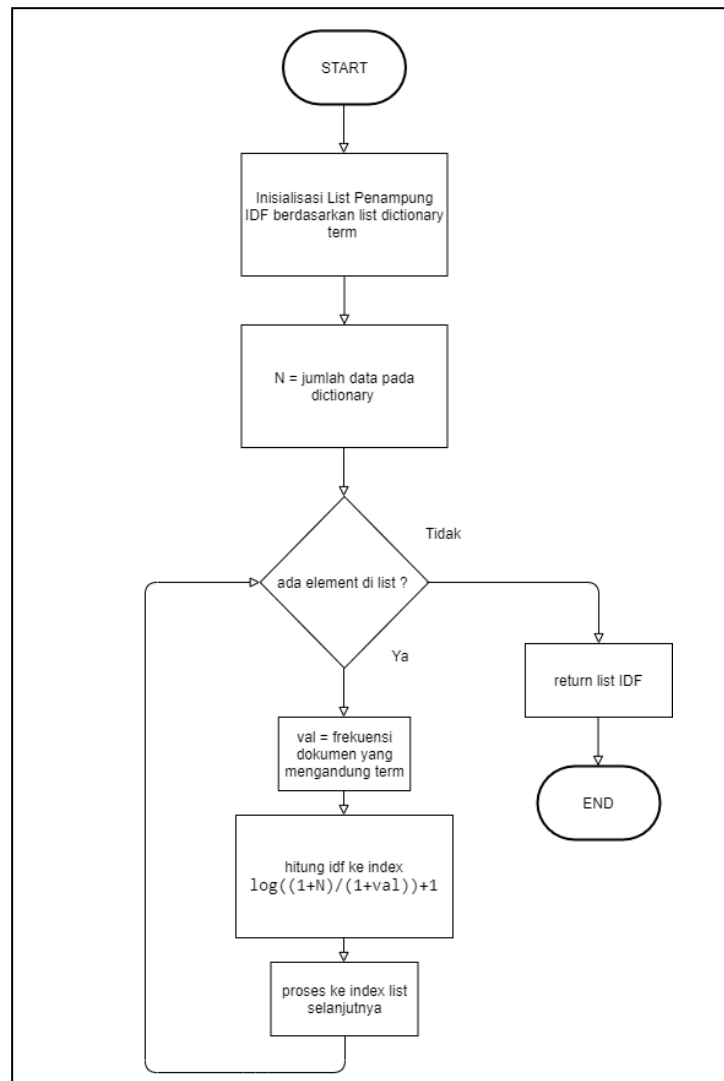


Gambar 3.3 Flowchart tahap proses vektorisasi TF-IDF

Setelah tahap *preprocessing*, akan dilakukan vektorisasi data menggunakan metode TF-IDF. Vektorisasi menggunakan TF-IDF mempunyai beberapa tahap yang dapat dilihat pada gambar 3.3 yang merupakan penjabaran untuk proses vektorisasi menggunakan TF-IDF. Langkah pertama yang dilakukan adalah mengubah data sentimen pengguna pada masing masing data *train* dan data *test* menjadi bentuk n-grams. Dalam penelitian ini, bentuk n-grams yang digunakan adalah bigrams dan trigrams. Setelah data diubah dalam bentuk n-grams, langkah selanjutnya adalah menginisialisasi data *list dictionary* setiap data *train*. *List* tersebut memiliki indeks berupa *term* untuk setiap baris data yang didapatkan dari proses pengubahan bentuk bigrams dan trigrams data *train*. Lalu tahap selanjutnya adalah menghitung frekuensi kemunculan tiap *term* untuk setiap baris data. Langkah ini diperlukan untuk menghitung nilai TF (*term frequency*).



Gambar 3.4 Flowchart tahap proses perhitungan nilai IDF

Setelah membuat *list dictionary* untuk masing-masing data *train*, langkah selanjutnya adalah menghitung nilai IDF (*Inverse Document Frequency*) pada setiap *term* untuk masing-masing data *train*. Gambar 3.4 merupakan tahapan untuk menghitung nilai IDF setiap *term* pada semua data *train*. Langkah pertama yang dilakukan adalah inisialisasi *list* yang akan digunakan untuk menampung nilai IDF. Lalu dilakukan *looping* untuk setiap *term* yang terdapat pada *list dictionary*.

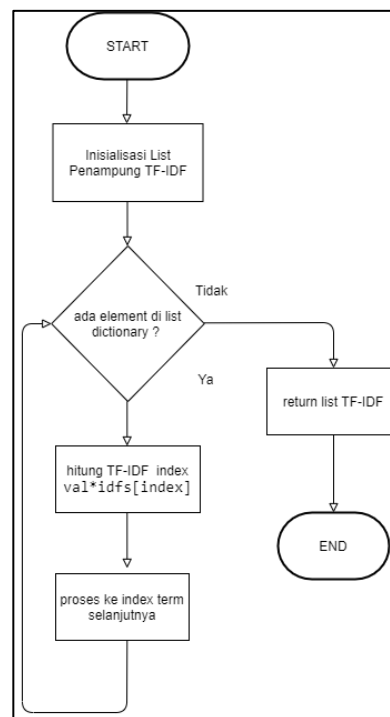
Didalam *loop* tersebut, dilakukan perhitungan nilai IDF pada *term* yang sedang menjadi *index looping* tersebut. Rumus IDF yang digunakan adalah sebagai berikut

$$IDF = \log ((1 + N) / (1 + Val )) + 1 \quad (3.1)$$

Keterangan :

1.  $N$  = Banyaknya baris data pada data *train*
2.  $Val$  = banyaknya data yang mempunyai *term* ke index

Rumus yang digunakan diatas terdapat sedikit perbedaan pada rumus IDF yaitu penambahan angka '1' pada pembilang dan penyebut, yang berguna jika menghasilkan nilai yang memicu kesalahan seperti *division by zero*.



Gambar 3.5 Flowchart tahap proses perhitungan nilai TF-IDF

Setelah menghitung dan membuat *list* nilai IDF, tahap selanjutnya adalah menghitung nilai TF-IDF untuk semua *term* pada setiap baris data *train*. Gambar 3.5 merupakan penjabaran untuk proses perhitungan nilai TF-IDF. Langkah pertama yang dilakukan adalah inisialisasi *list* yang akan digunakan untuk menampung nilai TF-IDF. Lalu dilakukan *looping* untuk setiap *term* yang terdapat satu baris data pada data train. Didalam *loop* tersebut, dilakukan perhitungan nilai TF-IDF non normalisasi pada *term* yang sedang menjadi index *looping* tersebut. Rumus TF-IDF yang digunakan adalah sebagai berikut

$$TF - IDF = Val * IDF \quad (3.2)$$

Keterangan :

1. Val = Frekuensi kemunculan *term* pada suatu data

Setelah menghitung nilai TF-IDF non normalisasi maka tahap selanjutnya adalah menormalisasi nilai TF-IDF didalam *list* tersebut. Berikut rumus yang digunakan untuk normalisasi nilai TF-IDF.

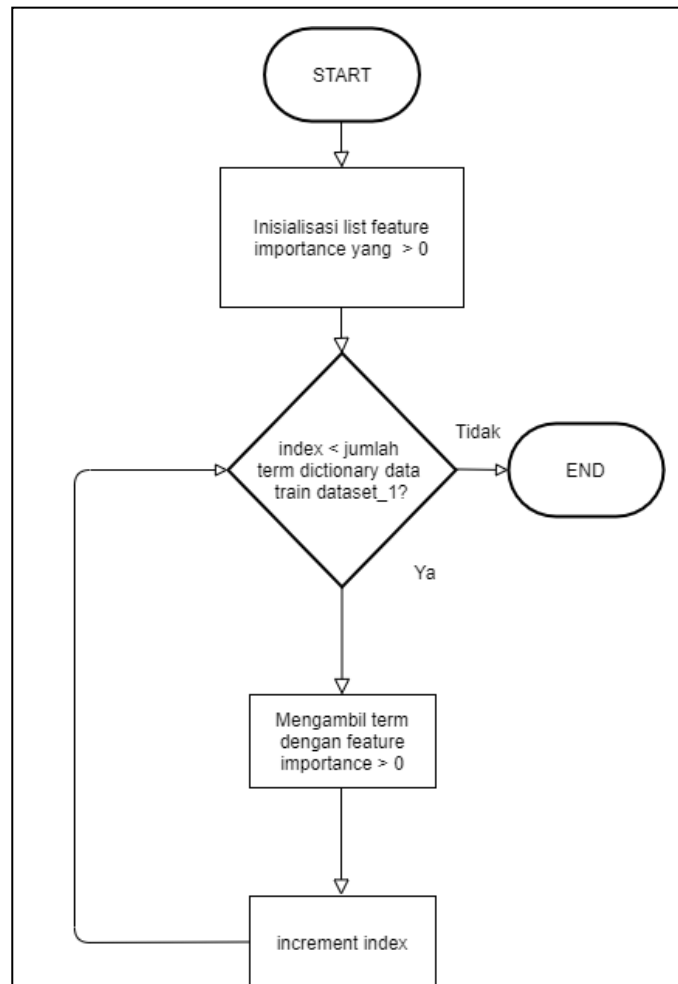
$$V_{norm} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \dots + v_n^2}} \quad (3.3)$$

Keterangan :

1. v = nilai TF-IDF non normalisasi

Setelah tahap perhitungan TF-IDF selesai, lalu dilakukan tahap selanjutnya yaitu membuat model Random Forest dengan menggunakan data train yang sudah diproses dan divektorisasi sebelumnya. Dalam penelitian ini model Random Forest dibuat dengan menggunakan *library* Random Forest Classifier dari scikit-learn,

karena *library* ini mempunyai fitur untuk mengembalikan nilai berupa *feature importance* tiap *term* yang nantinya akan digunakan untuk proses *transfer learning*.

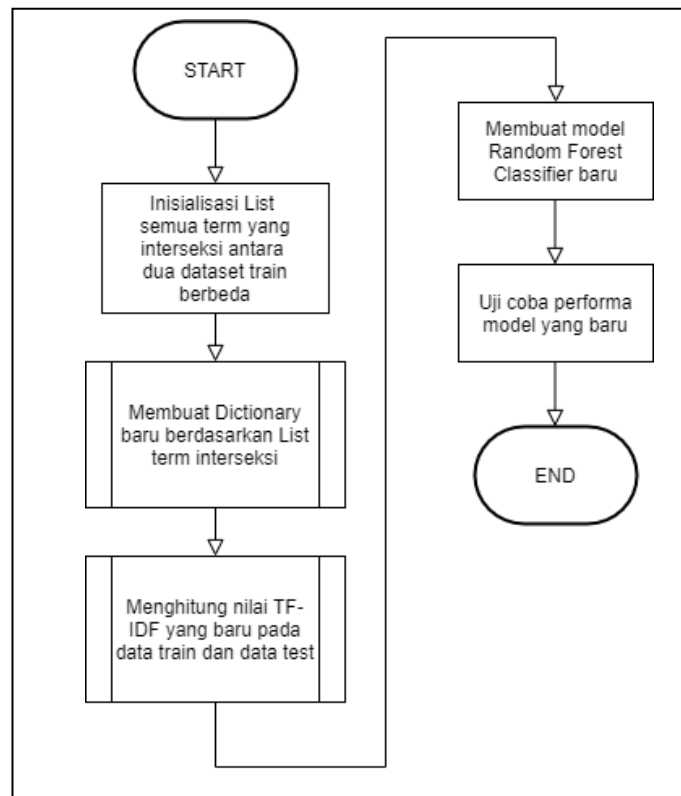


Gambar 3.6 Flowchart tahap proses pembuatan list *feature importance*

Lalu setelah model selesai dibuat, tahap selanjutnya adalah melakukan uji coba pada model tersebut. Uji coba dilakukan dengan data *test* yang sudah divektorisasi sebelumnya dan berikut dengan label-nya. Setelah uji performa model dilakukan maka akan didapatkan skor performa model seperti akurasi *train*, akurasi *test*, nilai *precision*, *recall*, dan *f1-score*. Serta juga dapat menghitung berapa lama model memerlukan waktu untuk *training* dan *testing* data. Dalam penelitian ini juga

dilakukan tahap *transfer learning* untuk mengetahui apakah dapat mempengaruhi performa dari model Random Forest yang telah dibangun.

*Transfer learning* pertama yang dilakukan adalah dengan menggunakan informasi nilai dari *feature importance* untuk setiap *term* pada model Random Forest yang di *training* pada *dataset* pertama dan kemudian diterapkan pada *dataset* selanjutnya. Tahap pertama yang dilakukan adalah membuat *list* dari *term-term* yang memiliki nilai *feature importance* lebih dari 0 pada model Random Forest *dataset* pertama. Gambar 3.6 merupakan penjabaran langkah-langkah untuk membuat *list* tersebut. Langkah pertama yang dilakukan adalah inisialisasi *list* yang akan digunakan, lalu akan dilakukan *looping* untuk setiap *term* yang terdapat pada *dictionary dataset* pertama. Didalam *looping* tersebut akan dilakukan pengecekan apakah *term* tersebut mempunyai *feature importance* yang lebih besar dari 0, jika lebih besar maka *term* tersebut akan ditambahkan kedalam *list*. Setelah *looping* selesai mengakses semua *term* yang terdapat pada *dictionary* maka *list* akan di *return*.

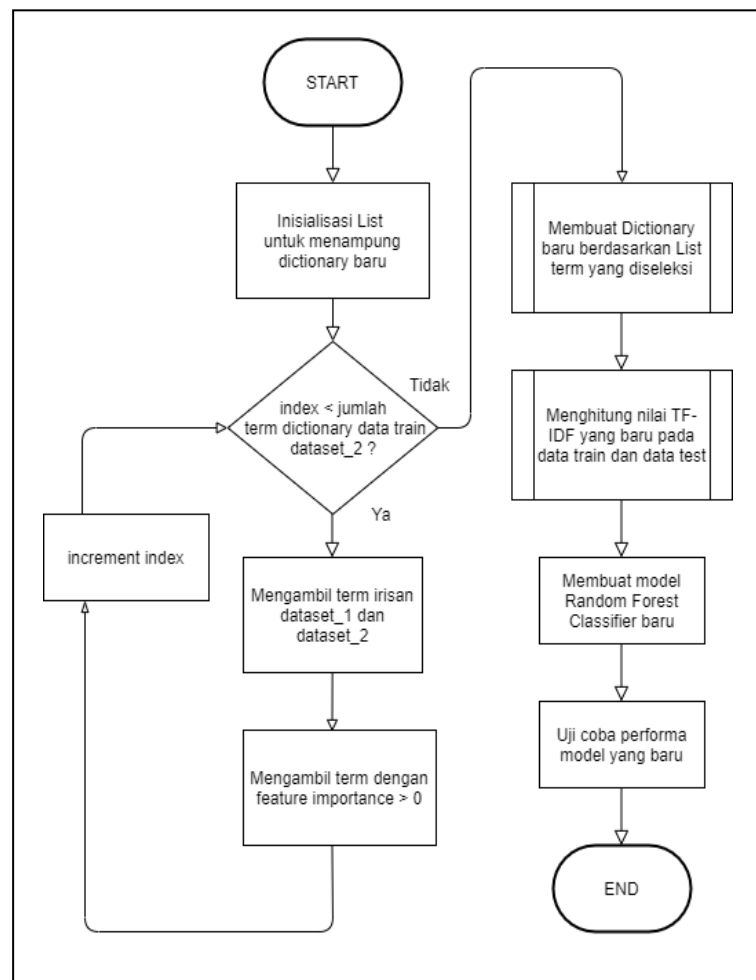


Gambar 3.7 Flowchart implementasi *transfer learning* pada interseksi *dataset*

Pada proses *transfer learning feature importance* akan dilakukan dua skenario uji coba yaitu skenario pertama dengan menggunakan data *dictionary* yang dibuat berdasarkan *term-term* yang saling berinterseksi saja antara *dataset* pertama dan *dataset* kedua, dan skenario kedua yaitu dengan menyeleksi data *term* pada *dataset* kedua berdasarkan informasi *feature importance* yang didapatkan dari model Random Forest *dataset* pertama. Gambar 3.7 merupakan *flowchart* penjabaran langkah yang dilakukan dalam implementasi *transfer learning feature importance* pada skenario pertama.

Langkah pertama yang dilakukan adalah inisialisasi *list* yang berisi semua *term* yang berinterseksi antara *dataset* pertama yang merupakan *dataset* yang telah

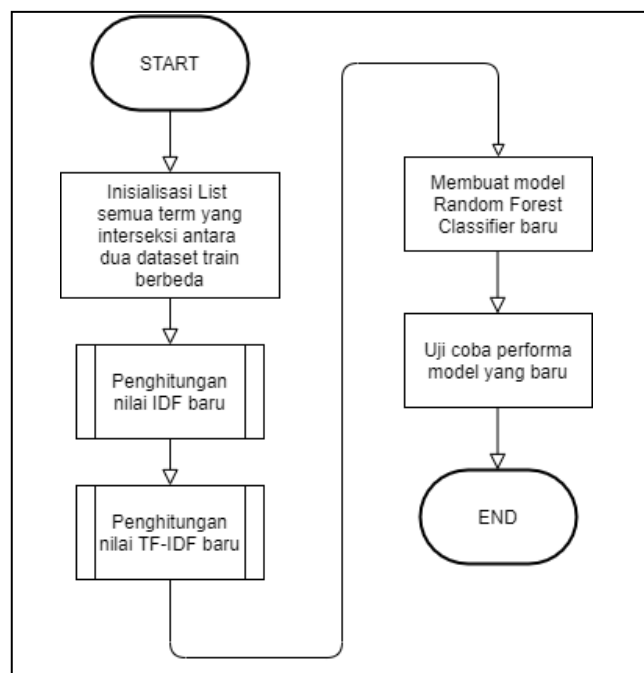
di *training* menjadi *pretrained model* dan *dataset* kedua. Lalu langkah selanjutnya adalah membuat *dictionary term* berdasarkan *term* dari *list* tersebut dan mengisi nilai dari tiap indeks *term* dengan frekuensi kemunculan *term* tersebut pada setiap baris data. Selanjutnya dilakukan perhitungan nilai TF-IDF pada *dictionary* tersebut sesuai langkah pada Gambar 3.3 yang telah dideskripsikan sebelumnya. Setelah tahap perhitungan TF-IDF selesai, lalu dilakukan tahap selanjutnya yaitu membuat model Random Forest dan melakukan pengujian kembali model Random Forest baru tersebut.



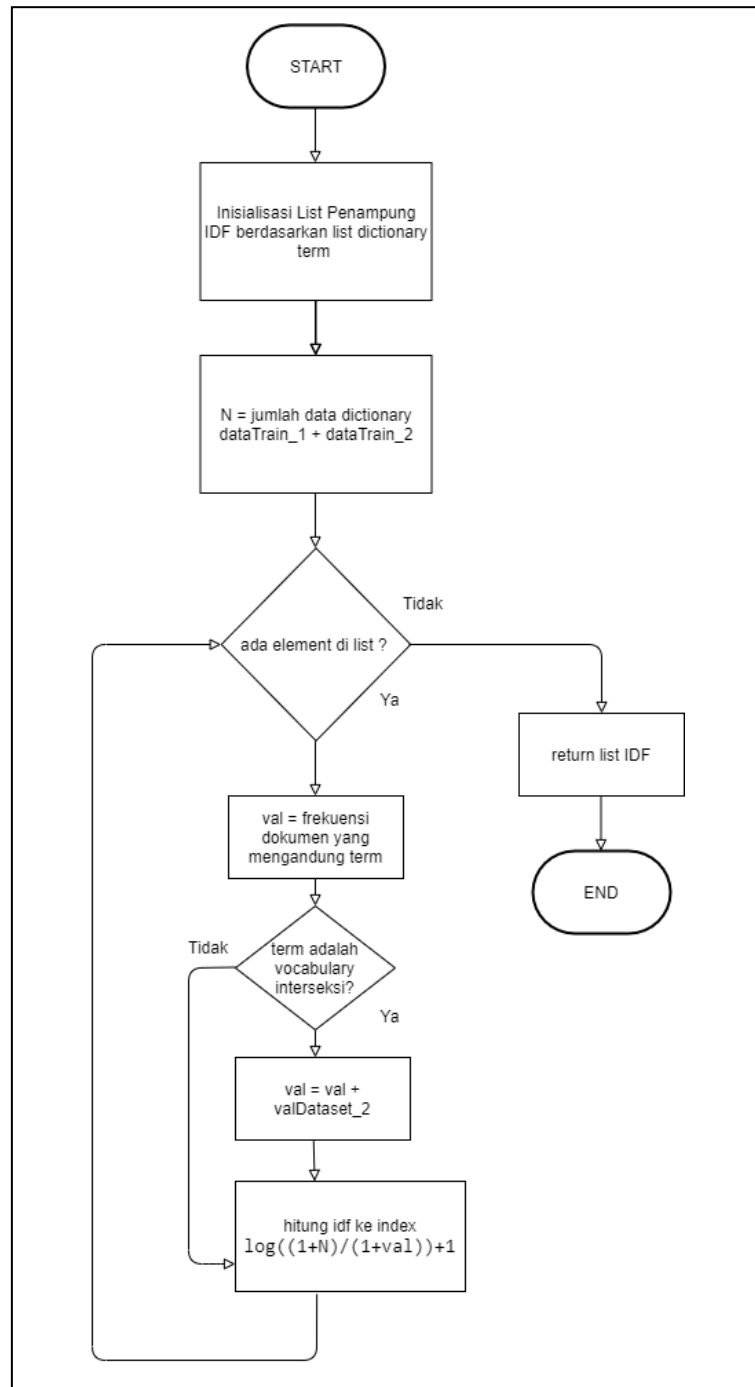
Gambar 3.8 Flowchart implementasi *transfer learning* dengan seleksi dataset



Gambar 3.8 merupakan langkah untuk implementasi *transfer learning* pada *feature importance* skenario kedua yaitu dengan seleksi *term* atau *feature* pada *dataset* kedua. Langkah pertama yang dilakukan adalah inisialisasi *dictionary* yang akan menampung *term-term* baru hasil seleksi dari *dataset* kedua. Lalu akan dilakukan *looping* untuk setiap *term* terdapat pada *dictionary dataset* kedua. Pada *looping* tersebut dilakukan pengambilan data yang berinterseksi antara *dataset\_1* dan *dataset\_2* dan memiliki *feature importance* yang lebih besar dari nol . Setelah *looping* selesai mengakses semua *term* yang terdapat pada *dataset* kedua, maka proses dilanjutkan pada tahap membuat *dictionary term* yang baru berdasarkan *term* yang sudah diseleksi dan mengisi nilai dari tiap indeks *term* dengan frekuensi kemunculan *term* tersebut pada setiap baris data. Sama seperti skenario pertama, proses selanjutnya adalah menghitung nilai TF-IDF dan membangun model Random Forest yang baru berdasarkan nilai TF-IDF yang baru dan melakukan uji coba performa.



Gambar 3.9 Flowchart implementasi *transfer learning* dengan nilai IDF



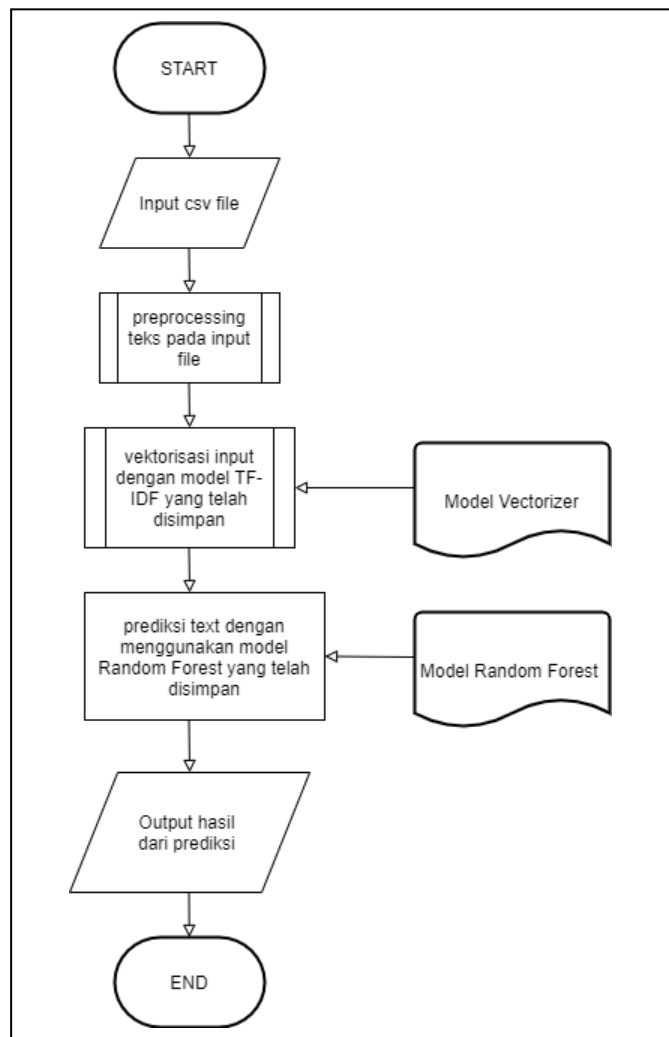
Gambar 3.10 Flowchart perhitungan nilai IDF *transfer learning*

Selain *transfer learning* Pada penelitian ini juga melakukan *transfer learning* nilai IDF pada *dataset* pertama dan menerapkan pada *dataset* kedua. Gambar 3.9 adalah *flowchart* yang menjabarkan langkah-langkah untuk menerapkan *transfer learning* IDF. Langkah pertama yang dilakukan adalah

inisialisasi *list* untuk menampung *term-term* yang saling interseksi antara *dataset* pertama dan *dataset* kedua. Lalu langkah selanjutnya adalah melakukan perhitungan terhadap nilai IDF baru pada *dictionary dataset* kedua dengan informasi pada nilai IDF pada *dataset* pertama.

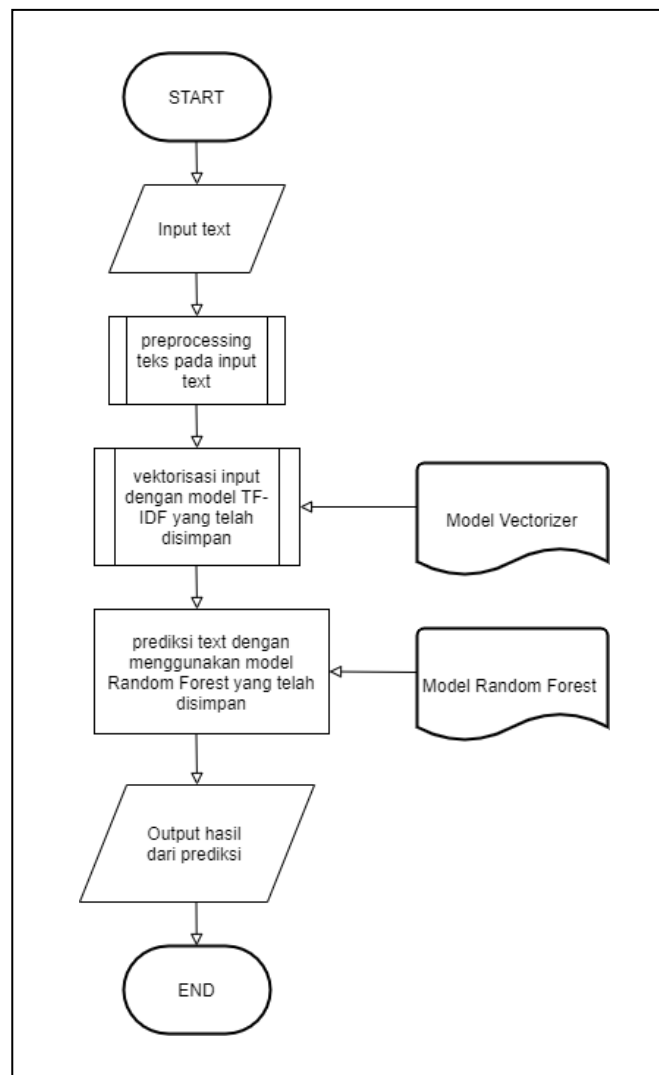
Gambar 3.10 merupakan *flowchart* yang menjabarkan langkah untuk menghitung nilai IDF yang baru pada *dictionary dataset* kedua. Langkah pertama yang dilakukan adalah me inisialisasi *dictionary* untuk menampung nilai IDF berdasarkan *term* dari *dictionary dataset* kedua. Selanjutnya akan dilakukan *looping* untuk setiap *term* yang terdapat pada *dictionary* yang telah diinisialisasi sebelumnya. Didalam *looping* tersebut terdapat kondisi apakah *term* yang sedang diakses terdapat pada masing-masing *dataset* pertama dan *dataset* kedua. Jika kondisi tersebut bernilai *true*, maka nilai variabel *val* yang semula hanya menampung nilai frekuensi *term* indeks yang sedang diakses pada *dataset* kedua menjadi ditambahkan dengan *frekuensi term* indeks pada *dataset* pertama. Langkah selanjutnya adalah menghitung nilai IDF pada *term* yang sedang diakses dengan menggunakan rumus yang sama dengan perhitungan nilai IDF sebelumnya. Jika *looping* telah selesai mengakses semua indeks *term* maka *list dictionary* IDF akan di *return*. Langkah selanjutnya adalah menghitung nilai TF-IDF dengan *list* nilai IDF yang baru dan membangun model Random Forest yang baru berdasarkan nilai TF-IDF yang baru dan melakukan uji coba performa.

Gambar 3.11 merupakan *flowchart* untuk proses prediksi *input* berupa data sentimen berbentuk file *csv* yang diunggah ke dalam aplikasi *web*. Ketika pengguna telah selesai *submit file csv*, maka akan dilakukan tahap preprocessing dengan langkah-langkah yang sama dengan sebelumnya. Selanjutnya dilakukan vektorisasi pada *input* dengan menggunakan model vectorizer TFIDF yang telah disimpan sebelumnya. Lalu dilakukan prediksi terhadap hasil vektorisasi tersebut dengan model Random Forest yang telah disimpan. Lalu hasil dari prediksi tersebut ditampilkan dengan menggunakan tabel.



Gambar 3.11 Flowchart Prediksi Input File pada aplikasi web

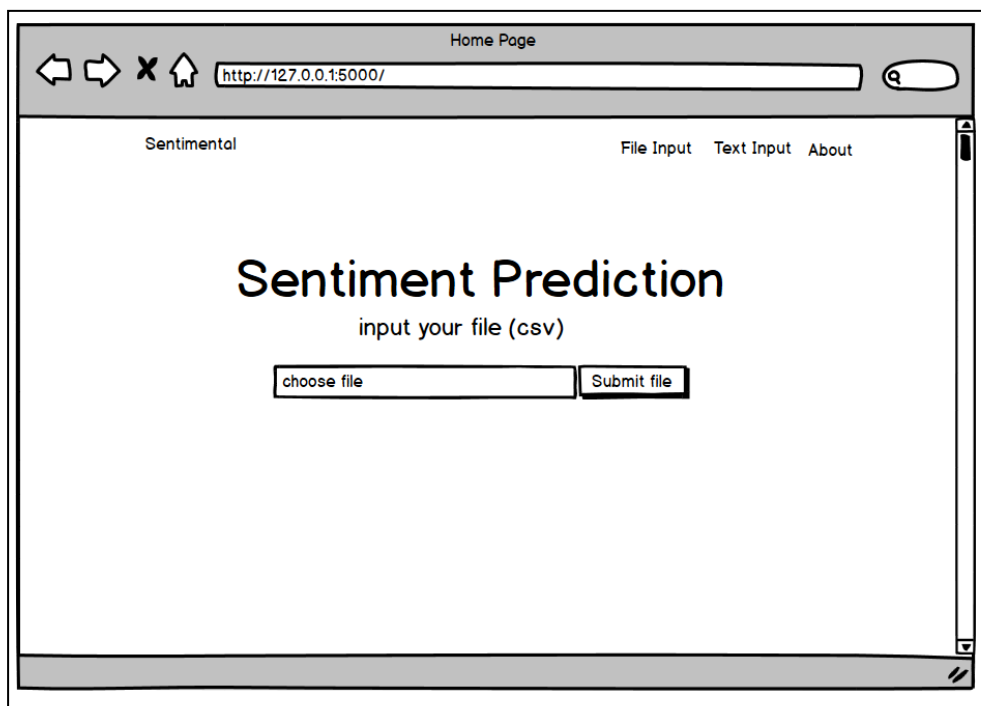
Gambar 3.12 merupakan *flowchart* untuk proses prediksi *input* berupa data sentimen berbentuk teks langsung. Ketika pengguna telah selesai *input* teks, maka proses yang sama seperti proses klasifikasi pada Gambar 3.11 akan dilakukan. Hasil prediksi akan ditampilkan pada halaman *web* apakah teks tersebut memiliki sentimen positif atau negatif.



Gambar 3.12 Flowchart Prediksi Input Teks pada aplikasi web

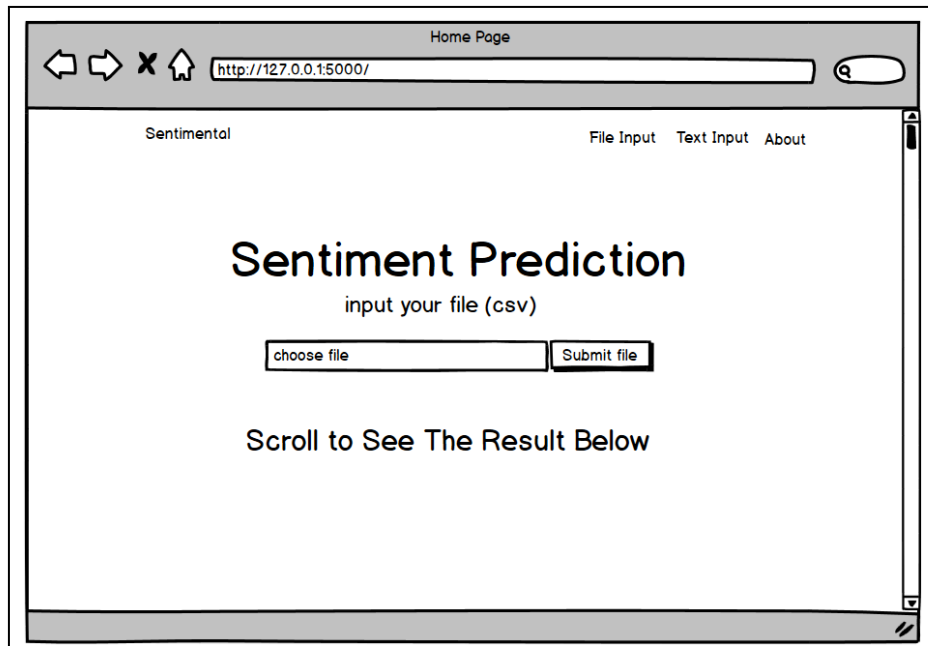
### 3.2.2 Rancangan Antarmuka

Rancangan antarmuka *website* terbagi menjadi tiga halaman, yaitu halaman *upload File*, *text input*, dan *about*. Pada Gambar 3.13 terdapat rancangan antarmuka untuk halaman *upload file*. Di halaman ini pengguna dapat *upload* dokumen dengan format *csv* yang mengandung data untuk diprediksi.



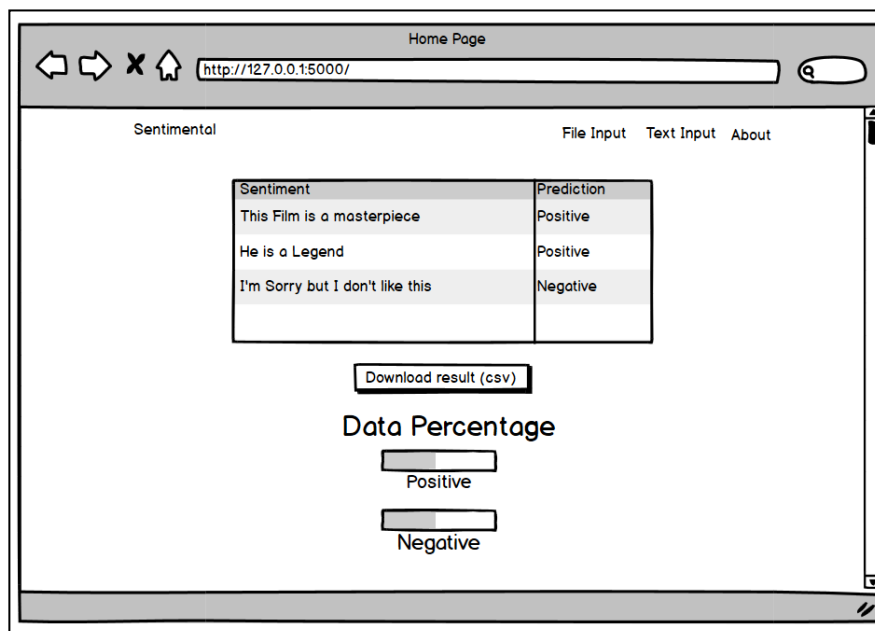
Gambar 3.13 Halaman *upload file*

Pada Gambar 3.14 terdapat rancangan antarmuka pada halaman *upload file* jika pengguna telah berhasil *upload file csv*. Berbeda dengan antarmuka sebelumnya, pada antarmuka ini, terdapat informasi untuk pengguna agar *scroll* untuk melihat hasil dari prediksi *upload file* tersebut.



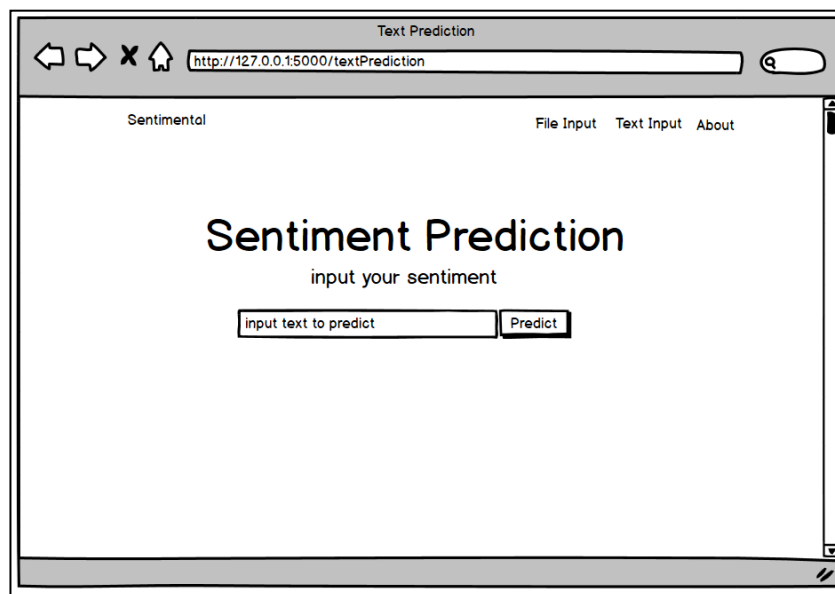
Gambar 3.14 Halaman *upload file* jika telah berhasil upload

Gambar 3.15 adalah rancangan antarmuka pada halaman *upload file* bagian bawah. Pada bagian ini, pengguna dapat melihat hasil dari prediksi data file yang telah di *upload* dan dapat *download* hasil dari prediksi tersebut. Pada bagian ini pengguna juga dapat melihat persentase hasil prediksi yang telah dimasukkan.

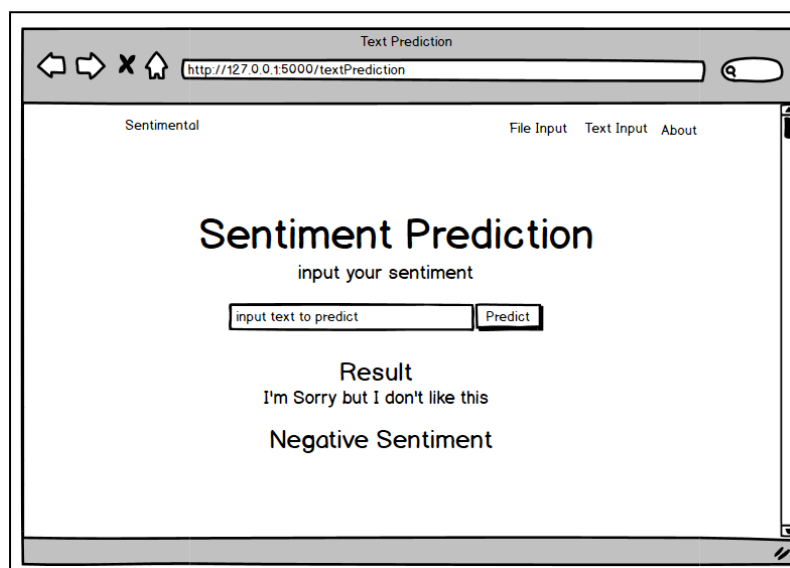


Gambar 3.15 Halaman *upload file* untuk menampilkan hasil prediksi

Pada Gambar 3.16 terdapat rancangan antarmuka untuk halaman *text input*. Di halaman ini pengguna dapat memasukkan *input* sentimen yang kemudian bisa diprediksi dan ditampilkan hasilnya secara langsung. Gambar 3.17 merupakan tampilan halaman *text input* jika pengguna selesai memasukan input. Pada tampilan tersebut terdapat sentiment yang pengguna *input* berikut dengan hasil prediksinya.



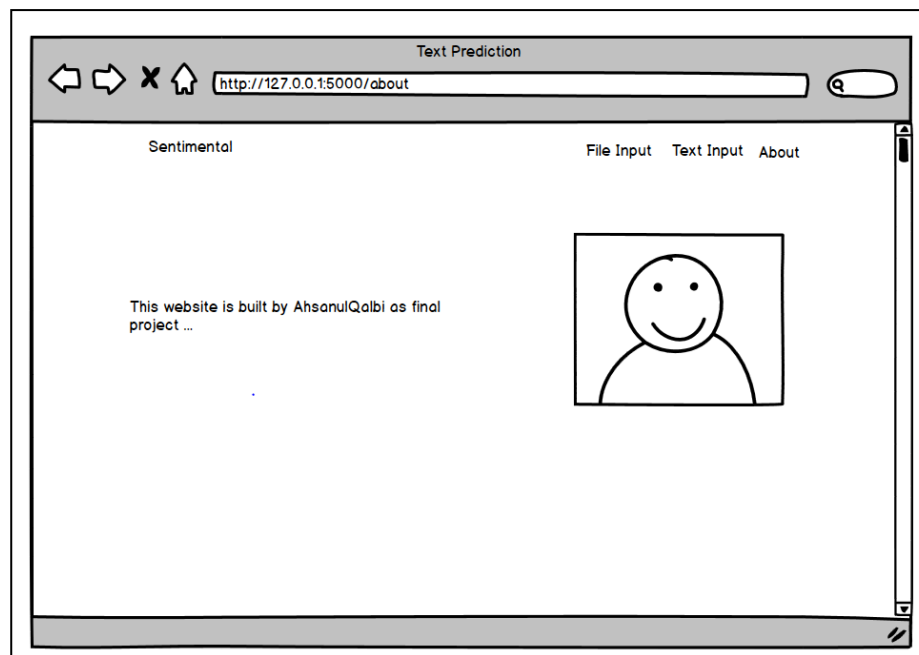
Gambar 3.16 Halaman *input text*



Gambar 3.17 Halaman *input text* dengan hasil input



Pada Gambar 3.18 terdapat rancangan antarmuka untuk halaman *about me*. Di halaman ini terdapat informasi mengenai *website* berupa deskripsi dan alasan mengapa *website* dibuat. Pada halaman ini juga terdapat profil peneliti.



Gambar 3.18 Halaman *about me*

## **BAB IV**

### **IMPLEMENTASI DAN ANALISIS**

#### **4.1 Spesifikasi Sistem**

Pada pelaksanaan penelitian ini digunakan beberapa *tool* atau alat untuk mendukung pelaksanaan penelitian baik itu perangkat keras (*hardware*) maupun perangkat lunak (*software*). Berikut merupakan komponen perangkat keras yang digunakan selama pengerjaan penelitian :

1. Laptop : MSI GL63 8SD
2. Processor : Intel Core i7-8750H CPU
3. Graphic Processing Unit : Nvidia GeForce GTX 1660 Ti
4. Memory : 16 GB RAM

Adapun *software* yang digunakan selama penelitian antara lain :

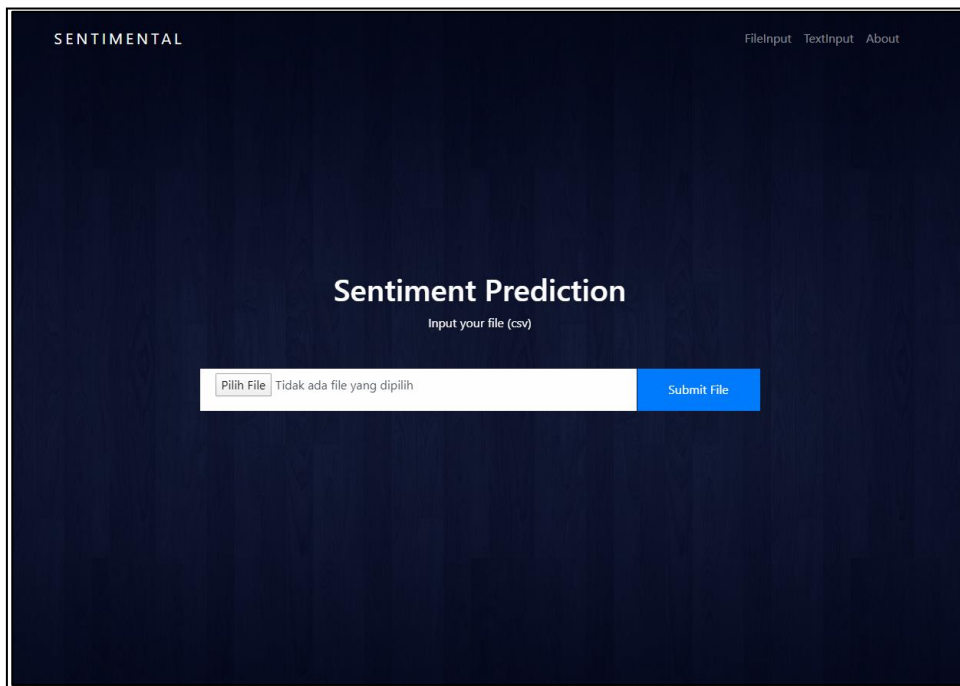
1. Python 3.7.4
2. Anaconda Prompt
3. Jupyter Notebook
4. Visual Studio Code
5. Framework Flask

## 4.2 Implementasi

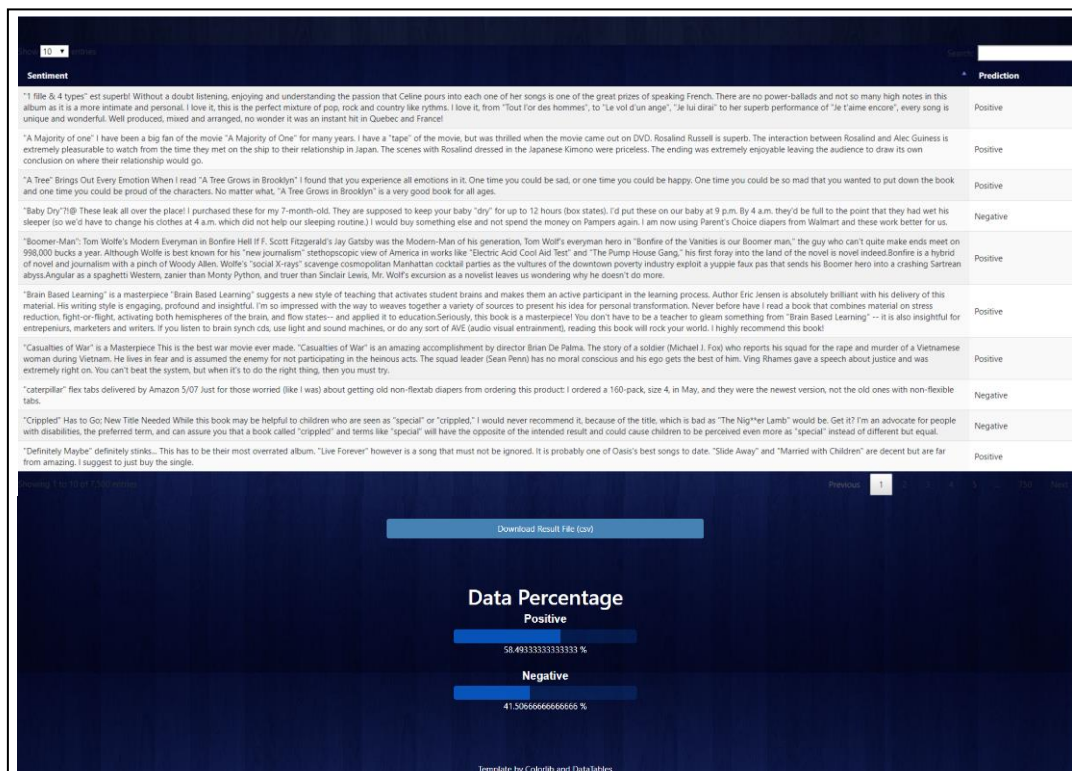
Dalam proses implementasi penelitian, aplikasi *web* dibangun dengan menggunakan bahasa pemrograman Python dengan bantuan *framework* Flask. Berikut adalah tampilan aplikasi dan potongan kode hasil implementasi aplikasi.

### 4.2.1 Implementasi Aplikasi Web

Gambar 4.1 merupakan halaman utama aplikasi *web* yaitu halaman *file upload*. Pada halaman utama ini memiliki *bar* navigasi berupa nama aplikasi, tombol navigasi halaman utama (*file upload*), tombol navigasi halaman *text upload* dan tombol navigasi halaman *about*. Ketiga tombol navigasi tersebut berfungsi untuk berpindah ke halaman *web* lainnya, *bar* navigasi tersebut juga terdapat di setiap halaman lainnya. Pada halaman utama ini juga memiliki *form* yang berguna agar pengguna dapat *upload file csv* yang berisi data sentiment yang ingin diprediksi. Gambar 4.2 merupakan bagian bawah halaman utama aplikasi *web* ketika pengguna telah selesai *submit file csv*. Pada bagian ini terdapat hasil dari prediksi sentimen yang telah di-*upload* oleh pengguna. Hasil prediksi tersebut ditampilkan dengan menggunakan tabel yang terdiri atas kolom sentimen dan hasil prediksi. Tabel tersebut dibangun menggunakan *template* dari DataTables. Selanjutnya terdapat tombol *download result file*, yang berguna agar pengguna dapat mengunduh hasil dari prediksi sentimen dalam bentuk *file csv*. Selanjutnya juga terdapat tampilan persentase label data hasil dari prediksi.

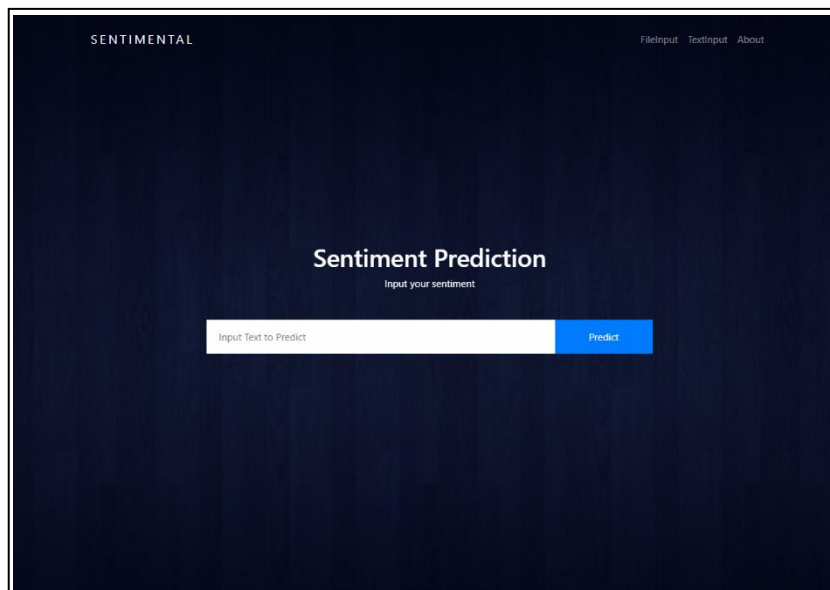


Gambar 4.1 Halaman utama aplikasi *web*

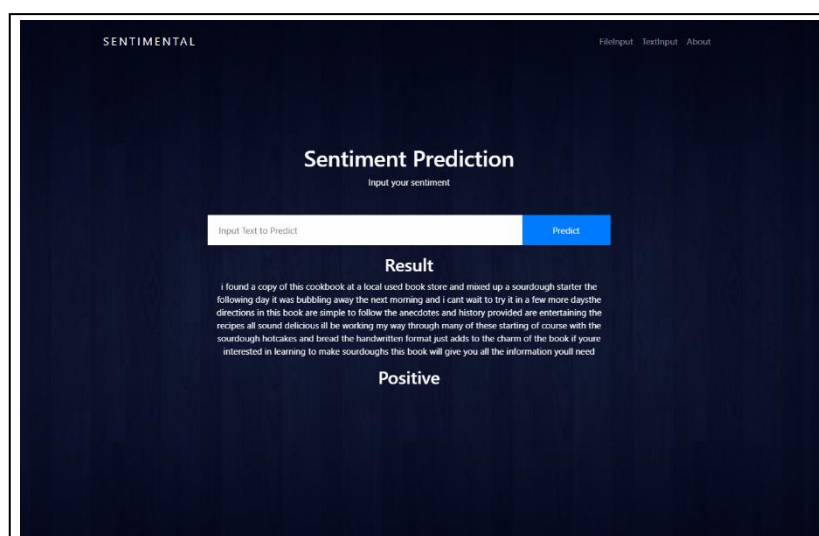


Gambar 4.2 Halaman utama web hasil prediksi file

Gambar 4.3 merupakan tampilan halaman input teks pada aplikasi *web*. Pada halaman ini terdapat *form* berupa *input* teks. *Form* tersebut berguna untuk pengguna dapat memasukan sentimen secara langsung dengan *input* teks. Jika pengguna telah melakukan *submit* pada *form* teks, maka akan ditampilkan hasil prediksi seperti yang digambarkan oleh Gambar 4.4.

The screenshot shows a web application interface with a dark blue background. At the top left, the word "SENTIMENTAL" is displayed in white. At the top right, there are links for "FileInput", "TextInput", and "About". In the center, the title "Sentiment Prediction" is shown in white, with the subtitle "Input your sentiment" below it. A white text input field with the placeholder "Input Text to Predict" is positioned above a blue button labeled "Predict".

Gambar 4.3 Halaman input teks

The screenshot shows the same web application interface as in Gambar 4.3, but with the prediction result displayed. Below the "Predict" button, the word "Result" is shown in white. Underneath, a paragraph of text is displayed in white, followed by the word "Positive" in white. The text in the paragraph reads: "I found a copy of this cookbook at a local used book store and mixed up a sourdough starter the following day it was bubbling away the next morning and i cant wait to try it in a few more days the directions in this book are simple to follow the anecdotes and history provided are entertaining the recipes all sound delicious ill be working my way through many of these starting of course with the sourdough hotcakes and bread the handwritten format just adds to the charm of the book if youre interested in learning to make sourdoughs this book will give you all the information youll need".

Gambar 4.4 Halaman input teks hasil prediksi

### 4.2.2 Implementasi Algoritma

Gambar 4.5 menunjukkan proses pengambilan data *train* dan data *test* dari *file* CSV untuk *dataset* Amazon, IMDB, dan Yelp. Tiap *file* tersebut terdiri dari kolom sentimen dan kolom label. Kolom sentimen terdiri dari data sentimen *review* tiap *dataset* yang belum diproses, kolom label terdiri dari label-label setiap sentimen yang terdiri dari angka satu (1) dan angka (0) yang merepresentasikan sentimen positif atau sentimen negatif.

### Baca file csv

```
In [2]: #baca csv
data_train_amazon = pd.read_csv('Amazon_Train.csv')
data_train_yelp = pd.read_csv('Yelp_Train.csv')
data_test_amazon = pd.read_csv('Amazon_Test.csv')
data_test_yelp = pd.read_csv('Yelp_Test.csv')
data_train_imdb = pd.read_csv('IMDB_Train.csv')
data_test_imdb = pd.read_csv('IMDB_Test.csv')
```

```
In [4]: print(data_train_amazon.head())
print("\n")
print(data_train_yelp.head())
print("\n")
print(data_train_imdb.head())
```

	Label	Sentimen
0	0	Buyer beware This is a self-published book, an...
1	0	The Worst! A complete waste of time. Typograph...
2	0	Oh please I guess you have to be a romance nov...
3	0	Awful beyond belief! I feel I have to write to...
4	0	Another Abysmal Digital Copy Rather than scrat...

	Label	Sentimen
0	0	I don't know what Dr. Goldberg was like before...
1	0	I'm writing this review to give you a heads up...
2	0	Owning a driving range inside the city limits ...
3	0	This place is absolute garbage... Half of the...
4	0	Used to go there for tires, brakes, etc. Thei...

	Label	Sentimen
0	0	Story of a man who has unnatural feelings for ...
1	0	Airport '77 starts as a brand new luxury 747 p...
2	0	This film lacked something I couldn't put my f...
3	0	Sorry everyone,,, I know this is supposed to b...
4	0	When I was little my parents took me along to ...

Gambar 4.5 Proses membaca *dataset*

Gambar 4.6 merupakan proses *preprocessing* data *train* dan data *test* pada masing-masing dataset. Proses ini terdiri dari konversi ke *lowercase*, menghilangkan angka, menghilangkan tanda baca, dan menghilangkan karakter non alfabet dimana proses tersebut menggunakan *library* regular expression dan *string*. *Preprocessing* ini dilakukan pada setiap baris data pada masing-masing *dataset* dengan menggunakan fungsi *apply* dan fungsi *lamda* yang berguna untuk mengubah semua data pada *dataframe pandas* menjadi data dengan nilai yang diinginkan.

**Preprocessing**

```

data_Preprocessing_Amazon_Train = data_train_amazon.Sentimen.astype(str)
data_Preprocessing_Amazon_Train = data_Preprocessing_Amazon_Train.apply(lambda x: x.lower())
data_Preprocessing_Amazon_Train = data_Preprocessing_Amazon_Train.apply(lambda x: re.sub(r"\d", "", x))
data_Preprocessing_Amazon_Train = data_Preprocessing_Amazon_Train.apply(
    lambda x: x.translate(str.maketrans('', '', string.punctuation)))
data_Preprocessing_Amazon_Train = data_Preprocessing_Amazon_Train.apply(
    lambda x: " ".join(re.findall("[a-zA-Z]+", x)))

data_Preprocessing_Yelp_Train = data_train_yelp.Sentimen.astype(str)
data_Preprocessing_Yelp_Train = data_Preprocessing_Yelp_Train.apply(lambda x: x.lower())
data_Preprocessing_Yelp_Train = data_Preprocessing_Yelp_Train.apply(lambda x: re.sub(r"\d", "", x))
data_Preprocessing_Yelp_Train = data_Preprocessing_Yelp_Train.apply(
    lambda x: x.translate(str.maketrans('', '', string.punctuation)))
data_Preprocessing_Yelp_Train = data_Preprocessing_Yelp_Train.apply(
    lambda x: " ".join(re.findall("[a-zA-Z]+", x)))

data_Preprocessing_IMDB_Train = data_train_imdb.Sentimen.astype(str)
data_Preprocessing_IMDB_Train = data_Preprocessing_IMDB_Train.apply(lambda x: x.lower())
data_Preprocessing_IMDB_Train = data_Preprocessing_IMDB_Train.apply(lambda x: re.sub(r"\d", "", x))
data_Preprocessing_IMDB_Train = data_Preprocessing_IMDB_Train.apply(
    lambda x: x.translate(str.maketrans('', '', string.punctuation)))
data_Preprocessing_IMDB_Train = data_Preprocessing_IMDB_Train.apply(
    lambda x: " ".join(re.findall("[a-zA-Z]+", x)))

data_Preprocessing_Amazon_Test = data_test_amazon.Sentimen.astype(str)
data_Preprocessing_Amazon_Test = data_Preprocessing_Amazon_Test.apply(lambda x: x.lower())
data_Preprocessing_Amazon_Test = data_Preprocessing_Amazon_Test.apply(lambda x: re.sub(r"\d", "", x))
data_Preprocessing_Amazon_Test = data_Preprocessing_Amazon_Test.apply(lambda x: x.translate(
    str.maketrans('', '', string.punctuation)))
data_Preprocessing_Amazon_Test = data_Preprocessing_Amazon_Test.apply(lambda x: " ".join(
    re.findall("[a-zA-Z]+", x)))

data_Preprocessing_Yelp_Test = data_test_yelp.Sentimen.astype(str)
data_Preprocessing_Yelp_Test = data_Preprocessing_Yelp_Test.apply(lambda x: x.lower())
data_Preprocessing_Yelp_Test = data_Preprocessing_Yelp_Test.apply(lambda x: re.sub(r"\d", "", x))
data_Preprocessing_Yelp_Test = data_Preprocessing_Yelp_Test.apply(lambda x: x.translate(
    str.maketrans('', '', string.punctuation)))
data_Preprocessing_Yelp_Test = data_Preprocessing_Yelp_Test.apply(lambda x: " ".join(
    re.findall("[a-zA-Z]+", x)))

data_Preprocessing_IMDB_Test = data_test_imdb.Sentimen.astype(str)
data_Preprocessing_IMDB_Test = data_Preprocessing_IMDB_Test.apply(lambda x: x.lower())
data_Preprocessing_IMDB_Test = data_Preprocessing_IMDB_Test.apply(lambda x: re.sub(r"\d", "", x))
data_Preprocessing_IMDB_Test = data_Preprocessing_IMDB_Test.apply(lambda x: x.translate(
    str.maketrans('', '', string.punctuation)))
data_Preprocessing_IMDB_Test = data_Preprocessing_IMDB_Test.apply(lambda x: " ".join(
    re.findall("[a-zA-Z]+", x)))

```

Gambar 4.5 Tahap preprocessing dataset

Gambar 4.6 menunjukkan sebuah fungsi yang digunakan untuk mengubah data sentimen yang telah diproses menjadi bentuk *bigrams* dan *trigrams*. Dalam penggunaan fungsi tersebut, akan mengembalikan *list* n-grams (*bigrams* dan *trigrams*) untuk setiap data pada *dataset* yang di-*passing* dan mengembalikan *vocabulary* atau semua *term* n-grams (*bigrams* dan *trigrams*) unik yang terdapat pada *dataset* tersebut. Dalam penerapannya, *list term* n-grams didapatkan dengan menggunakan *library* n-grams dari nltk.

**N-Gram (bigram dan trigram)**

```
def bigram_trigram(Clean_Sentiment) :
    iterator = 0
    Vocabulary = []
    ngram_result = []
    while iterator < len(Clean_Sentiment) :
        ngram = 2
        sentence = Clean_Sentiment[iterator]
        vocab = list(ngrams(sentence, ngram))
        temp = vocab

        iterator2 = 0
        while iterator2 < len(vocab):
            if(vocab[iterator2] not in Vocabulary) :
                Vocabulary.append(vocab[iterator2])
            iterator2 = iterator2 + 1

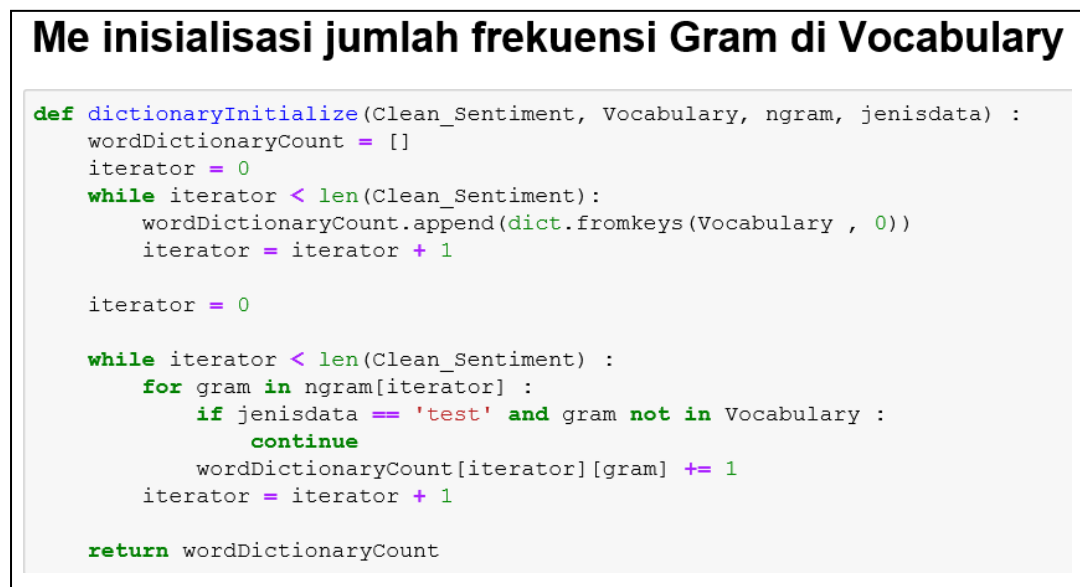
        ngram = 3
        vocab = list(ngrams(sentence, ngram))
        temp = temp + vocab
        ngram_result.append(temp)
        iterator2 = 0
        while iterator2 < len(vocab):
            if(vocab[iterator2] not in Vocabulary) :
                Vocabulary.append(vocab[iterator2])
            iterator2 = iterator2 + 1
        iterator = iterator + 1

    return ngram_result, Vocabulary
```

Gambar 4.6 pengubahan sentimen dalam bentuk n-grams



Gambar 4.7 menunjukkan sebuah fungsi yang digunakan untuk inisialisasi sebuah *dictionary* untuk data *train* dan data *test* untuk setiap *dataset*. Kolom dari *dictionary* tersebut merupakan *vocabulary* atau semua *term* yang terdapat dari suatu *dataset* dan baris dari *dictionary* tersebut merupakan frekuensi kemunculan setiap *term* pada suatu baris data. Gambar 4.8 menunjukkan dua contoh hasil dari inisialisasi *dictionary* yang didapatkan dari proses pada Gambar 4.7 pada penerapan data *train* Amazon.



Gambar 4.7 fungsi inisialisasi dictionary n-grams

pd.DataFrame([WordDictionaryCount\_Amazon\_Train[0],WordDictionaryCount\_Amazon\_Train[1]])

	bu	uy	ye	er	r	b	be	ew	wa	ar	...	umy	myu	mmc	sez	fja	coz	ozy	kav	gju	ukw
0	1	1	1	11	9	13	5	3	4	7	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	5	1	0	0	1	2	...	0	0	0	0	0	0	0	0	0	0

Gambar 4.8 Hasil inisialisasi *dictionary* n-grams

Gambar 4.9 menunjukkan sebuah fungsi yang digunakan untuk menghitung nilai inverse document frequency atau IDF untuk setiap term pada data *train* dan data *test*. Langkah pertama yang dilakukan adalah membuat sebuah *list* yang berfungsi untuk menampung jumlah data atau dokumen yang memiliki *term t* di dalamnya. Lalu dihitunglah nilai IDF untuk setiap *term* dengan menggunakan rumus sebagai berikut :

$$IDF = \log ((1 + N) / (1 + Val )) + 1 \quad (4.1)$$

Keterangan :

1. *N* = Banyaknya baris data pada data *train*
2. *Val* = banyaknya data yang mempunyai *term* ke index

Fungsi tersebut akan mengembalikan nilai berupa *list* nilai IDF untuk setiap *term* dan list yang menampung jumlah data atau dokumen yang memiliki *term t* di dalamnya yang akan digunakan untuk melakukan *transfer learning* nilai IDF.

```
def computeIDF(WordDict):
    idfDict = {}
    number_of_document_with_term_t_in_it = {}
    N = len(WordDict)

    idfDict = dict.fromkeys(WordDict[0].keys(), 0)
    for doc in WordDict:
        for word, val in doc.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        number_of_document_with_term_t_in_it[word] = float(val)
        idfDict[word] = np.log((1+N) / (1+val)) + 1

    return idfDict, number_of_document_with_term_t_in_it
```

Gambar 4.9 Fungsi untuk inisialisasi nilai IDF

Gambar 4.10 menunjukkan sebuah fungsi yang digunakan untuk menghitung nilai Term Frequency- Inverse Document Frequency (TF-IDF) sebelum normalisasi untuk setiap *term* pada setiap baris data suatu *dataset*. Nilai TF-IDF non normalisasi pada suatu *term* dihitung dengan cara mengalikan nilai frekuensi kemunculan *term* tersebut pada suatu baris data dengan nilai IDF *term* tersebut yang telah dihitung pada fungsi sebelumnya. Fungsi ini mengembalikan nilai hasil perhitungan TF-IDF tersebut. Langkah selanjutnya adalah menormalisasi nilai TF-IDF tersebut. Gambar 4.11 menunjukkan sebuah fungsi yang digunakan untuk menormalisasi nilai TF-IDF pada setiap *term*. Untuk menormalisasi nilai TF-IDF, digunakan rumus sebagai berikut :

$$V_{norm} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \dots + v_n^2}} \quad (4.2)$$

Keterangan :

1.  $v$  = nilai TF-IDF non normalisasi

```
def computeTFIDFnonScale(CountFrequency, idfs):
    tfidf = None
    tfidf = {}
    for word, val in CountFrequency.items():
        tfidf[word] = val*idfs[word]
    return tfidf
```

Gambar 4.10 Fungsi untuk inialisasi nilai TF-IDF

```
def computeTFIDF(tfidf_nonscaled) :
    tfidf_nonscaled = np.array(tfidf_nonscaled)
    tfidf_list = tfidf_nonscaled/sum(tfidf_nonscaled**2)**0.5
    return tfidf_list
```

Gambar 4.11 Fungsi untuk normalisasi nilai TF-IDF

Hasil dari normalisasi nilai TF-IDF ditunjukkan pada Gambar 4.12 berupa *dataframe* dari *dictionary* nilai TF-IDF untuk setiap *term* pada setiap baris data. Gambar 4.13 menunjukkan proses deklarasi model klasifikasi dengan menggunakan Random Forest Classifier dari scikit-learn dengan *hyperparameter* yang didapatkan dari *tuning* menggunakan GridSearchCV. GridSearchCV merupakan salah satu metode untuk mencari nilai dari *hyperparameter* yang memiliki pengaruh yang lebih baik untuk performa suatu model. Berikut merupakan *hyperparameter* yang digunakan.

1. `max_depth`, yaitu kedalaman maksimal dari *tree* yang digunakan adalah lima tingkat.
2. `n_estimators`, yaitu banyaknya *tree* yang digunakan di dalam *forest* adalah sebanyak 800 *tree*.
3. `min_samples_split`, yaitu jumlah sampel minimal yang diperlukan untuk membelah *node* internal adalah sebanyak lima sampel.
4. `min_samples_leaf`, yaitu jumlah minimum sampel yang diperlukan untuk berada pada *leaf node* sebanyak satu sampel.
5. `bootstrap`, yang bernilai *false* yang berarti seluruh *data* digunakan untuk membuat setiap *tree*.
6. `random_state`, yaitu indeks untuk mengontrol pengacakan angka yang di *generate* adalah 42.

```
DataFrame_Amazon_Train = pd.DataFrame(Vectorizer_Amazon_Train, columns = Vocabulary_Amazon_Train)
DataFrame_Amazon_Train = DataFrame_Amazon_Train.reindex(sorted(DataFrame_Amazon_Train.columns), axis=1)
print (DataFrame_Amazon_Train)
```

	( , a)	( , a, )	( , a, a)	( , a, b)	( , a, c)	( , a, d) \
0	0.174588	0.080901	0.0	0.000000	0.000000	0.000000
1	0.149335	0.047575	0.0	0.044579	0.052005	0.056086
2	0.122767	0.056888	0.0	0.026653	0.000000	0.000000
3	0.106905	0.041917	0.0	0.000000	0.000000	0.000000
4	0.121334	0.023193	0.0	0.014488	0.016902	0.000000
...	...	...	...	...	...	...
3745	0.134475	0.000000	0.0	0.035683	0.041627	0.000000
3746	0.093831	0.027593	0.0	0.000000	0.000000	0.000000
3747	0.112662	0.028713	0.0	0.026905	0.000000	0.033850
3748	0.136468	0.041405	0.0	0.000000	0.000000	0.000000
3749	0.093960	0.059867	0.0	0.037398	0.000000	0.000000

	( , a, e)	( , a, f)	( , a, g)	( , a, h)	... (z, z, a)	(z, z, b) \
0	0.0	0.000000	0.000000	0.0	...	0.0
1	0.0	0.000000	0.000000	0.0	...	0.0
2	0.0	0.033833	0.000000	0.0	...	0.0
3	0.0	0.000000	0.024315	0.0	...	0.0
4	0.0	0.000000	0.000000	0.0	...	0.0
...	...	...	...	...	...	...
3745	0.0	0.090590	0.000000	0.0	...	0.0
3746	0.0	0.000000	0.000000	0.0	...	0.0
3747	0.0	0.034153	0.000000	0.0	...	0.0
3748	0.0	0.019700	0.000000	0.0	...	0.0
3749	0.0	0.000000	0.000000	0.0	...	0.0

Gambar 4.12 Hasil dari normalisasi nilai TF-IDF

```
Waktu_Training = time()
RF_Classifier_Yelp = RandomForestClassifier(
    max_depth= 5, n_estimators = 800,
    random_state=42, bootstrap = False,
    min_samples_split = 5, min_samples_leaf = 1)
RF_Classifier_Yelp.fit(Vectorizer_Yelp_Train, Label_Yelp_Train)
print(f"\nWaktu Training Yelp: {round(time()-Waktu_Training, 3)}s")

Waktu Training Yelp: 25.348s
```

Gambar 4.13 Inisialisasi model klasifikasi Random Forest

Gambar 4.14 menggambarkan proses yang digunakan untuk mengukur performa dari model Random Forest Classifier yang sudah dibangun sebelumnya, pada contoh ini, diambil uji coba pada model *dataset* Yelp. Performa yang di uji antara lain adalah waktu yang diperlukan untuk membangun model klasifikasi, waktu prediksi data, skor akurasi, precision, recall, dan f1-score. Untuk menghitung skor akurasi digunakan fitur pengembalian *score* pada Random Forest Classifier, sedangkan untuk menghitung nilai precision, recall, dan f1-score digunakan fitur *metrics classification report* dari scikit learn.

```

Waktu_Predict_Train = time()
Skor_Train_Yelp = RF_Classifier_Yelp.score(TF_IDF_Yelp_Train_, Label_Yelp_Train)
print(f"waktu prediksi (train): {round(time()-Waktu_Predict_Train, 3)}s")

Waktu_Predict_Test = time()
Skor_Test_Yelp = RF_Classifier_Yelp.score(TF_IDF_Yelp_Test_, Label_Yelp_Test)
print(f"waktu prediksi (test): {round(time()-Waktu_Predict_Test, 3)}s")

print("\nSkor Random Forest Train Interseksi : {}".format(Skor_Train_Yelp))
print("Skor Random Forest Test Interseksi : {}".format(Skor_Test_Yelp))
print("-----\n\n")

RFCClassifier_predict = RF_Classifier_Yelp.predict(TF_IDF_Yelp_Train_)
Confusion_matrix = confusion_matrix(Label_Yelp_Train, RFCClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_Yelp_Train, RFCClassifier_predict))

print("-----\n\n")
RFCClassifier_predict = RF_Classifier_Yelp.predict(TF_IDF_Yelp_Test_)
Confusion_matrix = confusion_matrix(Label_Yelp_Test, RFCClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_Yelp_Test, RFCClassifier_predict))

waktu prediksi (train): 2.55s
waktu prediksi (test): 0.861s

Skor Random Forest Train Interseksi : 0.9048
Skor Random Forest Test Interseksi : 0.8624
-----

[[1653  222]
 [ 135 1740]]
      precision    recall  f1-score   support

         0         0.92      0.88      0.90        1875
         1         0.89      0.93      0.91        1875

 accuracy          0.90        3750
 macro avg         0.91      0.90      0.90        3750
weighted avg         0.91      0.90      0.90        3750

-----

[[539  86]
 [ 86 539]]
      precision    recall  f1-score   support

         0         0.86      0.86      0.86         625
         1         0.86      0.86      0.86         625

 accuracy          0.86        1250
 macro avg         0.86      0.86      0.86        1250
weighted avg         0.86      0.86      0.86        1250

```

Gambar 4.14 Uji performa model klasifikasi

### 4.3 Uji Coba Metode Transfer Learning

Pada penelitian ini dilakukan metode *transfer learning* pada informasi atau *knowledge* yang diterima pada model yang telah dibangun dengan menggunakan suatu *dataset* akan diterapkan untuk membangun model klasifikasi lainnya dengan menggunakan *dataset* selanjutnya. Pada penerapannya, *transfer learning* akan dilakukan dengan menggunakan informasi dari *list feature\_importances* yang didapatkan dari fitur pengembalian Random Forest Classifier. *Transfer learning* juga diterapkan untuk mengirim informasi berupa nilai IDF pada suatu model dan menerapkan pada model berikutnya. Untuk masing-masing penerapan *transfer learning* akan dilakukan dengan dua kali uji coba yakni dengan menggunakan informasi model klasifikasi dari *dataset* Amazon yang diterapkan kepada *dataset* Yelp, dan menggunakan informasi dari model klasifikasi *transfer learning* tersebut untuk digunakan pada *dataset* IMDB.

#### 4.3.1 Uji Coba Transfer Learning Feature Importances

Untuk melakukan *transfer learning feature importance* diperlukan *list* yang terdiri dari daftar *term-term* yang memiliki nilai *feature importance* yang lebih dari nol. Gambar 4.15 menunjukkan langkah untuk inisialisasi *list* tersebut. *List* tersebut diinisialisasi dengan cara *looping* setiap *term* yang terdapat pada *vocabulary* masing-masing *dataset*, dan mendapatkan *term* yang memiliki *feature importance* yang lebih dari nol.

```
Vocabulary_Importance_Amazon = []
iterator = 0
length = len (Vocabulary_Train_Amazon)
while iterator < length :
    if RF_Classifier_Amazon.feature_importances_[iterator] > 0 :
        Vocabulary_Importance_Amazon.append(Vocabulary_Train_Amazon[iterator])
    iterator = iterator + 1
```

Gambar 4.15 Fungsi untuk inisialisai *list term feature importance*

Skenario pertama dilakukan dengan hanya mengambil *term-term* yang interseksi pada kedua *dataset*. Uji coba pertama dalam penerapan *transfer learning feature importance* skenario ini adalah dengan menggunakan model klasifikasi dengan *dataset* Amazon dan menerapkannya pada *dataset* Yelp yang akan dijadikan model klasifikasi. Gambar 4.16 menunjukkan langkah untuk mendapatkan *vocabulary* interseksi antara *dataset* Amazon dan Yelp. Setelah *list vocabulary term* interseksi didapatkan, maka langkah selanjutnya adalah membuat *dictionary* dan menghitung frekuensi masing-masing *term* disetiap baris data. Gambar 4.17 merupakan proses untuk membuat *dictionary* frekuensi tiap *term* pada data *train* dan data *test* Yelp dengan *vocabulary* interseksi yang memanggil fungsi *dictionaryInitialize* yang telah ditunjukkan sebelumnya pada Gambar 4.7.

```
Vocabulary_intersection = list(set(  
    Vocabulary_Importance_Amazon) & set (  
    Vocabulary_Train_Yelp))
```

Gambar 4.16 Proses pembuatan *list term* interseksi

```
WordDictionary_Transfer_Learning_Train = dictionaryInitialize(  
    Clean_Yelp_Sentiment_Train,  
    Vocabulary_intersection,  
    ngram_yelp_train, 'transfer')  
  
WordDictionary_feature_importance_Test = dictionaryInitialize(  
    Clean_Yelp_Sentiment_Test,  
    Vocabulary_intersection,  
    ngram_yelp_test, 'test')|
```

Gambar 4.17 Proses pembuatan *dictionary train test* baru



Gambar 4.18 menunjukkan hasil dari *dictionary term* pada data *train* untuk membangun model klasifikasi dari dataset Yelp sebelum dan setelah diberlakukan pemilihan *term* secara interseksi pada data *train* Yelp dan data *train* Amazon yang memiliki nilai *feature\_importances* lebih besar dari nol. Sebelum diberlakukannya seleksi *term*, jumlah fitur *term* pada data *train* Yelp adalah sebanyak 8197 fitur, namun setelah diberlakukan seleksi *term* menjadi 2822 fitur. Hal ini menunjukkan terdapat sebanyak 5375 fitur yang memiliki nilai *feature importances* lebih kecil atau sama dengan nol.

```
pd.DataFrame([WordDictionaryCount_Yelp_Train[0], WordDictionaryCount_Yelp_Train[1]])
```

	(i,)	(, d)	(d, o)	(o, n)	(n, t)	(t,)	(, k)	(k, n)	(n, o)	(o, w)	...	(w, m, o)	(i, n, x)	(d, s, z)	(s, z, i)	(r, f, f)	(n, w,)	(s, n, q)	(u, b, g)	(b, g, u)	(y, c, r)
0	4	11	6	13	7	19	1	1	5	1	...	0	0	0	0	0	0	0	0	0	0
1	10	7	6	10	13	21	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0

2 rows x 8197 columns

```
pd.DataFrame([WordDictionary_feature_importance_Train[0], WordDictionary_feature_importance_Train[1]])
```

	(g, l)	(u, t, s)	(d, u, l)	(r, m, o)	(n, o, w)	(n, i, a)	(, s, l)	(k, w)	(, w, r)	(e, r, v)	...	(t, e, s)	(i, n)	(i, p, l)	(t, u, d)	(h, n,)	(, k, u)	(e, r, w)	(n, j)	(i, l, m)	(l, s, e)
0	0	0	1	0	1	0	1	0	1	1	...	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	1	0	...	0	0	1	2	0	0	0	0	0	0

2 rows x 2822 columns

Gambar 4.18 Hasil pemilihan *term* dengan interseksi *feature importance*

Setelah membuat *dictionary* baru, langkah selanjutnya adalah menghitung nilai IDF berdasarkan *list term* pada *dictionary* baru. Gambar 4.19 merupakan proses untuk membuat *list* IDF tersebut dengan memanggil fungsi yang telah diinisialisasi yang ditunjukkan pada Gambar 4.9. Fungsi tersebut akan mengembalikan nilai IDF yang baru.

```

IDF_Feature_Importance, number_term = computeIDF(WordDictionary_feature_importance_Train)
IDF_Feature_Importance
('e', 'r', 'v'): 1.9470165704762945,
('m', 'y', 's'): 4.425756705348631,
('o', 'd'): 1.5394917294051194,
('i', 'b'): 2.5302774099202097,
('y', 't', 'i'): 5.9339408840775585,
('s', 't'): 1.1000133042877158,
('e', 'd', 'g'): 4.346975827495516,
('u', 'r', 's'): 3.051833636031287,
('o', 't', 'e'): 3.4337199993165153,
('l', 'e', 'r'): 3.765945945056277,
('l', 'u', 'c'): 4.5758173999243645,
('u', 't', 'u'): 5.222444564849416,
('n', 'h', 'e'): 5.152240306176168,
('n', 'o', 'y'): 5.445588116163626,
('o', 'r', ' '): 1.345577816405848,
('r', 'a'): 1.5240649261874597,
('r', 'o', 't'): 4.205897229235611,
('a', ' ', 'm'): 2.8361869961312554,
('o', 'h'): 3.791698441158692,

```

Gambar 4.19 Proses perhitungan nilai IDF *feature importances*

Setelah membuat *list* nilai IDF baru, langkah selanjutnya adalah menghitung nilai TF-IDF berdasarkan *list* nilai IDF yang baru tersebut. Gambar 4.20 merupakan proses untuk membuat *list* TF-IDF tersebut dengan memanggil fungsi yang telah diinisialisasi yang ditunjukkan pada Gambar 4.10 dan Gambar 4.11.

```

def getTFIDF(wordDictionary, CountFrequency, IDF):
    TFIDF_non_scaled = None
    TFIDF_non_scaled = []
    iterator = 0
    while iterator < len(wordDictionary):
        TFIDF_non_scaled.append(computeTFIDFNonScale(CountFrequency[iterator], IDF))
        iterator = iterator + 1

    TFIDF = []
    iterator = 0
    while iterator < len(wordDictionary):
        tf_idf_list = list(TFIDF_non_scaled[iterator].values())
        TFIDF.append(computeTFIDF(tf_idf_list))
        iterator = iterator + 1

    return TFIDF

TFIDF_Feature_importance_Test = getTFIDF(
    WordDictionary_feature_importance_Train,
    CountFrequency_Sentimen_Train_FI,
    IDF_Feature_Importance_Train)

```

Gambar 4.20 Proses perhitungan nilai TF-IDF *feature importance*

Setelah membuat *list* nilai TF-IDF baru, langkah selanjutnya adalah membuat model klasifikasi dari Random Forest Classifier berdasarkan *list* nilai TF-IDF yang baru. Gambar 4.21 menunjukkan proses deklarasi model klasifikasi dengan menggunakan Random Forest Classifier berdasarkan nilai TF-IDF tersebut. Model klasifikasi yang dibangun menggunakan *hyperparameter* yang sama dengan sebelumnya. Dengan waktu training sekitar 20 detik, lebih cepat sekitar 5 detik dibandingkan sebelum diterapkan penyisihan *term* yang dapat dilihat pada Gambar 4.13 Hasil uji performa yang dihasilkan oleh model yang baru ditunjukkan pada Gambar 4.22.

```
Waktu_Training = time()
RF_Classifier_TF_FI = RandomForestClassifier(
    max_depth= 5, n_estimators = 800, random_state=42,
    bootstrap = False, min_samples_split = 5, min_samples_leaf = 1, max_features = 'auto')
RF_Classifier_TF_FI.fit(TFIDF_Feature_Importance_Train, Label_Yelp_Train)
print(f"\nWaktu Training: {round(time()-Waktu_Training, 3)}s")
```

Waktu Training: 20.673s

Gambar 4.21 Inisialisasi model klasifikasi

```

Waktu_Predict_Train = time()
Skor_Train_Interseksi_Yelp = RF_Classifier_TF_FI.score(TFIDF_Feature_importance_Train, Label_Yelp_Train)
print(f"waktu prediksi (train): {round(time()-Waktu_Predict_Train, 3)}s")

Waktu_Predict_Test = time()
Skor_Test_Interseksi_Yelp = RF_Classifier_TF_FI.score(TFIDF_Feature_importance_Test, Label_Yelp_Test)
print(f"waktu prediksi (test): {round(time()-Waktu_Predict_Test, 3)}s")

print("\nSkor Random Forest Train Interseksi : {}".format(Skor_Train_Interseksi_Yelp))
print("Skor Random Forest Test Interseksi : {}".format(Skor_Test_Interseksi_Yelp))
print("-----\n\n")

RFCClassifier_predict = RF_Classifier_TF_FI.predict(TFIDF_Feature_importance_Train)
Confusion_matrix = confusion_matrix(Label_Yelp_Train, RFCClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_Yelp_Train, RFCClassifier_predict))

print("-----\n\n")
RFCClassifier_predict = RF_Classifier_TF_FI.predict(TFIDF_Feature_importance_Test)
Confusion_matrix = confusion_matrix(Label_Yelp_Test, RFCClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_Yelp_Test, RFCClassifier_predict))

waktu prediksi (train): 1.157s
waktu prediksi (test): 0.386s

Skor Random Forest Train Interseksi : 0.9069333333333334
Skor Random Forest Test Interseksi : 0.8672
-----

[[1689 186]
 [ 163 1712]]
      precision    recall  f1-score   support

         0       0.91      0.90      0.91      1875
         1       0.90      0.91      0.91      1875

 accuracy          0.91
 macro avg          0.91
weighted avg          0.91

-----

[[548 77]
 [ 89 536]]
      precision    recall  f1-score   support

         0       0.86      0.88      0.87      625
         1       0.87      0.86      0.87      625

 accuracy          0.87
 macro avg          0.87
weighted avg          0.87

```

Gambar 4.22 Hasil uji performa model

Uji coba kedua dalam penerapan *transfer learning feature importance* skenario ini adalah dengan menggunakan informasi model klasifikasi dengan *dataset* Amazon dan menerapkannya pada *dataset* IMDB yang akan dijadikan model klasifikasi. Proses pengerjaannya sama seperti uji coba pertama pada skenario ini. Prosesnya yaitu mencari *vocabulary* interseksi antara kedua *dataset* (Amazon dan IMDB), membuat *dictionary* dan menghitung frekuensi masing-masing *term* disetiap baris data berdasarkan *vocabulary* yang baru, menghitung nilai IDF baru pada masing-masing, menghitung nilai TF-IDF, dan membuat model klasifikasi yang baru.

Gambar 4.23 menunjukkan hasil dari *dictionary term* pada data *train* untuk membangun model klasifikasi dari dataset IMDB sebelum dan setelah diberlakukan pemilihan *term* secara interseksi pada data *train* IMDB dan data *train* Amazon yang memiliki nilai *feature\_importances* lebih besar dari nol. Sebelum diberlakukannya seleksi *term*, jumlah fitur *term* pada data *train* Yelp adalah sebanyak 9284 fitur, namun setelah diberlakukan seleksi *term* menjadi 2834 fitur. Hal ini menunjukkan terdapat sebanyak 6450 fitur yang memiliki nilai *feature importances* lebih kecil atau sama dengan nol. Gambar 4.24 hasil dari pembuatan nilai IDF yang baru berdasarkan *dictionary term* yang baru.

```
pd.DataFrame([WordDictionaryCount_IMDB_Train[0], WordDictionaryCount_IMDB_Train[1]])
```

	(s, t)	(t, o)	(o, r)	(r, y)	(y, .)	(. , o)	(o, f)	(f, .)	(. , a)	(a, .)	...	(k, b, l)	(d, m, y)	(p, s, n)	(g, p, i)	(. , l, w)	(w, r, .)	(t, t, t)	(u, a, x)	(h, b, y)	(w, o, f)
0	7	5	6	2	10	9	5	5	12	9	...	0	0	0	0	0	0	0	0	0	0
1	25	19	47	5	33	65	25	29	81	28	...	0	0	0	0	0	0	0	0	0	0

2 rows × 9284 columns

```
pd.DataFrame([WordDictionary_feature_importance_Train[0], WordDictionary_feature_importance_Train[1]])
```

	(c, l, i)	(l, e, t)	(r, o, g)	(w, e, .)	(s, h, o)	(l, l, u)	(l, a, n)	(k, l, a)	(r, i, g)	(u, i, c)	...	(t, h, l)	(t, h, e)	(s, u, s)	(p, o, e)	(g, ., m)	(n, k, .)	(f, ., h)	(s, p, u)	(e, t, y)	(a, r, a)
0	0	0	0	0	1	0	1	1	0	0	...	0	4	0	0	0	1	0	0	0	0
1	0	2	0	0	4	0	13	0	6	0	...	1	72	2	0	0	2	1	0	0	3

2 rows × 2834 columns

Gambar 4.23 Hasil pemilihan *term* dengan interseksi *feature importance*

```
IDF_Feature_Importance, number_term = computeIDF(WordDictionary_feature_importance_Train)
IDF_Feature_Importance
```

```
{('c', 'l', 'i'): 3.2258906829753484,
 ('l', 'e', 't'): 2.398903515435708,
 ('x', 'o', 'g'): 3.9264728420228114,
 ('w', 'e', ' '): 2.470522479418194,
 ('s', 'h', 'o'): 1.6817487801468731,
 ('l', 'l', 'u'): 4.493579301687392,
 ('l', 'a', 'n'): 2.353513138191121,
 ('k', 'l', 'a'): 5.379630148371829,
 ('x', 'i', 'g'): 2.3371361089097986,
 ('u', 'i', 'c'): 3.941510719387352,
 ('c', 'e', 'l'): 3.0191776730572344,
 (' ', 't', 'h'): 1.0008001067235743,
 ('l', ' ', 'd'): 2.7683095737281698,
 ('d', 'a', 'n'): 3.131703467915647,
 ('k', ' ', 'f'): 3.3326238824451466,
 ('c', 'a', 's'): 2.2905238040403795,
 ('n', 't', 'e'): 1.6409478717740744,
 ('b', 'u', 'n'): 4.259964450505887,
 ('i', 's', 't'): 1.7298012091297656,
 ('t', ' ', 't'): 4.885073222222222}
```

Gambar 4.24 Proses perhitungan nilai IDF *feature importances*

Sama dengan uji coba pertama, langkah selanjutnya adalah menghitung kembali nilai TF-IDF dan inialisasi model klasifikasi Random Forest Classifier dengan *list* nilai TF-IDF tersebut yang ditunjukkan oleh Gambar 4.25. Dengan waktu training sekitar 26 detik, lebih cepat sekitar 5 detik dibandingkan sebelum diterapkan penyisihan *term* yang dapat dilihat pada Gambar 4.26. Hasil uji coba performa model klasifikasi baru ditunjukkan pada Gambar 4.27.

```

: Waktu_Training = time()
RF_Classifier_TF_FI = RandomForestClassifier(
    max_depth= 5, n_estimators = 800, random_state=42,
    bootstrap = False, min_samples_split = 5, min_samples_leaf = 1, max_features = 'auto')
RF_Classifier_TF_FI.fit(TFIDF_Feature_importance_Train, Label_IMDB_Train)
print(f"\nWaktu Training: {round(time()-Waktu_Training, 3)}s")

```

Waktu Training: 26.89s

Gambar 4.25 Inisialisasi model klasifikasi setelah diterapkan seleksi *term*

```

: Waktu_Training = time()
RF_Classifier_IMDB = RandomForestClassifier(max_depth= 5,
    n_estimators = 800, random_state=42,
    bootstrap = False, min_samples_split = 5, min_samples_leaf = 1, max_features = 'auto')
RF_Classifier_IMDB.fit(Vectorizer_IMDB_Train, Label_IMDB_Train)
print(f"\nWaktu Training IMDB: {round(time()-Waktu_Training, 3)}s")

```

Waktu Training IMDB: 31.08s

Gambar 4.26 Inisialisasi model klasifikasi sebelum diterapkan seleksi *term*

```

Waktu_Predict_Train = time()
Skor_Train_Interseksi_IMDB = RF_Classifier_TF_FI.score(TFIDF_Feature_importance_Train, Label_IMDB_Train)
print(f"waktu prediksi (train): {round(time()-Waktu_Predict_Train, 3)}s")

Waktu_Predict_Test = time()
Skor_Test_Interseksi_IMDB = RF_Classifier_TF_FI.score(TFIDF_Feature_importance_Test, Label_IMDB_Test)
print(f"waktu prediksi (test): {round(time()-Waktu_Predict_Test, 3)}s")

print("\nSkor Random Forest Train Interseksi : {}".format(Skor_Train_Interseksi_IMDB))
print("Skor Random Forest Test Interseksi : {}".format(Skor_Test_Interseksi_IMDB))
print("-----\n\n")

RFCClassifier_predict = RF_Classifier_TF_FI.predict(TFIDF_Feature_importance_Train)
Confusion_matrix = confusion_matrix(Label_IMDB_Train, RFCClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_IMDB_Train, RFCClassifier_predict))

print("-----\n\n")
RFCClassifier_predict = RF_Classifier_TF_FI.predict(TFIDF_Feature_importance_Test)
Confusion_matrix = confusion_matrix(Label_IMDB_Test, RFCClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_IMDB_Test, RFCClassifier_predict))

waktu prediksi (train): 1.298s
waktu prediksi (test): 0.472s

Skor Random Forest Train Interseksi : 0.9026666666666666
Skor Random Forest Test Interseksi : 0.7752
-----

[[1670 205]
 [ 160 1715]]
      precision    recall  f1-score   support

     0       0.91      0.89      0.90       1875
     1       0.89      0.91      0.90       1875

 accuracy          0.90
 macro avg          0.90
weighted avg          0.90

-----

[[468 157]
 [124 501]]
      precision    recall  f1-score   support

     0       0.79      0.75      0.77        625
     1       0.76      0.80      0.78        625

 accuracy          0.78
 macro avg          0.78
weighted avg          0.78

```

Gambar 4.27 Hasil uji performa model

Skenario selanjutnya yaitu skenario kedua dilakukan dengan menyeleksi *term* dengan tidak memasukan *term* yang memiliki nilai *feature importances* kurang dari atau sama dengan nol pada *dataset* yang akan dijadikan model klasifikasi. Sama seperti sebelumnya, uji coba pertama dalam penerapan *transfer learning feature importance* skenario ini adalah dengan menggunakan model klasifikasi dengan *dataset* Amazon dan menerapkannya pada *dataset* Yelp yang akan dijadikan model klasifikasi. Gambar 4.28 menunjukkan langkah untuk menyeleksi *term* pada *dataset* Yelp berdasarkan informasi *feature importances* dari *dataset* Amazon. Proses tersebut akan *looping* tiap *term* dalam *vocabulary dataset* Yelp dan mengecek *term* tersebut jika terdapat pada *vocabulary* Amazon dan bukan merupakan *term* yang memiliki nilai *feature importance* lebih dari nol, maka tidak dimasukan ke dalam *list vocabulary* yang baru. Setelah inisialisasi *vocabulary* yang baru, maka proses selanjutnya adalah sama seperti skenario sebelumnya yaitu mulai dari membuat *dictionary* baru, hingga menguji performa model klasifikasi yang baru.

```
Vocabulary_Seleksi_Yelp = []
iterator = 0
length = len(Vocabulary_Yelp_Train)

while iterator < length :
    # jika interseksi dengan amazon dan bukan fitur importance maka vocab tidak dimasukan
    if (Vocabulary_Yelp_Train[iterator] in Vocabulary_Amazon_Train) and (
        Vocabulary_Yelp_Train[iterator] not in Vocabulary_Importance_Amazon_Train) :
        iterator = iterator + 10
        continue

    Vocabulary_Seleksi_Yelp.append(Vocabulary_Yelp_Train[iterator])
    iterator = iterator + 1
```

Gambar 4.28 Inisialisasi model klasifikasi sebelum diterapkan seleksi *term*



Gambar 4.29 menunjukkan hasil dari *dictionary term* pada data *train* untuk membangun model klasifikasi dari dataset Yelp sebelum dan setelah diberlakukan seleksi *term*. Hal ini menunjukkan bahwa terdapat sebanyak 4050 *term* diseleksi dan tidak dipergunakan. Gambar 4.30 menunjukkan perbandingan waktu yang diperlukan untuk inialisasi model dengan menggunakan *dataset* Yelp sebelum diseleksi dan sesudah diseleksi. Gambar 4.31 merupakan hasil uji coba performa model klasifikasi yang baru.

pd.DataFrame([WordDictionaryCount_Yelp_Train[0],WordDictionaryCount_Yelp_Train[1]])																				
(i,)	(, d)	(d, o)	(o, n)	(n, t)	(t,)	(, k)	(k, n)	(n, o)	(o, w)	...	(w, m, o)	(i, n, x)	(d, s, z)	(s, z, i)	(r, f, f)	(n, w,)	(s, n, q)	(u, b, g)	(b, g, u)	(y, c, r)
0	4	11	6	13	7	19	1	1	5	1	...	0	0	0	0	0	0	0	0	0
1	10	7	6	10	13	21	1	1	1	1	...	0	0	0	0	0	0	0	0	0
2 rows × 8197 columns																				
pd.DataFrame([WordDictionary_feature_importance_Train[0],WordDictionary_feature_importance_Train[1]])																				
(i,)	(, d)	(d, o)	(o, n)	(n, t)	(t,)	(, k)	(k, n)	(n, o)	(o, w)	...	(u, a, w)	(s, f, m)	(f, m, n)	(d, s, z)	(s, z, i)	(r, f, f)	(s, n, q)	(u, b, g)	(b, g, u)	(y, c, r)
0	4	11	6	13	7	19	1	1	5	1	...	0	0	0	0	0	0	0	0	0
1	10	7	6	10	13	21	1	1	1	1	...	0	0	0	0	0	0	0	0	0
2 rows × 4147 columns																				

Gambar 4.29 Hasil pemilihan *term* dengan seleksi *feature importance*

<pre> Waktu_Training = time() RF_Classifier_TF_FI = RandomForestClassifier(     max_depth= 5, n_estimators = 800, random_state=42,     bootstrap = False, min_samples_split = 5, min_samples_leaf = 1, max_features = 'auto') RF_Classifier_TF_FI.fit(TFIDF_Feature_importance_Train, Label_Yelp_Train) print(f"\nWaktu Training: {round(time()-Waktu_Training, 3)}s") </pre>	
<p>Waktu Training: 21.471s</p>	
<pre> Waktu_Training = time() RF_Classifier_Yelp = RandomForestClassifier(max_depth= 5, n_estimators = 800, random_state=42,     bootstrap = False, min_samples_split = 5, min_samples_leaf = 1, max_features = ' RF_Classifier_Yelp.fit(Vectorizer_Yelp_Train, Label_Yelp_Train) print(f"\nWaktu Training Yelp: {round(time()-Waktu_Training, 3)}s") </pre>	
<p>Waktu Training Yelp: 26.543s</p>	

Gambar 4.30 Inialisasi model klasifikasi sebelum dan setelah diterapkan seleksi *term*

```

Waktu_Predict_Train = time()
Skor_Train_Seleksi_Yelp = RF_Classifier_TF_FI.score(TFIDF_Feature_importance_Train, Label_Yelp_Train)
print(f"waktu prediksi (train): {round(time()-Waktu_Predict_Train, 3)}s")

Waktu_Predict_Test = time()
Skor_Test_Seleksi_Yelp = RF_Classifier_TF_FI.score(TFIDF_Feature_importance_Test, Label_Yelp_Test)
print(f"waktu prediksi (test): {round(time()-Waktu_Predict_Test, 3)}s")

print("\nSkor Random Forest Train Interseksi : {}".format(Skor_Train_Seleksi_Yelp))
print("Skor Random Forest Test Interseksi : {}".format(Skor_Test_Seleksi_Yelp))
print("-----\n\n")

RFClassifier_predict = RF_Classifier_TF_FI.predict(TFIDF_Feature_importance_Train)
Confusion_matrix = confusion_matrix(Label_Yelp_Train, RFClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_Yelp_Train, RFClassifier_predict))

print("-----\n\n")
RFClassifier_predict = RF_Classifier_TF_FI.predict(TFIDF_Feature_importance_Test)
Confusion_matrix = confusion_matrix(Label_Yelp_Test, RFClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_Yelp_Test, RFClassifier_predict))

waktu prediksi (train): 1.546s
waktu prediksi (test): 0.516s

Skor Random Forest Train Interseksi : 0.9072
Skor Random Forest Test Interseksi : 0.8608
-----

[[1678 197]
 [ 151 1724]]
      precision    recall  f1-score   support

         0       0.92      0.89      0.91      1875
         1       0.90      0.92      0.91      1875

 accuracy          0.91
 macro avg          0.91
weighted avg          0.91

-----

[[544 81]
 [ 93 532]]
      precision    recall  f1-score   support

         0       0.85      0.87      0.86      625
         1       0.87      0.85      0.86      625

 accuracy          0.86
 macro avg          0.86
weighted avg          0.86

```

Gambar 4.31 Hasil uji performa model

Sama seperti skenario sebelumnya, uji coba kedua dalam penerapan *transfer learning feature importance* skenario ini adalah dengan menggunakan informasi model klasifikasi dengan *dataset* Amazon dan menerapkannya pada *dataset* IMDB yang akan dijadikan model klasifikasi. Gambar 4.32 menunjukkan hasil dari *dictionary term* pada data *train* untuk membangun model klasifikasi dari dataset IMDB sebelum dan setelah diberlakukan pemilihan *term* secara seleksi. Gambar 4.33 menunjukkan perbandingan waktu yang diperlukan untuk inisialisasi model dengan menggunakan *dataset* IMDB sebelum diseleksi dan sesudah diseleksi. Gambar 4.34 merupakan hasil uji coba performa model klasifikasi yang baru.

pd.DataFrame([WordDictionaryCount_IMDB_Train[0],WordDictionaryCount_IMDB_Train[1]])	
(s,t) (t,o) (o,r) (r,y) (y,) (.o) (o,f) (f,) (.a) (a,) ... (k,b,l) (d,m,y) (p,s,n) (g,p,i) (.l,w) (w,r,) (t,t,t) (u,a,x) (h,b,y) (w,o,f)	
0	7 5 6 2 10 9 5 5 12 9 ... 0 0 0 0 0 0 0 0 0
1	25 19 47 5 33 65 25 29 81 28 ... 0 0 0 0 0 0 0 0 0
2 rows x 9284 columns	
pd.DataFrame([WordDictionary_feature_importance_Train[0],WordDictionary_feature_importance_Train[1]])	
(s,t) (t,o) (o,r) (r,y) (y,) (.o) (o,f) (f,) (.a) (a,) ... (t,h,k) (h,k,n) (q,u,t) (h,s,u) (p,s,n) (g,p,i) (.l,w) (w,r,) (u,a,x) (h,b,y)	
0	7 5 6 2 10 9 5 5 12 9 ... 0 0 0 0 0 0 0 0 0
1	25 19 47 5 33 65 25 29 81 28 ... 0 0 0 0 0 0 0 0 0
2 rows x 4721 columns	

Gambar 4.32 Hasil pemilihan *term* dengan seleksi *feature importance*

```

Waktu_Training = time()
RF_Classifier_IMDB = RandomForestClassifier(max_depth= 5,
n_estimators = 800, random_state=42,
bootstrap = False, min_samples_split = 5, min_samples_leaf = 1, max_features = 'auto')
RF_Classifier_IMDB.fit(Vectorizer_IMDB_Train, Label_IMDB_Train)
print(f"\nWaktu Training IMDB: {round(time()-Waktu_Training, 3)}s")

Waktu Training IMDB: 31.129s

Waktu_Training = time()
RF_Classifier_TF_FI = RandomForestClassifier(
max_depth= 5, n_estimators = 800, random_state=42,
bootstrap = False, min_samples_split = 5, min_samples_leaf = 1, max_features = 'auto')
RF_Classifier_TF_FI.fit(TFIDF_Feature_importance_Train, Label_IMDB_Train)
print(f"\nWaktu Training: {round(time()-Waktu_Training, 3)}s")

Waktu Training: 27.458s

```

Gambar 4.33 Inisialisasi model klasifikasi sebelum dan setelah diterapkan seleksi *term*

```

Waktu_Predict_Train = time()
Skor_Train_Seleksi_IMDB = RF_Classifier_TF_FI.score(TFIDF_Feature_importance_Train, Label_IMDB_Train)
print(f"waktu prediksi (train): {round(time()-Waktu_Predict_Train, 3)}s")

Waktu_Predict_Test = time()
Skor_Test_Seleksi_IMDB = RF_Classifier_TF_FI.score(TFIDF_Feature_importance_Test, Label_IMDB_Test)
print(f"waktu prediksi (test): {round(time()-Waktu_Predict_Test, 3)}s")

print("\nSkor Random Forest Train Interseksi : {}".format(Skor_Train_Seleksi_IMDB))
print("Skor Random Forest Test Interseksi : {}".format(Skor_Test_Seleksi_IMDB))
print("-----\n\n")

RFCClassifier_predict = RF_Classifier_TF_FI.predict(TFIDF_Feature_importance_Train)
Confusion_matrix = confusion_matrix(Label_IMDB_Train, RFCClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_IMDB_Train, RFCClassifier_predict))

print("-----\n\n")
RFCClassifier_predict = RF_Classifier_TF_FI.predict(TFIDF_Feature_importance_Test)
Confusion_matrix = confusion_matrix(Label_IMDB_Test, RFCClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_IMDB_Test, RFCClassifier_predict))

waktu prediksi (train): 1.82s
waktu prediksi (test): 0.597s

Skor Random Forest Train Interseksi : 0.9109333333333334
Skor Random Forest Test Interseksi : 0.7784
-----

[[1676 199]
 [ 135 1740]]
      precision    recall  f1-score   support

         0       0.93      0.89      0.91      1875
         1       0.90      0.93      0.91      1875

 accuracy          0.91
 macro avg          0.91
weighted avg          0.91

-----

[[470 155]
 [122 503]]
      precision    recall  f1-score   support

         0       0.79      0.75      0.77      625
         1       0.76      0.80      0.78      625

 accuracy          0.78
 macro avg          0.78
weighted avg          0.78

```

Gambar 4.34 Hasil uji performa model

### 4.3.2 Uji Coba Transfer Learning Nilai IDF

Pada Uji coba *transfer learning* nilai IDF hal yang dilakukan adalah melakukan penambahan nilai frekuensi dokumen yang memiliki suatu *term x* pada *dataset* yang akan menjadi model klasifikasi jika *term x* tersebut juga terdapat pada *dataset* yang telah di-*training* menjadi model klasifikasi. Gambar 4.35 menunjukkan sebuah fungsi untuk menghitung kembali nilai IDF pada suatu *data train*. Berbeda dengan perhitungan IDF sebelumnya (Gambar 4.9) pada fungsi tersebut, jumlah dokumen (N) pada *dataset* tersebut akan ditambahkan dengan jumlah dokumen pada *dataset* yang telah di-*training* menjadi model klasifikasi. Fungsi tersebut juga akan melakukan pengecekan apakah sebuah *term x* berinterseksi antar *dataset*, jika ya, maka nilai frekuensi dokumen kemunculan *term x* akan ditambahkan dengan nilai yang terdapat pada *dataset* yang telah di-*training*.

```
def RecomputeIDF(WordDictAmazon, WordDictYelp, additional_term_value, Vocabulary_Intersection):
    idfDict = {}
    number_of_document_with_term_t_in_it = {}
    N = len(WordDictYelp) + len(WordDictAmazon)
    idfDict = dict.fromkeys(WordDictYelp[0].keys(), 0)
    for doc in WordDictYelp:
        for word, val in doc.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        if word in Vocabulary_Intersection:
            val = val + additional_term_value[word]
            idfDict[word] = np.log((1+N) / (1+val)) + 1

    return idfDict

New_IDF_Yelp = RecomputeIDF(WordDictionaryCount_Amazon_Train,
                             WordDictionaryCount_Yelp_Train,
                             number_of_document_with_term_t_Amazon_Train,
                             Vocabulary_Intersection)
```

Gambar 4.35 Fungsi inialisasi ulang nilai IDF

Setelah selesai menghitung kembali nilai IDF, langkah selanjutnya sama seperti pada skenario sebelumnya, yaitu dengan menghitung kembali nilai TF-IDF sampai dengan menguji kembali model klasifikasi baru yang telah dibuat. Gambar 4.36 merupakan proses inialisasi model klasifikasi baru yang dibangun dengan menggunakan data *train* Yelp dengan *transfer* learning informasi IDF dari data *train* Amazon dan uji coba performa model tersebut, sedangkan Gambar 4.37 dibangun dengan menggunakan data *train* IMDB sebanyak 1000 baris data dengan *transfer* learning informasi IDF model yang dibangun dengan data *train* Amazon sebanyak 5000 baris data.

```
Waktu_Training = time()
New_RF_Classifier_Yelp = RandomForestClassifier(max_depth= 5, n_estimators = 800, random_state=42,
                                                bootstrap = False, min_samples_split = 5, min_samples_leaf = 1,
                                                max_features = 'auto')
New_RF_Classifier_Yelp.fit(New_TFIDF_Train, Label_Yelp_Train)
print(f"\nWaktu Training: {round(time()-Waktu_Training, 3)}s")

Waktu Training: 26.129s
```

Gambar 4.36 Uji coba model klasifikasi model Yelp

```

Waktu_Predict_Train = time()
Skor_Train_Yelp = New_RF_Classifier_Yelp.score(New_TFIDF_Train, Label_Yelp_Train)
print(f"waktu prediksi (train): {round(time()-Waktu_Predict_Train, 3)}s")

Waktu_Predict_Test = time()
Skor_Test_Yelp = New_RF_Classifier_Yelp.score(New_TFIDF_Test, Label_Yelp_Test)
print(f"waktu prediksi (test): {round(time()-Waktu_Predict_Test, 3)}s")

print("\nSkor Random Forest Train : {}".format(Skor_Train_Yelp))
print("Skor Random Forest Test : {}".format(Skor_Test_Yelp))
print("-----\n\n")

RFClassifier_predict = New_RF_Classifier_Yelp.predict(New_TFIDF_Train)
Confusion_matrix = confusion_matrix(Label_Yelp_Train, RFClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_Yelp_Train, RFClassifier_predict))

print("-----\n\n")
RFClassifier_predict = New_RF_Classifier_Yelp.predict(New_TFIDF_Test)
Confusion_matrix = confusion_matrix(Label_Yelp_Test, RFClassifier_predict)
print(Confusion_matrix)
print(metrics.classification_report(Label_Yelp_Test, RFClassifier_predict))

waktu prediksi (train): 2.61s
waktu prediksi (test): 0.868s

Skor Random Forest Train : 0.9050666666666667
Skor Random Forest Test : 0.8616
-----

[[1652 223]
 [ 133 1742]]
      precision    recall  f1-score   support

         0       0.93      0.88      0.90      1875
         1       0.89      0.93      0.91      1875

 accuracy          0.91
 macro avg          0.91
weighted avg          0.91

-----

[[540 85]
 [ 88 537]]
      precision    recall  f1-score   support

         0       0.86      0.86      0.86      625
         1       0.86      0.86      0.86      625

 accuracy          0.86
 macro avg          0.86
weighted avg          0.86

```

Gambar 4.37 Hasil Uji coba model klasifikasi model Yelp

### 4.3 Evaluasi Hasil Uji Coba

Hasil dari kedua jenis uji coba *transfer learning (feature importance* dan nilai IDF) dengan masing-masing skenario adalah sebagai berikut :

Tabel 4.2 Hasil Uji Coba penerapan *transfer learning feature importance*

Skenario	Waktu Training (sebelum / sesudah)	Akurasi (sebelum / setelah penerapan)	Label	Precision (sebelum / setelah penerapan)	Recall (sebelum / setelah penerapan)	F1 (sebelum / setelah penerapan)
Interseksi Amazon & Yelp	29, 272 / 21, 217 Detik	0.8624 / 0.8648	positif	0.86 / 0.87	0.86/0.86	0.86/0.86
			negatif	0.86 / 0.86	0.86/0.87	0.86/0.87
Interseksi Amazon & IMDB	31, 174 / 26, 664 Detik	0.7816 / 0.7704	positif	0.77/0.76	0.81/0.79	0.79/0.78
			negatif	0.80/0.78	0.75/0.75	0.78/0.76
Seleksi Amazon & Yelp	25, 727 / 21,14 Detik	0.8624 / 0.8608	positif	0.86/0.87	0.86/0.85	0.86/0.86
			negatif	0.86/0.85	0.86/0.87	0.86/0.86
Seleksi Amazon & IMDB	30, 614 / 26, 154 Detik	0.7816/ 0.7784	positif	0.77/ 0.76	0.81/0.80	0.79/0.78
			negatif	0.80/0.79	0.75/0.75	0.78/0.77

Tabel 4.2 Hasil Uji Coba penerapan *transfer learning* nilai IDF

Skenario	Jumlah data <i>train</i>	Akurasi (sebelum / setelah penerapan)	Label	Precision (sebelum / setelah penerapan)	Recall (sebelum / setelah penerapan)	F1 (sebelum / setelah penerapan)
dataset Amazon & IMDB	200	0.6672/ 0.6696	positif	0.68 / 0.68	0.63/0.64	0.66/0.66
			negatif	0.66 / 0.66	0.70/0.70	0.68/0.68
dataset Amazon & IMDB	1000	0.7382 / 0.7448	positif	0.73/0.75	0.75/0.75	0.74/0.75
			negatif	0.74/0.74	0.73/0.74	0.74/0.74



Tabel 4.2 Hasil Uji Coba penerapan *transfer learning* nilai IDF (lanjutan)

dataset Amazon & Yelp	200	0.8248/ 0.8296	positif	0.82 / 0.84	0.83/0.82	0.83/0.83
			negatif	0.83 / 0.82	0.82/0.84	0.82 / 0.83
dataset Amazon & Yelp	1000	0.8544 / 0.8512	positif	0.85/0.85	0.87/0.86	0.86/0.85
			negatif	0.86/0.86	0.84/0.84	0.85/0.85

Dari hasil uji performa penerapan *transfer learning feature importances* yang ditunjukkan oleh tabel 4.1, pada skenario pengambilan data secara interseksi antara data pada model yang telah di-*training* yang mengembalikan nilai *feature importance* untuk digunakan oleh *dataset* selanjutnya berdampak positif terhadap perbedaan waktu untuk melakukan *training* pada suatu *dataset*. Hal ini disebabkan karena pengurangan *term* atau fitur yang signifikan pada *dataset* yang akan di-*training*. Pada pengeambilan *term* yang interseksi antara *term dataset* Amazon yang memiliki *feature importance* lebih dari nol dengan *term* dari *dataset* Yelp yang semula memiliki *term* sebanyak 8197 *term*, namun setelah diberlakukan seleksi *term* menjadi hanya 2822 fitur, yang berdampak pada penggunaan waktu *training* data dari yang semula 29, 272 detik menjadi 21, 217 detik. Demikian juga pada skenario uji coba interseksi antara *term* Amazon yang memiliki *feature importance* lebih besar dari nol dengan *term* pada *dataset* IMDB yang semula memiliki *term* sebanyak 9284 *term*, menjadi 2834 *term* yang digunakan untuk *training* data. Waktu untuk *training* data tersebut yang semua adalah 31, 174 detik, berkurang menjadi 26, 664 detik. Selain pengurangan waktu *training*, pada skenario pengambilan *term* secara interseksi ini tidak mengurangi akurasi dan performa pada

model klasifikasi tersebut. Pada uji coba penerapan interseksi antara *term* Amazon dengan Yelp akurasi yang semula 0.8624 menjadi 0.8648 dan performa *precision*, *recall*, dan f1-score untuk setiap label positif dan negatif tidak terlalu berdampak, paling besar hanya berbeda 0,02 saja.

Pada skenario pengambilan data secara seleksi *term* pada *dataset* yang akan di-training dengan model yang telah di-*training* yang mengembalikan nilai *feature importance* juga berdampak positif terhadap perbedaan waktu untuk melakukan *training* pada suatu *dataset*. Berbeda dengan uji coba dengan hanya mengambil *term* yang interseksi, pada uji coba ini dengan menyeleksi *term* pada *dataset* yang akan di-*training* cenderung menghasilkan *list term* yang lebih banyak dibandingkan pengambilan secara *interseksi*. Hasilnya pada uji coba menggunakan data *feature importances* yang diterapkan pada *dataset* Yelp yang sebelumnya menggunakan sebanyak 8197 *term* untuk *training*, menjadi menggunakan 4147 *term* untuk *training* setelah diseleksi memerlukan waktu *training* selama 25, 727 detik, menjadi 21, 14 detik dengan akurasi dan performa *precision*, *recall*, dan f1-score yang cenderung tidak jauh berbeda dan hampir serupa.

Hasil uji penerapan *transfer learning* nilai IDF dilihat pada tabel 4.2. Uji coba tersebut bertujuan untuk mengubah nilai IDF pada *term dataset* yang akan di-*training* yang berinterseksi dengan *term* pada *dataset* yang telah di-*training* sebelumnya. Nilai IDF pada sebuah *term* berubah jika jumlah frekuensi dokumen atau data yang terdapat *term* tersebut lebih besar pada *dataset* yang telah di-*training* dibandingkan dengan yang terdapat pada *dataset* yang akan di-*training*. Pada percobaan penerapan dengan *dataset* model Amazon pada *dataset* Yelp yang akan

di-*training* dengan uji performa yang dilakukan, mendapatkan akurasi dan performa yang cukup stabil dan hampir sama dengan sebelum diterapkannya yang berarti model dapat mempertahankan akurasi meskipun diubah nilai IDF tersebut pada sebagian *term*.

Dari hasil skenario yang telah dilakukan, dengan menggunakan *transfer learning* mendapatkan hal yang positif seperti *resource* untuk melakukan n-grams sebagai input TF-IDF tidak perlu lagi dilakukan dan performa yang dihasilkan tidak mengalami penurunan. Hal ini dapat dimanfaatkan pada menggunakan *dataset* baru dengan jumlah *term* atau fitur yang cukup banyak dan menggunakan *resource* yang besar.

## BAB V

### SIMPULAN DAN SARAN

#### 5.1 Simpulan

Berdasarkan hasil dari implementasi dan hasil uji coba yang telah dilakukan, simpulan dari penelitian adalah sebagai berikut.

1. Implementasi metode *transfer learning* untuk klasifikasi sentimen dengan Random Forest Classifier menggunakan TF-IDF telah selesai dilakukan dengan menggunakan *dataset review* perusahaan Amazon, Yelp, dan IMDB
2. Hasil dari uji coba *transfer learning* dengan eliminasi *term* berdasarkan *list feature importance* menunjukkan bahwa memiliki hasil yang positif yakni dengan waktu *training* data yang berkurang dan akurasi yang stabil pada model yang diseleksi *termnya* tersebut.
3. Hasil dari uji coba *transfer learning* dengan nilai IDF menunjukkan hasil yang positif. Meskipun mengubah nilai IDF pada suatu *term* dengan tujuan untuk memperbaiki nilai kepentingan dari *term* tersebut dengan acuan dokumen yang lebih besar, akurasi dan performa yang didapatkan cenderung stabil.

## 5.2 Saran

Berdasarkan penelitian yang dilakukan, terdapat beberapa saran untuk pengembangan lanjutan, yaitu sebagai berikut.

1. Penelitian selanjutnya diharapkan dapat membangun model klasifikasi yang lebih baik dengan *hyperparameter* yang lebih bervariasi menggunakan skala *GridsearchCv* yang lebih besar, dan melakukan *preprocessing text* yang lebih bervariasi.
2. Mencoba merapkan *transfer learning* dengan metode yang berbeda selain *feature importances* dan nilai IDF yaitu dengan informasi lainnya yang didapatkan dari model yang telah di-*training* sebelumnya.
3. Mencoba menggunakan *classifier* lainnya selain Random Forest Classifier, dan mencoba *word embedding* disamping TF-IDF, seperti *Word2Vec* dan yang lainnya.
4. Menggunakan model yang memiliki *base* performa dan akurasi yang lebih baik untuk diambil informasi dari model tersebut dan menerapkan *transfer learning*.

## DAFTAR PUSTAKA

- Al Amrani, Y., Lazaar, M. and El Kadiri, K.E. (2018). Random Forest and Support Vector Machine based Hybrid Approach to Sentiment Analysis. *Procedia Computer Science*, [online] 127, halaman 511. Tersedia di: <https://www.sciencedirect.com/science/article/pii/S1877050918301625> [Diakses 2 Feb. 2020].
- Dhawangkhar, M. and Riksakomara, E. (2017). Prediksi Intensitas Hujan Kota Surabaya dengan Matlab menggunakan Teknik Random Forest dan CART (Studi Kasus Kota Surabaya). *Jurnal Teknik ITS*, 6(1).
- Fauzi, M.A. (2018). Random Forest Approach for Sentiment Analysis in Indonesian Language. *Indonesian Journal of Electrical Engineering and Computer Science*, 12(1), halaman.46.
- Indhiarta, W (2017). Penggunaan N-Gram Pada Analisa Sentimen Pemilihan Kepala Daerah Jakarta Menggunakan Algoritma Naïve Bayes. Diploma thesis, Universitas Muhammadiyah Surakarta.
- Liu, B. (2012). Sentiment Analysis and Opinion Mining. *Synthesis Lectures on Human Language Technologies*, 5(1), halaman.1–167
- Lisangan, E. (2013). Implementasi n-gram Technique dalam Deteksi Plagiarisme pada Tugas Mahasiswa. *Jurnal Tematika*, [online] 1 (2) . Tersedia di: [https://www.researchgate.net/publication/265686891\\_Implementasi\\_n-gram\\_Technique\\_dalam\\_Deteksi\\_Plagiarisme\\_pada\\_Tugas\\_Mahasiswa](https://www.researchgate.net/publication/265686891_Implementasi_n-gram_Technique_dalam_Deteksi_Plagiarisme_pada_Tugas_Mahasiswa) [Diakses 6 Feb. 2020].
- Nurjannah, M., Hamdani., dan Astuti, I. (2013). Penerapan Algoritma Term Frequency-Inverse Document Frequency (TF-IDF) Untuk Text Mining. *Jurnal Informatika Mulawarman*, [online] Volume(8), halaman 110. Tersedia di: <http://ejournals.unmul.ac.id/index.php/JIM/article/view/113/pdf> [Diakses 9 Feb. 2020].
- Oktanisa, I. dan Supianto, A.A. (2018). Perbandingan Teknik Klasifikasi Dalam Data Mining Untuk Bank Direct Marketing. *Jurnal Teknologi Informasi dan Ilmu Komputer*, [online] 5(5), halaman 567. Tersedia di: <http://jtiik.ub.ac.id/index.php/jtiik/article/view/958/xml> [Diakses 24 Mar. 2019].
- Prabowo, Y.D., Marselino, T.L. and Suryawiguna, M. (2019). Pembentukan Vector Space Model Bahasa Indonesia Menggunakan Metode Word to Vector. *Jurnal Buana Informatika*, 10(1), halaman.29.

Tresnawati, Y (2018). Analisis Sentimen Pada Twitter Menggunakan Pendekatan Agglomerative Hierarchical Clustering. Tersedia di: [https://repository.usd.ac.id/11493/3/135314018\\_full.pdf](https://repository.usd.ac.id/11493/3/135314018_full.pdf) [Diakses 9 Feb. 2020].

Weiss, K., Khoshgoftaar, T., dan Wang, D (2016). A Survey of Transfer Learning Journal of Big Data, 3(9), Tersedia di: <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-016-0043-6> [Diakses 5 Feb. 2019].

Yanuar, A (2018). Random Forest. [online] Menara Ilmu Machine Learning UGM. Tersedia di: <http://machinelearning.mipa.ugm.ac.id/2018/07/28/random-forest/>. [Diakses 7 Feb. 2020].

Zulfa, I. and Winarko, E. (2017). Sentimen Analisis Tweet Berbahasa Indonesia Dengan Deep Belief Network. IJCCS (Indonesian Journal of Computing and Cybernetics Systems), 11(2), halaman.187

Zhang, D., Liu, J., Heng, W., Ren, K. and Song, J. (2018). Transfer Learning with Convolutional Neural Networks for SAR Ship Recognition. *IOP Conference Series: Materials Science and Engineering*, 322(1)