DAFTAR PUSTAKA

[1] Saice, D.2019. The Number of Tweets per Day in 2019. Diakses pada 17/09/2020 https://www.dsayce.com/social-media/tweets-day/

[2] Lunando, E. dan Purwarianti, A. 2013. Indonesian Social Media Sentiment Analysis with Sarcasm Detection. *International Conference on Advanced Computer Science and Information Systems (ICACSIS).* ICACSIS, Bali. hal. 195-198, doi: 10.1109/ICACSIS.2013.6761575.

[3] Rajadesingan, A., Zafarani,R., dan Liu, H. 2015. Sarcasm Detection on Twitter: A Behavioral Modeling Approach. WSDM 2015 - *Proceedings of the 8th ACM International Conference on Web Search and Data Mining*. WDSM, hal. 97-106. doi: 10.1145/2684822.2685316.

[4] Bouazizi, M. dan Ohtsuki, T. 2016. A Pattern-Based Approach for Sarcasm Detection on Twitter. IEEE Access, vol. 4, hal. 5477-5488. doi: 10.1109/ACCESS.2016.2594194.

[5] Prasad dkk. 2017. Sentiment Analysis for Sarcasm Detection on Streaming Short Text Data. *2nd International Conference on Knowledge Engineering and Applications (ICKEA).* IEEE, London, hal. 1-5, doi: 10.1109/ICKEA.2017.8169892.

[6] Septiani, L. dan Sibaroni, Y. 2019. Sentiment Analysis Terhadap Tweet Bernada Sarkasme Berbahasa Indonesia. Jurnal Linguistik Komputasional, 2(2), hal. 62 - 67. http://inacl.id/journal/index.php/jlk/article/view/23

[7] Yunitasari, Y., Musdholifah, A., dan Sari, A. K. 2019. Sarcasm Detection for Sentiment Analysis in Indonesian Language Tweets. IJCCS (Indonesian J. Comput. Cybern. Syst., vol. 13, no. 1, p. 53.

[8] Twitter Inc. 2019. About Twitter's API. Diakses pada 17/09/2020 https://help.twitter.com/en/rules-and-policies/twitter-api

[9] Gusriani, S., Wardhani, K. D. K., dan Zul, M. I. 2016. Analisis Sentimen Terhadap Toko Online di Sosial Media Menggunakan Metode Klasifikasi

Naïve Bayes (Studi Kasus: Facebook Page BerryBenka). 4th *Applied Business and Engineering Conference*. Politeknik Caltex, Riau.

[10] Luo, T. dkk. 2013. Sentiment Analysis. Dalam: Luo, T. dkk. Trust-based Collective View Prediction. Springer, New York, hal. 53-68.

[11] Dinari, I. 2015. Jenis-Jenis Penanda Majas Sarkasme dalam Novel The Return of Sherlock Holmes. *Prosiding Seminar Naional PRASASTI II "Kajian Pragmatik dalam Berbagai Bidang*". Universitas Sebelas Maret, Surakarta, hal. 497–503.

[12] Kurnianti, M. P. 2020. Gaya Bahasa Ironi, Sinisme dan Sarkasme dalam Situs Artikel Opini Mojo.co. Skripsi. Universitas Sanata Dharma, Yogyakarta.

[13] Bowes, A. dan Katz, A. N. 2014. When Sarcasm Stings. Discourse Processes, 4, hal. 215-236.

[14] Melita, R., Amrizal, V., Suseno, H. B., dan Dirjam, T. 2018. Penerapan Metode Term Frequency Inverse Document Frequency (Tf-Idf) dan Cosine Similarity pada Sistem Temu Kembali Informasi untuk Mengetahui Syarah Hadits Berbasis Web (Studi Kasus : Syarah Umdatil Ahkam). JURNAL TEKNIK INFORMATIKA, 11, hal. 149-164.

[15] Andayani, S. dan Ryansyah, A. 2017. Implementasi Algoritma TF-IDF Pada Pengukuran Kesamaan Dokumen. JuSiTik : Jurnal Sistem dan Teknologi Informasi Komunikasi, 1(1), hal. 53-62. http://ojs.ukmc.ac.id/index.php/JUTSI/article/view/218

[16] Komariah, S. 2016. Interjeksi dalam Novel Donyane Wong Culika Karya Suparta Brata. Jurnal Balai Bahasa Jawa Timur, 4(1), hal. 55-64.

[17] Baccianella, S., Esuli, A., dan Sebastiani, F. 2010. SENTI WORD NET 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2010, 17-23 May 2010, Valletta, Malta.* European Language Resources Association.

[18] Syarli dan Muin, A. A. 2016. Metode Naive Bayes Untuk Prediksi Kelulusan (Studi Kasus : Data Mahasiswa Baru Perguruan Tinggi). Jurnal Ilmu Komputer, 2(1), hal. 22–26.

[19] Saleh, A. 2016. Implementasi Metode Klasifikasi Naïve Bayes Dalam Memprediksi Besarnya Penggunaan Listrik Rumah Tangga. Creative Information Technology Journal, 2, hal. 207-217.

[20] Haristu, R. A. 2019. Penerapan Metode Random Forest untuk Prediksi Win Ratio Pemain Player Unkown BattleGround. Skripsi. Universitas Sanata Dharma, Yogyakarta.

[21] Cutler, A., Cutler, D .R., dan Stevens. J. R. 2012. Random Forests. Dalam: Zhang C., Ma Y. (eds) Ensemble Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-9326-7_5

[22] Hartati, A.,Zain, I., dan Ulama, B. 2012. Analisis CART (Classification And Regression Trees) pada Faktor-Faktor yang Mempengaruhi Kepala Rumah Tangga di Jawa Timur Melakukan Urbanisasi. *J. SAINS DAN SENI ITS*, 1(1), hal. 100-105

[23] Baker D. 2017. gen_emoji_dict.py. Diakses pada 09/17/2020 https://gist.github.com/dnbaker/4c6576b706d6ec61308db7638c263a71

[24] Fikri dan Sarno. 2019. A Comparative Study of Sentiment Analysis using SVM and Sentiwordnet Indonesian. Journal of Electrical Engineering and Computer Science, 13(3), hal. 902-909

[25] Denecke. 2008. Using SentiWordNet for Multilingual Sentiment Analysis. *Proceedings International Conference on Data Engineering*. IEEE, Cancún, México, hal 505-512

[26] Lailiyah M, Sumpeno S, Purnama I.K.E. 2017. Sentiment Analysis of Public Complaint Using Lexical Resources between Indonesian Sentiment Lexicon and Sentiwordnet.*Proceedings of International Seminar on Intelligent Technology and Its Application 2017*, IEEE, Surabaya, hal 307-312.

LAMPIRAN

Lampiran 1 Tabel Keyword dan Tanggal Crawling Data

Keyword	Tanggal
#UsutJembatan5M	24 Desember 2019
#TangkapAriAskharaSEKARANG	27 Desember 2019
#AniesBaswedanJuaraBOHONG	25 Februari 2020
#AniesTenggelamkanDKI	25 Februari 2020
#PapuaNKRImaju	3 Maret 2020
#GejayanGerakanProvokasi	10 Maret 2020
#OmnibusLaw	11 Maret 2020
#MarufAminNgapain	11 Maret 2020
#AbahBekerja	11 Maret 2020
#BongkarSkandalFormulaE	12 Maret 2020
#BauAmiesPolitisirCorona	13 Maret 2020
#AniesKeren	13 April 2020
#UndipKokJahatSih	3 Mei 2020
pepres	17 Mei 2020
Bpjs	19 Mei 2020
#BalikinDanaHaji	3 Juni 2020
#JokowiDekatDihati	9 Juni 2020
#KalungAntiBego	5 Juli 2020
Kementan	5 Juli 2020
Bu susi	10 Juli 2020

Lampiran 2. Class DataPreparation

```
class DataPreparation():

  def preprocessData(self,data):
    processed_tweets = []
    for tweet in range(0, len(data)):
      #remove unicode
      processed_tweet = data[tweet].encode().decode('unicode_escape')
      processed_tweet = processed_tweet.encode('ascii', errors='ignore').decode("utf-
                        8")
      #remove url
      processed_tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','',processed_tweet)
      #Remove username
      processed_tweet = re.sub('@[^ \s]+','',processed_tweet)
      # Remove all the special characters
      processed_tweet = re.sub(r'\W', ' ', processed_tweet)
      # remove all single characters
      processed_tweet = re.sub(r'\s+[a-zA-Z]\s+', ' ', processed_tweet)
      # Remove single characters from the start
      processed_tweet = re.sub(r'\^[a-zA-Z]\s+', ' ', processed_tweet)
      # Remove single characters from the start
      processed_tweet = re.sub(r'[0-9]+', '', processed_tweet)#Remove number
      processed_tweet = re.sub(r'^b\s+', '', processed_tweet) # Removing prefixed 'b'
      # Substituting multiple spaces with single space
      processed_tweet= re.sub(r'\s+', ' ', processed_tweet, flags=re.I)
      processed_tweet = processed_tweet.lower()# Converting to Lowercase
      processed_tweets.append(processed_tweet)
    return processed_tweets
def stemmingTweet(self,data):
    stemmer = StemmerFactory().create_stemmer()
    stemmed_data = []
    for tweet in data:
      stemmed_data.append(stemmer.stem(tweet)) # Stemming data
    return stemmed_data
def extractEmoji(self,data):
    from ipynb.fs.full.EmojidictCopy1 import emoji_dict
    emoji,tweets_em = "", []
    for i in range(len(data)):
      processed_tweet = re.sub(r'\\', '/', data[i])
      for j in emoji_dict.keys():
```

```
            if(re.search(j,processed_tweet)):
                emoji = " " + emoji  + str(emoji_dict[j])
                j = re.sub(r'/', '\\\\', j)
                data[i]=data[i].replace(j, emoji)
        tweets_em.append(str(data[i]))
        emoji = ""
    return tweets_em

def splitHashtag(self,data):
    t_hashtag = []
    for x in data:
        hashtag = re.findall(r"#(\w+)", x)
        for letter in hashtag:
            splitted=''
            for i in range(len(letter)):
                if len(splitted)==0: splitted+=" "+letter[i]
                if(i+1 <len(letter)):
                    if letter[i+1].islower():
                        splitted += letter[i+1]
                    elif letter[i].isupper() and letter[i+1].isupper():
                        splitted += letter[i+1]
                    else:
                        splitted+=" "+letter[i+1]
            x=x.replace("#"+letter, splitted)
        t_hashtag.append(x)
    return t_hashtag
  def removeStopword(self, data):
    stopword = ['yang', 'untuk', 'pada', 'ke', 'para', 'namun', 'menurut', 'antara', 'dia', 'dua',
                'ia', 'seperti','jika','sehingga', 'kembali', 'dan', 'ini', 'karena', 'kepada',
                'oleh', 'saat', 'harus', 'sementara', 'setelah', 'kami', 'sekitar', 'bagi', 'serta',
                'di', 'dari', 'telah', 'sebagai', 'masih', 'hal', 'ketika', 'adalah', 'itu', 'dalam',
                'bahwa', 'atau', 'hanya', 'kita', 'dengan', 'akan', 'juga', 'ada', 'mereka',
                'saya', 'terhadap', 'secara', 'agar', 'lain', 'anda', 'begitu', 'mengapa',
                'kenapa', 'yaitu', 'yakni', 'daripada', 'itulah', 'lagi', 'maka', 'tentang',
                'demi', 'dimana', 'kemana', 'pula', 'sambil', 'sebelum', 'sesudah', 'supaya',
                'guna', 'kah', 'pun', 'sampai', 'sedangkan', 'selagi',  'sementara',
                'tetapi', 'apakah', 'kecuali', 'sebab', 'selain', 'seolah', 'seraya','seterusnya',
                'agak', 'boleh', 'dapat', 'dsb', 'dst', 'dll', 'dahulu', 'dulunya', 'anu',
                'demikian', 'tapi', 'ingin', 'juga', 'mari', 'nanti', 'melainkan',  'ok',
                'seharusnya', 'sebetulnya', 'setiap', 'setidaknya', 'sesuatu', 'pasti',
                'saja', 'toh', 'ya', 'walau', 'tolong', 'tentu', 'amat', 'apalagi', 'pak'
                'bagaimanapun', 'yg' ,'nya', 'nyang', 'nih', 'gue', 'di', 'ini', 'klo', 'kalau', 'n',
                'pak', 'kok', 'dong', 'nyuk', 'lah', 'dia', 'krn', 'utk', 'nyg', 'jadi', 'aja', 'apa',
                'nkri', 'garuda', 'seharusnya', 'sebetulnya', 'setiap', 'setidaknya', 'sesuatu',
                'pasti', 'saja', 'toh', 'ya', 'walau', 'tolong', 'tentu', 'amat', 'apalagi',  'kok',
                'bagaimanapun', 'yg' ,'nya', 'nyang', 'nih', 'gue', 'di', 'ini', 'klo', 'kalau', 'n',
```

```
            if(re.search(j,processed_tweet)):
                emoji = " " + emoji  + str(emoji_dict[j])
                j = re.sub(r'/', '\\\\', j)
                data[i]=data[i].replace(j, emoji)
        tweets_em.append(str(data[i]))
        emoji = ""
    return tweets_em

def splitHashtag(self,data):
    t_hashtag = []
    for x in data:
        hashtag = re.findall(r"#(\w+)", x)
        for letter in hashtag:
            splitted=''
            for i in range(len(letter)):
                if len(splitted)==0: splitted+=" "+letter[i]
                if(i+1 <len(letter)):
                    if letter[i+1].islower():
                        splitted += letter[i+1]
                    elif letter[i].isupper() and letter[i+1].isupper():
                        splitted += letter[i+1]
                    else:
                        splitted+=" "+letter[i+1]
            x=x.replace("#"+letter, splitted)
        t_hashtag.append(x)
    return t_hashtag
 def removeStopword(self, data):
    stopword = ['yang', 'untuk', 'pada', 'ke', 'para', 'namun', 'menurut', 'antara', 'dia', 'dua',
                'ia', 'seperti','jika','sehingga', 'kembali', 'dan', 'ini', 'karena', 'kepada',
                'oleh', 'saat', 'harus', 'sementara', 'setelah', 'kami', 'sekitar', 'bagi', 'serta',
                'di', 'dari', 'telah', 'sebagai', 'masih', 'hal', 'ketika', 'adalah', 'itu', 'dalam',
                'bahwa', 'atau', 'hanya', 'kita', 'dengan', 'akan', 'juga', 'ada', 'mereka',
                'saya', 'terhadap', 'secara', 'agar', 'lain', 'anda', 'begitu', 'mengapa',
                'kenapa', 'yaitu', 'yakni', 'daripada', 'itulah', 'lagi', 'maka', 'tentang',
                'demi', 'dimana', 'kemana', 'pula', 'sambil', 'sebelum', 'sesudah', 'supaya',
                'guna', 'kah', 'pun', 'sampai', 'sedangkan', 'selagi',  'sementara',
                'tetapi', 'apakah', 'kecuali', 'sebab', 'selain', 'seolah', 'seraya','seterusnya',
                'agak', 'boleh', 'dapat', 'dsb', 'dst', 'dll', 'dahulu', 'dulunya', 'anu',
                'demikian', 'tapi', 'ingin', 'juga', 'mari', 'nanti', 'melainkan',  'ok',
                'seharusnya', 'sebetulnya', 'setiap', 'setidaknya', 'sesuatu', 'pasti',
                'saja', 'toh', 'ya', 'walau', 'tolong', 'tentu', 'amat', 'apalagi', 'pak'
                'bagaimanapun', 'yg' ,'nya', 'nyang', 'nih', 'gue', 'di', 'ini', 'klo', 'kalau', 'n',
                'pak', 'kok', 'dong', 'nyuk', 'lah', 'dia', 'krn', 'utk', 'nyg', 'jadi', 'aja', 'apa',
                'nkri', 'garuda', 'seharusnya', 'sebetulnya', 'setiap', 'setidaknya', 'sesuatu',
                'pasti', 'saja', 'toh', 'ya', 'walau', 'tolong', 'tentu', 'amat', 'apalagi',  'kok',
                'bagaimanapun', 'yg' ,'nya', 'nyang', 'nih', 'gue', 'di', 'ini', 'klo', 'kalau', 'n',
```

```
def translateTweet(self, data):
    translator = Translator()
    translate = []
    for x in data:
        t = translator.translate(x, src='id', dest='en').text
        translate.append(t)
    return translate

def splitTweet(self,data):
    awal = ""
    akhir = ""
    t_awal = []
    t_akhir = []
    for ptweet in data:
        tokens = nltk.word_tokenize(ptweet)
        n = len(tokens)
        awal  = ' '.join(tokens[0:int(n/2)])
        akhir = ' '.join(tokens[int(n/2):n])
        t_awal.append(awal)
        t_akhir.append(akhir)
        awal=""
```

Lampiran 3. Kelas TF-IDF

```
class TFIDFValue():
  def word_freq(self, processed_tweets,freq):
    wordfreq = {}
    for tweet in processed_tweets:
      for token in nltk.word_tokenize(tweet):
        wordfreq[token] = 1 if token not in wordfreq.keys() else wordfreq[token]+1
    most_freq = heapq.nlargest(freq, wordfreq, key=wordfreq.get)
    return most_freq

  def idf_value(self,most_freq, processed_tweets):
    idf_values = {}
    for token in most_freq:
      token_tweet = 0
      for tweet in processed_tweets:
        if token in nltk.word_tokenize(tweet):
          token_tweet +=1
      idf_values[token]=0 if token_tweet==0 else np.log(len(processed_tweets)
                                                        /(token_tweet))
    return idf_values
```

```
def tf_value(self,most_freq, processed_tweets):
        word_tf_values = {}
        for token in most_freq:
            tf = []
            for tweet in processed_tweets:
                term_freq = 0
                for word in nltk.word_tokenize(tweet):
                    if token == word: term_freq +=1
                word_tf = term_freq/len(nltk.word_tokenize(tweet))
                tf.append(word_tf)
            word_tf_values[token] =tf
        return word_tf_values

    def tf_idf_value(self,word_tf_values, idf_values):
        tfidf_x = []
        for token in word_tf_values.keys():
            tfidf_tweet = []
            for tf_tweet in word_tf_values[token]:
                tfidf_tweet.append(tf_tweet * idf_values[token])
            tfidf_x.append(tfidf_tweet)
        return tfidf_x

    def main(self,mostfreqword,processed_tweets):
        #mostfreqword = self.word_freq(processed_tweets)
        tweetidfvalue = self.idf_value(mostfreqword,processed_tweets)
        tweettfvalue = self.tf_value(mostfreqword, processed_tweets)
        tweettfidfvalue = self.tf_idf_value(tweettfvalue, tweetidfvalue)
        Xn = np.asarray(tweettfidfvalue)
        Xn = np.transpose(Xn)
        return Xn
```

Lampiran 4. Kelas Naïve Bayes

```
class NaiveBayes():
    def mean(self,numbers):
        return sum(numbers)/float(len(numbers))
    def variance(self,numbers,_epsilon):
        avg = self.mean(numbers)
        v = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
        return sqrt(v) + _epsilon
    def divide_by_class(self,dataset, label):
        division = dict()
        for i in range(len(dataset)):
            vector = dataset[i]
            class_label = label[i]
```

```
        if (class_label not in division):
            division[class_label] = list()
        division[class_label].append(vector)
    return division
def dataset_values(self,dataset,_epsilon):
    values = [(self.mean(column), self.variance(column,_epsilon), len(column)) for
                column in zip(*dataset)]
    return value
#Mean, variance tiap kelas
def values_by_class(self,dataset, label):
    division = self.divide_by_class(dataset, label)
    values = dict()
    for class_label, rows in division.items():
        values[class_label] = self.dataset_values(rows,0)
    l=[]
    for x in list(values.values()):
        for y in x:
            l.append(y[1])
    var_smoothing = 1e-3
    _epsilon = var_smoothing * max(l)
    values = dict()
    for class_label, rows in division.items():
        values[class_label] = self.dataset_values(rows,_epsilon)
    return values
def calculate_probability(self,x, mean, variance):
    exponent = exp(-((x-mean)**2 / (2 * variance**2 )))
    return (1 / (sqrt(2 * pi) * variance)) * exponent
def calculate_class_probabilities(self,values, row):
    total_rows = sum([values[label][0][2] for label in values])
    probabilities = dict()
    for class_label, class_values in values.items():
        probabilities[class_label] = values[class_label][0][2]/float(total_rows)
        for i in range(len(class_values)):
            mean, stdev, _ = class_values[i]
            probabilities[class_label] *= self.calculate_probability(row[i], mean, stdev)
    return probabilities
def predict_nb(self,values, row):
    probabilities = self.calculate_class_probabilities(values, row)
    best_label, best_prob = None, -1
    for class_label, probability in probabilities.items():
        if best_label is None or probability > best_prob:
            best_prob = probability
            best_label = class_label
    return best_label
```

```
def naive_bayes(self,values,x_test):
        predictions = list()
        for row in x_test:
            output = self.predict_nb(values, row)
            predictions.append(output)
        return predictions
```

Lampiran 5. Kelas Sentiment Score

```
class SentimentScore():
    import re
    from nltk.corpus import stopwords
    def split_line(self, line):
        cols = line.split("\t")
        return cols
    def get_words(self,cols):
        words_ids = cols[4].split(" ")
        words = [w.split("#")[0] for w in words_ids]
        return words
    def get_positive(self,cols):
        col = float(cols[2])
        if (col>=0.5):
            col = col*3
        return col
    def get_scores(self,filepath, text):
def get_scores(self,filepath, text):
        f = open(filepath)
        count =0.0
        totalpositive =0.0
        totalnegative =0.0
        for line in f:
            if not line.startswith("#"):
                cols = self.split_line(line)
                words = self.get_words(cols)
                for i in range(len(text)):
                    if text[i] in words:
                        if text[i-1] == "not":
                            totalnegative = totalnegative +float(self.get_positive(cols))
                            totalpositive = totalpositive + float(self.get_negative(cols))
                        else:
                            totalpositive = totalpositive + float(self.get_positive(cols))
                            totalnegative = totalnegative + float(self.get_negative(cols))
                        count =count + 1
        #sentimentscore = ((p*Pw + pw)-(p*Nw+nw))/((p.Pw+pw)+(p.Nw+nw))
```

```
        if(totalpositive ==0 and totalnegative == 0):
            sentimentscore = 0
        else:
            sentimentscore = (totalpositive-totalnegative)/(totalpositive+totalnegative)
        return sentimentscore

    def main(self, data):
        from nltk.corpus import stopwords
        scores = []
        for tweet in data:
            text = str(tweet).split(" ")
            filtered_sentence = [w for w in text]
             scores.append(self.get_scores("C:/Users/asus/Documents/MateriKuliah/smstr6/
             MetodePenelitian/sacrasm/sentiwordnet/sentiwordnet/SentiWordNet_3.0.0.txt",f
             iltered_sentence))
        return scores
```

Lampiran 6. Kelas *Random Forest*

```
class RandomForest():
    def bootstrapping(self,train_df, n_bootstrap,seed):
        random.seed(seed)
        bootstrap_index = [random.randint(0, len(train_df)-1) for i in range(0,n_bootstrap)]
        return train_df.iloc[bootstrap_index]
    def random_forest_algorithm(self,train_df, n_trees, n_bootstrap):
        forest = []
        c = CART()
        random.seed(0)
        r_seed = [random.randint(0, len(train_df)-1) for i in range(0,n_trees)]
        #r_seed = random.sample(range(0, len(train_df)-1), n_trees)
        for i in range(n_trees):
            df_bootstrapped = self.bootstrapping(train_df, n_bootstrap, r_seed[i])
            forest.append(c.build_tree(np.array(df_bootstrapped)))
        return forest
    def random_forest_predictions(self,forest,test_df):
        df_predictions = {}
        c = CART()
        for i in range(len(forest)):
            column_name = "tree_{}".format(i)
            df_predictions[column_name] = c.decision_tree(tree=forest[i],test=test_df)
        random_forest_predictions = pd.DataFrame(df_predictions).mode(axis=1)[0]
        return random_forest_predictions
```

Lampiran 7. Kelas CART

```
class CART():
    def test_split(self,index, value, dataset):
        left, right = list(), list()
        for row in dataset: left.append(row) if row[index]<value else right.append(row)
        return left, right

    def gini_index(self, groups, classes):
        n_instances = float(sum([len(group) for group in groups]))
        gini = 0.0
        for group in groups:
            size = float(len(group))
            if size == 0:
                continue
            score = 0.0
            # score the group based on the score for each class
            for class_val in classes:
                p = [row[-1] for row in group].count(class_val) / size
                score += p**2
            # weight the group score by its relative size
            gini += (1.0 - score) * (size / n_instances)
        return gini

    def get_best_split(self,dataset):
        class_values = list(set(row[-1] for row in dataset))
        b_index, b_value, b_score, b_groups = 999, 999, 999, None
        for index in range(len(dataset[0])-1):
            for row in dataset:
                groups = self.test_split(index, row[index], dataset)
                gini = self.gini_index(groups, class_values)
                if gini < b_score:
                    b_index, b_value, b_score, b_groups = index, row[index], gini, groups
        return {'index':b_index, 'value':b_value, 'groups':b_groups}
    def to_terminal(self,group):
        outcomes = [row[-1] for row in group]
        return max(set(outcomes), key=outcomes.count)

    def split(self,node):
        #print("Def Split")
        left, right = node['groups']
        del(node['groups'])
        # check for a no split
        if not left or not right:
            node['left'] = node['right'] = self.to_terminal(left + right)
            return
```

```
        # process left child
        if len(left) <= 1:
            node['left'] = self.to_terminal(left)
        else:
            node['left'] = self.get_best_split(left)
            self.split(node['left'])
        # process right child
        if len(right) <= 1:
            node['right'] = self.to_terminal(right)
        else:
            node['right'] = self.get_best_split(right)
            self.split(node['right'])
    def build_tree(self,train):
        root = self.get_best_split(train)
        self.split(root)
        return root
    def predict(self, node, row):
        if row[node['index']] < node['value']:
            return self.predict(node['left'], row) if isinstance(node['left'], dict) else
node['left']
        else:
            return self.predict(node['right'], row) if isinstance(node['right'], dict) else
node['right']
    def decision_tree(self,tree, test):
        predictions = list()
        for row in test:  predictions.append(self.predict(tree, row))
        return(predictions)
```