LAMPIRAN

Analisis Sentimen

```
def sentimentNANDOperator(a, b):
    # Measure two value with NAND Operator
    if a == 0 or b == 0: return (a+b)
    return 1 if a+b > 0 else -1

def sentimentANDOperator(a, b):
    # Measure two value with AND Operator
    if a == 0 or b == 0: return (a+b)
    return 1 if a == b else -1

def singleRule(data):
    # Return sentiment degree of a word
    return posdf.loc[posdf["word"] == data,
'sentiment'].iloc[0]

def verbAdjectiveRule(idx, words, pos):
    try:
        # Find next pattern index
        idxAdj = pos[idx+1:].index("Adjektiva")+(idx+1)
        if idxAdj >= idx+1:
            adjsenti = posdf.loc[posdf["word"] ==
words[idxAdj], 'sentiment'].iloc[0]
            return idxAdj, adjsenti
        return False
    except ValueError:
        return False

def verbRule(idx, words, pos):
    verbsenti = posdf.loc[posdf["word"] == words[idx],
'sentiment'].iloc[0]
    try:
        isverbplusadjective, adjsenti =
verbAdjectiveRule(idx, words, pos)
        return isverbplusadjective,
sentimentNANDOperator(verbsenti, adjsenti)
    except TypeError:
        return idx, verbsenti

def prepositionAdjectiveRule(idx, preposenti,
datalist):
      # Measure Prepo + Adjective sentiment degree
    adjsenti = posdf.loc[posdf["word"] ==
datalist['words'][idx+1], 'sentiment'].iloc[0]
```

```
    return sentimentANDOperator(preposenti,adjsenti)

def prepositionVerbRule(idx, preposenti, datalist):
      # Measure Prepo + Verb sentiment degree
    isanyadjidx, verbsenti =
verbRule(idx+1,datalist['words'], datalist['pos'])
    if isanyadjidx: # Check if there's any adjective
after verb
        return isanyadjidx,
sentimentANDOperator(preposenti,verbsenti)
    return idx,
sentimentNANDOperator(preposenti,verbsenti)

def prepositionRule(idx, words, pos):
    datalist = {'words': words, 'pos': pos}
    preposenti =  posdf.loc[posdf["word"] ==
words[idx], 'sentiment'].iloc[0]
    try:
        if pos[idx+1] == "Adjektiva":
            return [idx+1],
prepositionAdjectiveRule(idx, preposenti, datalist)
        elif pos[idx+1] == "Verba":
            idxAdj, sentiment =
prepositionVerbRule(idx, preposenti, datalist)
            return [idx+1, idxAdj], sentiment
        return idx, 0
    except IndexError:
        return idx, 0

def adverbiaAdjectiveRule(idx, adverbsenti, datalist):
      # Measure advb + Adjective sentiment degree
    adjsenti = posdf.loc[posdf["word"] ==
datalist['words'][idx+1], 'sentiment'].iloc[0]
    return sentimentNANDOperator(adverbsenti,adjsenti)

def adverbiaVerbRule(idx, adverbsenti, datalist):
      # Measure advb + Verb sentiment degree
    isanyadjidx, verbsenti =
verbRule(idx+1,datalist['words'], datalist['pos'])
    if isanyadjidx: # Check if there's any adjective
after adverbia
        return isanyadjidx,
sentimentNANDOperator(adverbsenti,verbsenti)
    return idx,
sentimentNANDOperator(adverbsenti,verbsenti)

def adverbiaRule(idx, words, pos):
    datalist = {'words': words, 'pos': pos}
    adverbsenti =  posdf.loc[posdf["word"] ==
words[idx], 'sentiment'].iloc[0]
    try:
        if pos[idx+1] == "Adjektiva":
            return [idx+1], adverbiaAdjectiveRule(idx,
adverbsenti, datalist)
        elif pos[idx+1] == "Verba":
            idxAdj, sentiment = adverbiaVerbRule(idx,
adverbsenti, datalist)
            return [idx+1, idxAdj], sentiment
        return idx, 0
    except IndexError:
        return idx, 0

def getWordSentimentValue(idx, word, pos):
    if pos[idx] == "Verba":
        return verbRule(idx, word, pos)
    elif pos[idx] == "Preposisi":
        return prepositionRule(idx, word, pos)
    elif pos[idx] == "Adjektiva":
        return idx, singleRule(word[idx])
    elif pos[idx] == "Adverbia":
        return adverbiaRule(idx, word, pos)
    else:
        return idx, 0

def terimakasihPosition(sentence):
    try:
        try:
            if sentence.split(" ").index("terimakasih")
== 0:
                return 1
        except ValueError:
            idxterima = sentence.split("
").index("terima")
            idxkasih = sentence.split("
").index("kasih")
            if idxterima+1 == idxkasih and idxterima ==
0:
                return 1
    except:
        return 0

def anyFraseDuaKata(sentence):
    # Measure and check frase sentiment degree
    frasadf = pd.read_excel(filename, 'Frasa')
    frase = frasadf['word'].values.tolist()
    preponegatif = ["tidak", "belum", "anti", "bukan"]
    found = []
    try:
        sentence = sentence.split(" ")
        for idx, word in enumerate(sentence):
            twogram = sentence[idx]+' '+sentence[idx+1]
            if twogram in frase:
                sentimentfrase =
frasadf.loc[frasadf["word"] == twogram,
'sentiment'].iloc[0]
                if sentence[idx-1] in preponegatif:
                    sentimentfrase =
sentimentANDOperator(sentimentfrase, -1)
                found.append([twogram, sentimentfrase])
    except IndexError:
        pass
    return found

def checkFrase(sentence):
    # Measure and check frase sentiment degree
    check = anyFraseDuaKata(sentence)
    sentiment = 0
    if check != []:
        for item in check:
            sentence = sentence.replace(str(item[0]),
"")
            sentiment += item[1]
        return sentiment, sentence
    return 0

def normalizeSentimentVal(val):
    # Normalizing sentiment degree into three label
    # Positive (1), Negative (-1), Neutral (0)
    if val == 0: return 0
    elif val > 0: return 1
    else: return -1

def getSentiment(sentence, out):
    # Main method for get sentence sentiment degree
    totalsentiment = 0
    sentence_pre = preprocessing(sentence)
    sentencebreak = dotAndCommaBreak(sentence_pre)
    of = []
    ost = []
    sanitisinglist = ["se","begitu"]
    for istc, sentenceb in enumerate(sentencebreak):
        skipIndex = []
        sentimentval = 0
        if terimakasihPosition(sentenceb) == 1 and istc
== 0:
            sentenceb.replace("terima", "")
            sentenceb.replace("kasih", "")
            sentimentval += 1
        try:
            sentimentfrase, sentenceb =
checkFrase(sentenceb)
            sentimentval += sentimentfrase
        except TypeError:
            pass
        word = list(filter(None, sentenceb.split(" ")))
        #sanitising
        word = [item for item in word if item not in
sanitisinglist]
        word = filtering(word)
        of.append(word)
        word = stemmingWord(word)
        ost.append(word)
        pos = convertSentence(' '.join(word))
        kalimat = []
        for idx,tag in enumerate(pos):
            if idx not in skipIndex and tag in pattern:
                issentiment, sentiment =
getWordSentimentValue(idx,word,pos)
                if out == 2 :
                    ww = [word[idx], pos[idx],
sentiment]
                    kalimat.append(' '.join(str(v) for
v in ww))
                if issentiment is not None:
                    sentimentval += sentiment
                    try:
                        skipIndex = skipIndex +
issentiment
                    except TypeError:
                        skipIndex.append(issentiment)
                        continue
            else:
                continue
        totalsentiment += sentimentval
    if out == 1:
        re = [sentence_pre,', '.join(str(v) for v in
of), ', '.join(str(v) for v in ost),
normalizeSentimentVal(totalsentiment)]
        return re
    elif out == 2:
        re = [sentence_pre, ', '.join(str(v) for v in
of), ', '.join(str(v) for v in ost), kalimat,
normalizeSentimentVal(totalsentiment)]
        return re
    else :
        return normalizeSentimentVal(totalsentiment)
```