list, d_semua)

    #IDF
    if idf_train == {}:
        data_idf = idf(df_dict, d_list)
    else:
        data_idf = idf_train

    #TFIDF
    hasil = tfidf(data_tf, data_idf)

    tfidf_data = pd.DataFrame(hasil)

    if n==0:
        all_data=tfidf_data.join(data['label'])
        all_data.head()

        return tfidf_data, all_data, d_semua, data_idf
    else:
        return tfidf_data
```

Lampiran 3. *Script* Program Algoritma *Naïve Bayes*

```python
import numpy as np

class MultiNB:
    def __init__(self,alpha=1):
        self.alpha = alpha

    def _prior(self): # CHECKED
        P = np.zeros((self.n_classes_))
        _, self.dist = np.unique(self.y,return_counts=True)
        for i in range(self.classes_.shape[0]):
            P[i] = self.dist[i] / self.n_samples
        return P

    def fit(self, X, y): # CHECKED, matches with sklearn
        self.y = y
        self.n_samples, self.n_features = X.shape
        self.classes_ = np.unique(y)
        self.n_classes_ = self.classes_.shape[0]
        self.class_priors_ = self._prior()
        # distinct values in each features
        self.uniques = []
        for i in range(self.n_features):
            tmp = np.unique(X[:,i])
            self.uniques.append( tmp )
        self.N_yi = np.zeros((self.n_classes_, self.n_features)) # feature count
        self.N_y = np.zeros((self.n_classes_)) # total count
        for i in self.classes_: # x axis
            indices = np.argwhere(self.y==i).flatten()
            columnwise_sum = []
            for j in range(self.n_features): # y axis
                columnwise_sum.append(np.sum(X[indices,j]))
            self.N_yi[i] = columnwise_sum # 2d
            self.N_y[i] = np.sum(columnwise_sum) # 1d
        return self

    def _theta(self, x_i, i, h):
        Nyi = self.N_yi[h,i]
        Ny  = self.N_y[h]
        numerator = Nyi + self.alpha
        denominator = Ny + (self.alpha * self.n_features)
        return  (numerator / denominator)**x_i

    def _likelyhood(self, x, h):
        tmp = []
        for i in range(x.shape[0]):
            tmp.append(self._theta(x[i], i,h))
        return np.prod(tmp)

    def predict(self, X):
        samples, features = X.shape
        self.predict_proba = np.zeros((samples,self.n_classes_))
        for i in range(X.shape[0]):
            joint_likelyhood = np.zeros((self.n_classes_))

            for h in range(self.n_classes_):
                joint_likelyhood[h]  = self.class_priors_[h] * self._likelyhood(X[i],h) # P(y) P(X|y)
            denominator = np.sum(joint_likelyhood)
            for h in range(self.n_classes_):
                numerator = joint_likelyhood[h]
                self.predict_proba[i,h] = (numerator / denominator)
        indices = np.argmax(self.predict_proba,axis=1)
        return self.classes_[indices], self.predict_proba
```

Lampiran 4. *Script* Program Algoritma SVM

```python
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.utils import check_random_state
from sklearn.preprocessing import LabelEncoder

def projection_simplex(v, z=1):
    n_features = v.shape[0]
    u = np.sort(v)[::-1]
    cssv = np.cumsum(u) - z
    ind = np.arange(n_features) + 1
    cond = u - cssv / ind > 0
    rho = ind[cond][-1]
    theta = cssv[cond][-1] / float(rho)
    w = np.maximum(v - theta, 0)
    return w

class MulticlassSVM(BaseEstimator, ClassifierMixin):

    def __init__(self, C=1, max_iter=50, tol=0.05,
                 random_state=None, verbose=0):
        self.C = C
        self.max_iter = max_iter
        self.tol = tol,
        self.random_state = random_state
        self.verbose = verbose

    def _partial_gradient(self, X, y, i):
        g = np.dot(X[i], self.coef_.T) + 1
        g[y[i]] -= 1
        return g

    def _violation(self, g, y, i):
        smallest = np.inf
        for k in range(g.shape[0]):
            if k == y[i] and self.dual_coef_[k, i] >= self.C:
                continue
            elif k != y[i] and self.dual_coef_[k, i] >= 0:
                continue
            smallest = min(smallest, g[k])
        return g.max() - smallest

    def _solve_subproblem(self, g, y, norms, i):
        Ci = np.zeros(g.shape[0])
        Ci[y[i]] = self.C
        beta_hat = norms[i] * (Ci - self.dual_coef_[:, i]) + g / norms[i]
        z = self.C * norms[i]
        beta = projection_simplex(beta_hat, z)
        return Ci - self.dual_coef_[:, i] - beta / norms[i]

    def fit(self, X, y):
        n_samples, n_features = X.shape
        # Normalize labels.
        self._label_encoder = LabelEncoder()
        y = self._label_encoder.fit_transform(y)
        # Initialize primal and dual coefficients.
        n_classes = len(self._label_encoder.classes_)
        self.dual_coef_ = np.zeros((n_classes, n_samples), dtype=np.float64)
        self.coef_ = np.zeros((n_classes, n_features))
        # Pre-compute norms.
        norms = np.sqrt(np.sum(X ** 2, axis=1))
        # Shuffle sample indices.
        rs = check_random_state(self.random_state)
        ind = np.arange(n_samples)
        rs.shuffle(ind)

        violation_init = None
        for it in range(self.max_iter):
            violation_sum = 0
            for ii in range(n_samples):
                i = ind[ii]
                # All-zero samples can be safely ignored.
                if norms[i] == 0:
                    continue
                g = self._partial_gradient(X, y, i)
                v = self._violation(g, y, i)
                violation_sum += v
                if v < 1e-12:
                    continue
                # Solve subproblem for the ith sample.
                delta = self._solve_subproblem(g, y, norms, i)
                # Update primal and dual coefficients.
                self.coef_ += (delta * X[i][:, np.newaxis]).T
                self.dual_coef_[:, i] += delta
            if it == 0:
                violation_init = violation_sum
            vratio = violation_sum / violation_init
            if vratio < self.tol:
                if self.verbose >= 1:
                    print("Converged")
                break
        return self

    def predict_proba(self, X):
        decision = np.dot(X, self.coef_.T)
        return decision

    def predict(self, X):
        decision = self.predict_proba(X)
        pred = decision.argmax(axis=1)
        return self._label_encoder.inverse_transform(pred)
```

Lampiran 5. *Script* Program Algoritma *Logistic Regression*

```python
import numpy as np

class SoftmaxRegression:
    def __init__(self,label,fitur):
        seed_value = 50
        np.random.seed(seed_value)
        self.weights = np.random.uniform(0, 1, (label, fitur))
        self.label = label

    def softmax(self, x):
        return np.exp(x) / np.sum(np.exp(x))

    def train_sgd(self, x_train, y_train, max_itr=1, alpha=0.1, lamda=0):
        for itr in range(max_itr):
            for i in range(len(x_train)):
                x = x_train[i].reshape(-1, 1)
                y = np.zeros((self.label, 1))
                y[y_train[i]] = 1
                h_y = self.softmax(np.dot(self.weights, x))
                self.weights = self.weights - alpha * (np.dot((h_y - y), x.T)) + alpha * lamda * self.weights
        return self.weights

    def train_bgd(self, x_train, y_train, max_itr=100, alpha=0.1, lamda=0):  # batch gradient decent
        print("-----Training-----")
        for itr in range(max_itr):
            print('iteration:%d' % itr)
            err = np.zeros((5, 14942))
            for i in range(len(x_train)):
                x = x_train[i].reshape(-1, 1)
                y = np.zeros((5, 1))
                y[y_train.iloc[i]] = 1
                h_y = self.softmax(np.dot(self.weights, x))
                singleErr = np.dot((h_y - y), x.T)
                err += singleErr
            err = err / len(x_train)
            regular = (alpha * lamda * self.weights) / len(x_train)
            self.weights = self.weights - alpha * err + regular
        print('-----Train Finished-----')
        return self.weights

    def predict_proba(self, x_test):
        y_proba = []
        for i in range(len(x_test)):
            x = x_test[i]
            y_proba.append(np.dot(self.weights, x))
        return y_proba

    def predict(self, x_test):
        y_predict = []
        for i in range(len(x_test)):
            x = x_test[i]
            y = np.argmax(np.dot(self.weights, x))
            y_predict.append(y)
        y_predict = np.array(y_predict)
        return y_predict

    def test(self, y_test, y_predict):
        print('-----Testing-----')
        accuracy = 0
        for i in range(len(y_test)):
            if y_predict[i] == y_test.iloc[i]:
                accuracy += 1
        accuracy = accuracy / len(y_test)
        print('-----Test Finished-----')
        return accuracy
```

Lampiran 6. *Script* Program *Confusion Matrix* untuk Menghitung Performa

```
def performance_test(y_test, predict):
    conf_matrix = confusion_matrix(y_test, predict)

    if len(conf_matrix) == 3:
        accuracy = (conf_matrix[0,0] + conf_matrix[1,1] + conf_matrix[2,2])/len(y_test)
        precision = ((conf_matrix[0,0]/(conf_matrix[0,0]+(conf_matrix[1,0]+conf_matrix[2,0]))) +
  (conf_matrix[1,1]/(conf_matrix[1,1]+(conf_matrix[0,1]+conf_matrix[2,1]))) + (conf_matrix[2,2]/(conf_matrix[2,2]+
  (conf_matrix[0,2]+conf_matrix[1,2])))) / 3
        recall = ((conf_matrix[0,0]/(conf_matrix[0,0]+(conf_matrix[0,1]+conf_matrix[0,2]))) +
  (conf_matrix[1,1]/(conf_matrix[1,1]+(conf_matrix[1,0]+conf_matrix[1,2]))) + (conf_matrix[2,2]/(conf_matrix[2,2]+
  (conf_matrix[2,0]+conf_matrix[2,1])))) / 3
        f1 = (2 * (precision * recall)) / (precision + recall)
    else:
        accuracy = (conf_matrix[0,0] + conf_matrix[1,1]) / len(y_test)
        precision = conf_matrix[1,1] / (conf_matrix[1,1]+conf_matrix[0,1])
        recall = conf_matrix[1,1]/(conf_matrix[1,1]+conf_matrix[1,0])
        f1 = (2 * (precision * recall)) / (precision + recall)

    return conf_matrix, accuracy, precision, recall, f1

```

Lampiran 7. *Script* Program *Heterogeneous Ensemble Learning*

```python
import numpy as np
import pandas as pd
import math
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
from scipy.sparse import csr_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest, chi2
from MLAlgorithm.NaiveBayes import MultiNB
from MLAlgorithm.LogisticRegression import SoftmaxRegression
from MLAlgorithm.SVM import MulticlassSVM
import pickle

training_data = pd.read_csv("E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/vector_save/multiclass_training_data_new.csv", encoding='utf-8')
X_train = csr_matrix(training_data.iloc[:, :-1]).toarray()
label_encoding = LabelEncoder()
y_train = label_encoding.fit_transform(training_data['label'])

#PROSES TRAINING
#Naive Bayes
NBModel = MultiNB()
NBModel.fit(X_train, y_train)
#SVM
SVMModel = MulticlassSVM(C=1, tol=0.05, max_iter=1000, random_state=0, verbose=1)
SVMModel.fit(X_train, y_train)
#Logistic Regression
LRModel = SoftmaxRegression(len(set(y_train)), len(X_train[0]))
LRModel.train_sgd(X_train, y_train)

testing_data = pd.read_csv("E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/vector_save/multiclass_testing_data_new.csv", encoding='utf-8')
X_test = csr_matrix(testing_data.iloc[:, :-1]).toarray()
y_test = label_encoding.fit_transform(testing_data['label'])

#PROSES TESTING
#Naive Bayes
NBPredicted, NBPredicted_prob = NBModel.predict(X_test)
#SVM
SVMPredicted_prob = SVMModel.predict_proba(X_test)
#Logistic Regression
LRPredicted_prob = LRModel.predict_proba(X_test)

#PROSES VOTING DENGAN WEIGHTED SOFT VOTING
with open('E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/optimal_weight/heterogeneous_optimal_weights_new.pkl', 'rb') as f:
    optimal_weights = pickle.load(f)
predicted_hetero = []
for baris in range(len(NBPredicted_prob)):
    jml = ((np.array(NBPredicted_prob[baris]) * optimal_weights[0]) + (np.array(SVMPredicted_prob[baris]) *
optimal_weights[1]) + (np.array(LRPredicted_prob[baris]) * optimal_weights[2])) / 3
    if jml.argmax() == 0:
        predicted_hetero.append(0)
    elif jml.argmax() == 1:
        predicted_hetero.append(1)
    elif jml.argmax() == 2:
        predicted_hetero.append(2)
predicted_hetero = np.array(predicted_hetero)

conf_matrix, accuracy, precision, recall, f1 = performance_test(y_test, predicted_hetero)
```

Lampiran 8. *Script* Program *Homogeneous Ensemble Learning Naïve Bayes*

```
import numpy as np
import pandas as pd
import math
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
from scipy.sparse import csr_matrix
from sklearn.preprocessing import LabelEncoder
from MLAlgorithm.NaiveBayes import MultiNB
import pickle

training_data = pd.read_csv("E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/vector_save/multiclass_training_data_new.csv", encoding='utf-8')

#PROSES TRAINING
n_sample = 10
models = []
for i in range(n_sample):
    sample_data = training_data.sample(frac=0.7, replace=False, random_state=i)

    X_train = csr_matrix(sample_data.iloc[:, :-1]).toarray()
    label_encoding = LabelEncoder()
    y_train = label_encoding.fit_transform(sample_data['label'])

    NBModel = MultiNB()
    NBModel.fit(X_train, y_train)
    models.append(NBModel)

testing_data = pd.read_csv("E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/vector_save/multiclass_testing_data_new.csv", encoding='utf-8')
X_test = csr_matrix(testing_data.iloc[:, :-1]).toarray()
y_test = label_encoding.fit_transform(testing_data['label'])

#PROSES TESTING
NBPredicted_prob = []
for model in models:
    predicted, prob = model.predict(X_test)
    NBPredicted_prob.append(prob)

#PROSES VOTING DENGAN WEIGHTED SOFT VOTING
with open('E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/optimal_weight/homogeneousNB_optimal_weights_new.pkl', 'rb') as f:
    optimal_weights = pickle.load(f)

predicted_homogen = []
for baris in range(len(NBPredicted_prob[0])):
    jml=0
    for model in range(len(NBPredicted_prob)):
        jml = jml + (np.array(NBPredicted_prob[model][baris]) * optimal_weights[model])
    jml = jml/len(NBPredicted_prob)
    if jml.argmax() == 0:
        predicted_homogen.append(0)
    elif jml.argmax() == 1:
        predicted_homogen.append(1)
    elif jml.argmax() == 2:
        predicted_homogen.append(2)
predicted_homogen = np.array(predicted_homogen)

conf_matrix, accuracy, precision, recall, f1 = performance_test(y_test, predicted_homogen)
```

Lampiran 9. *Script* Program *Homogeneous Ensemble Learning* SVM

```
import numpy as np
import pandas as pd
import math
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
from scipy.sparse import csr_matrix
from sklearn.preprocessing import LabelEncoder
from MLAlgorithm.SVM import MulticlassSVM
import pickle

training_data = pd.read_csv("E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/vector_save/multiclass_training_data_new.csv", encoding='utf-8')

#PROSES TRAINING
n_sample = 10
models = []
for i in range(n_sample):
    sample_data = training_data.sample(frac=0.7, replace=False, random_state=i)

    X_train = csr_matrix(sample_data.iloc[:, :-1]).toarray()
    label_encoding = LabelEncoder()
    y_train = label_encoding.fit_transform(sample_data['label'])

    SVMModel = MulticlassSVM(C=1, tol=0.05, max_iter=1000, random_state=0, verbose=1)
    SVMModel.fit(X_train, y_train)
    models.append(SVMModel)

testing_data = pd.read_csv("E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/vector_save/multiclass_testing_data_new.csv", encoding='utf-8')
X_test = csr_matrix(testing_data.iloc[:, :-1]).toarray()
y_test = label_encoding.fit_transform(testing_data['label'])

#PROSES TESTING
SVMPredicted_prob = []
for model in models:
    prob = model.predict_proba(X_test)
    SVMPredicted_prob.append(prob)

#PROSES VOTING DENGAN WEIGHTED SOFT VOTING
with open('E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/optimal_weight/homogeneousSVM_optimal_weights_new.pkl', 'rb') as f:
    optimal_weights = pickle.load(f)

predicted_homogen = []
for baris in range(len(SVMPredicted_prob[0])):
    jml=0
    for model in range(len(SVMPredicted_prob)):
        jml = jml + (np.array(SVMPredicted_prob[model][baris]) * optimal_weights[model])
    jml = jml/len(SVMPredicted_prob)
    if jml.argmax() == 0:
        predicted_homogen.append(0)
    elif jml.argmax() == 1:
        predicted_homogen.append(1)
    elif jml.argmax() == 2:
        predicted_homogen.append(2)
predicted_homogen = np.array(predicted_homogen)

conf_matrix, accuracy, precision, recall, f1 = performance_test(y_test, predicted_homogen)
```

Lampiran 10. *Script* Program *Homogeneous Ensemble Learning* LR

```python
import numpy as np
import pandas as pd
import math
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
from scipy.sparse import csr_matrix
from sklearn.preprocessing import LabelEncoder
from MLAlgorithm.LogisticRegression import SoftmaxRegression
import pickle

training_data = pd.read_csv("E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/vector_save/multiclass_training_data_new.csv", encoding='utf-8')

#PROSES TRAINING
n_sample = 10
models = []
for i in range(n_sample):
    sample_data = training_data.sample(frac=0.7, replace=False, random_state=i)

    X_train = csr_matrix(sample_data.iloc[:, :-1]).toarray()
    label_encoding = LabelEncoder()
    y_train = label_encoding.fit_transform(sample_data['label'])

    LRModel = SoftmaxRegression(len(set(y_train)), len(X_train[0]))
    LRModel.train_sgd(X_train, y_train)
    models.append(LRModel)

testing_data = pd.read_csv("E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/vector_save/multiclass_testing_data_new.csv", encoding='utf-8')
X_test = csr_matrix(testing_data.iloc[:, :-1]).toarray()
y_test = label_encoding.fit_transform(testing_data['label'])

#PROSES TESTING
LRPredicted_prob = []
for model in models:
    prob = model.predict_proba(X_test)
    LRPredicted_prob.append(prob)

#PROSES VOTING DENGAN WEIGHTED SOFT VOTING
with open('E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/optimal_weight/homogeneousLR_optimal_weights_new.pkl', 'rb') as f:
    optimal_weights = pickle.load(f)

predicted_homogen = []
for baris in range(len(LRPredicted_prob[0])):
    jml=0
    for model in range(len(LRPredicted_prob)):
        jml = jml + (np.array(LRPredicted_prob[model][baris]) * optimal_weights[model])
    jml = jml/len(LRPredicted_prob)
    if jml.argmax() == 0:
        predicted_homogen.append(0)
    elif jml.argmax() == 1:
        predicted_homogen.append(1)
    elif jml.argmax() == 2:
        predicted_homogen.append(2)

predicted_homogen = np.array(predicted_homogen)

conf_matrix, accuracy, precision, recall, f1 = performance_test(y_test, predicted_homogen)
```

Lampiran 11. Potongan *Script* Algoritma PSO untuk Model *Heterogeneous*

```python
from pyswarm import pso

def objective_function(weights):
    # Split bobot untuk masing-masing model
    nb_weight, svm_weight, lr_weight = weights
    #Naive Bayes
    NBModel = MultiNB()
    NBModel.fit(X_train, y_train)
    #SVM
    SVMModel = MulticlassSVM(C=1, tol=0.05, max_iter=1000, random_state=0, verbose=1)
    SVMModel.fit(X_train, y_train)
    #Logistic Regression
    LRModel = SoftmaxRegression(len(set(y_train)), len(X_train[0]))
    LRModel.train_sgd(X_train, y_train)

    #Naive Bayes
    NBPredicted, NBPredicted_prob = NBModel.predict(X_test)
    #SVM
    SVMPredicted_prob = SVMModel.predict_proba(X_test)
    #Logistic Regression
    LRPredicted_prob = LRModel.predict_proba(X_test)

    predicted_hetero = []
    for baris in range(len(NBPredicted_prob)):
        jml = ((np.array(NBPredicted_prob[baris]) * nb_weight) + (np.array(SVMPredicted_prob[baris]) * svm_weight)
+ (np.array(LRPredicted_prob[baris]) * lr_weight)) / 3
        if jml.argmax() == 0:
            predicted_hetero.append(0)
        elif jml.argmax() == 1:
            predicted_hetero.append(1)
        elif jml.argmax() == 2:
            predicted_hetero.append(2)
    predicted_hetero = np.array(predicted_hetero)

    conf_matrix, accuracy, precision, recall, f1 = performance_test(y_test, predicted_hetero)

    return -accuracy

# Batasan bobot (dalam rentang [0, 1])
lb = [0, 0, 0]
ub = [1, 1, 1]

# Jalankan algoritma PSO untuk mencari bobot optimal
best_weights, _ = pso(objective_function, lb, ub, swarmsize=10, maxiter=30)
optimal_nb_weight, optimal_svm_weight, optimal_lr_weight = best_weights
optimal_weights = [optimal_nb_weight, optimal_svm_weight, optimal_lr_weight]

with open('E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/optimal_weight/heterogeneous_optimal_weights_new.pkl', 'wb') as f:
    pickle.dump(optimal_weights, f)
```

Lampiran 12. Potongan *Script* Algoritma PSO Model *Homogeneous* NB

```
from pyswarm import pso

def objective_function(weights):
    n_sample = 10
    models = []
    for i in range(n_sample):
        sample_data = training_data.sample(frac=0.7, replace=False, random_state=i)

        X_train = csr_matrix(sample_data.iloc[:, :-1]).toarray()
        label_encoding = LabelEncoder()
        y_train = label_encoding.fit_transform(sample_data['label'])

        NBModel = MultiNB()
        NBModel.fit(X_train, y_train)
        models.append(NBModel)

    NBPredicted_prob = []
    NBAccuracy = []
    for model in models:
        predicted, prob = model.predict(X_test)
        NBPredicted_prob.append(prob)

    predicted_homogen = []
    for baris in range(len(NBPredicted_prob[0])):
        jml=0
        for model in range(len(NBPredicted_prob)):
            jml = jml + (np.array(NBPredicted_prob[model][baris]) * weights[model])
        jml = jml/len(NBPredicted_prob)
        if jml.argmax() == 0:
            predicted_homogen.append(0)
        elif jml.argmax() == 1:
            predicted_homogen.append(1)
        elif jml.argmax() == 2:
            predicted_homogen.append(2)

    predicted_homogen = np.array(predicted_homogen)

    conf_matrix, accuracy, precision, recall, f1 = performance_test(y_test, predicted_homogen)

    return -accuracy

# Batasan bobot (dalam rentang [0, 1])
jml = 10
lb = [0 for _ in range(10)]
ub = [1 for _ in range(10)]

# Jalankan algoritma PSO untuk mencari bobot optimal
best_weights, _ = pso(objective_function, lb, ub, swarmsize=10, maxiter=5)
optimal_weights = best_weights

with open('E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/optimal_weight/homogeneousNB_optimal_weights_new.pkl', 'wb') as f:
    pickle.dump(optimal_weights, f)
```

Lampiran 13. Potongan *Script* Algoritma PSO Model *Homogeneous* SVM

```python
from pyswarm import pso

def objective_function(weights):
    n_sample = 10
    models = []
    for i in range(n_sample):
        sample_data = training_data.sample(frac=0.7, replace=False, random_state=i)

        X_train = csr_matrix(sample_data.iloc[:, :-1]).toarray()
        label_encoding = LabelEncoder()
        y_train = label_encoding.fit_transform(sample_data['label'])

        SVMModel = MulticlassSVM(C=1, tol=0.05, max_iter=1000, random_state=0, verbose=1)
        SVMModel.fit(X_train, y_train)
        models.append(SVMModel)

    SVMPredicted_prob = []
    SVMAccuracy = []
    for model in models:
        prob = model.predict_proba(X_test)
        SVMPredicted_prob.append(prob)

    predicted_homogen = []
    for baris in range(len(SVMPredicted_prob[0])):
        jml=0
        for model in range(len(SVMPredicted_prob)):
            jml = jml + (np.array(SVMPredicted_prob[model][baris]) * weights[model])
        jml = jml/len(SVMPredicted_prob)
        if jml.argmax() == 0:
            predicted_homogen.append(0)
        elif jml.argmax() == 1:
            predicted_homogen.append(1)
        elif jml.argmax() == 2:
            predicted_homogen.append(2)

    predicted_homogen = np.array(predicted_homogen)

    conf_matrix, accuracy, precision, recall, f1 = performance_test(y_test, predicted_homogen)

    return -accuracy

# Batasan bobot (dalam rentang [0, 1])
jml = 10
lb = [0 for _ in range(10)]
ub = [1 for _ in range(10)]

# Jalankan algoritma PSO untuk mencari bobot optimal
best_weights, _ = pso(objective_function, lb, ub, swarmsize=10, maxiter=30)
optimal_weights = best_weights

with open('E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/optimal_weight/homogeneousSVM_optimal_weights_new.pkl', 'wb') as f:
    pickle.dump(optimal_weights, f)
```

Lampiran 14. Potongan *Script* Algoritma PSO Model *Homogeneous* LR

```python
from pyswarm import pso

def objective_function(weights):
    n_sample = 10
    models = []
    for i in range(n_sample):
        sample_data = training_data.sample(frac=0.7, replace=False, random_state=i)

        X_train = csr_matrix(sample_data.iloc[:, :-1]).toarray()
        label_encoding = LabelEncoder()
        y_train = label_encoding.fit_transform(sample_data['label'])

        LRModel = SoftmaxRegression(len(set(y_train)), len(X_train[0]))
        LRModel.train_sgd(X_train, y_train)
        models.append(LRModel)

    LRPredicted_prob = []
    LRAccuracy = []
    for model in models:
        prob = model.predict_proba(X_test)
        LRPredicted_prob.append(prob)

    predicted_homogen = []
    for baris in range(len(LRPredicted_prob[0])):
        jml=0
        for model in range(len(LRPredicted_prob)):
            jml = jml + (np.array(LRPredicted_prob[model][baris]) * weights[model])
        jml = jml/len(LRPredicted_prob)
        if jml.argmax() == 0:
            predicted_homogen.append(0)
        elif jml.argmax() == 1:
            predicted_homogen.append(1)
        elif jml.argmax() == 2:
            predicted_homogen.append(2)

    predicted_homogen = np.array(predicted_homogen)

    conf_matrix, accuracy, precision, recall, f1 = performance_test(y_test, predicted_homogen)

    return -accuracy

# Batasan bobot (dalam rentang [0, 1])
jml = 10
lb = [0 for _ in range(10)]
ub = [1 for _ in range(10)]

# Jalankan algoritma PSO untuk mencari bobot optimal
best_weights, _ = pso(objective_function, lb, ub, swarmsize=10, maxiter=30)
optimal_weights = best_weights

with open('E:/Python/Penelitian Tesis/Sistem/Ready Sistem
New/optimal_weight/homogeneousLR_optimal_weights_new.pkl', 'wb') as f:
    pickle.dump(optimal_weights, f)
```