BAB IV
HASIL DAN PEMBAHASAN

Pada bab ini dijelaskan mengenai hasil dan pembahasan dari penelitian klasifikasi dokumen teks. Hasil dan pembahasan ini meliputi kode program dari proses klasifikasi dokumen medis. Pada subbab-subbab yang menampilkan kode program, dijelaskan juga hasil akurasi yang diperoleh dari proses klasifikasi pada data testing.

4.1 Pengumpulan Data

Pengambilan data dilakukan dengan mengumpulkan data artikel kesehatan pada *website* kesehatan yang ada di Indonesia dan secara otomatis artikel kesehatan tersebut berbahasa indonesia. Data yang digunakan pada percobaan tugas akhir ini terdiri dari 700 data *training* yang dibagi menjadi dua jenis yaitu data berlabel dan data tidak berlabel, kemudian data tersebut digunakan untuk melakukan *training*. Selain itu, ada 100 data *testing* yang digunakan untuk melakukan uji validitas. Data yang digunakan terbagi menjadi 10 kategori kelas yang berkaitan dengan kesehatan/medis yaitu kesehatan bayi, diabetes, diet, jantung, kecantikan, kehamilan, kesehatan gigi dan mulut, kolesterol, kulit, mata. Data tersebut dikumpulkan secara manual dengan melakukan *copy paste*. Proses yang dilakukan dengan men-*copy* artikel kesehatan pada *website* dan kemudian di-*paste* pada sebuah dokumen yang sudah disiapkan. Dokumen yang disimpan adalah dokumen txt dengan *extension file* .txt.

Data berupa dokumen teks yang sudah dikumpulkan akan diuji coba sebanyak 4 kali yang mana dua jenis data yaitu data berlabel dan data tidak berlabel akan dipisah dengan folder yang berbeda. Dari 700 data training yang telah dikumpulkan, pengujian pertama diberikan jumlah data berlabel dengan porsi yang banyak yaitu 400 data berlabel dan 300 data tidak berlabel. Melanjutkan pengujian kedua, porsi data berlabel dikurangi 100 data yang mana 100 data hasil dari pengurangan data berlabel tersebut ditambahkan pada data tidak berlabel, dan menghasilkan porsi 300 data berlabel dan 400 data tidak berlabel. Hal yang sama dilakukan pada pengujian ketiga, porsi data berlabel dikurangi 100 yang kemudian data tersebut ditambahkan pada data tidak berlabel menghasilkan porsi 200 data berlabel dan 500 data tidak berlabel. Pengujian ketiga juga menerapkan hal yang sama menghasilkan porsi 100 data berlabel dan 600 data tidak berlabel. Pengujian diberhentikan hingga mencapai nilai perbandingan terkecil untuk data berlabel.

4.2 *Preprocessing*

Preprocessing adalah sebuah proses untuk membersihkan data yang mengandung *noise,* sehingga dapat menghasilkan data yang bersih dan dapat memberikan hasil yang baik pada proses selanjutnya. Berikut ini beberapa tahapan yang dilakukan selama proses *preprocessing.*

Sebelum mulai melakukan *preprocessing,* seluruh dokumen yang disimpan pada folder yang berbeda baik dokumen berlabel maupun dokumen tidak berlabel dengan ekstensi file .txt dibaca menggunakan modul yang sudah disediakan python. Membaca dokumen dilakukan untuk mempermudah proses-proses yang akan dilakukan selanjutnya. Sebelum membaca dokumen yang akan diproses, dilakukan import untuk modul yang akan digunakan untuk membaca data seperti pada Gambar 4.1.

```
1  import glob
2  import os
```

Gambar 4. 1 *Import* modul *glob* dan *os*

Setelah modul *glob* dan *os* berhasil dimasukkan, selanjutnya adalah membuat kode program yang mampu membaca dokumen berektensi txt pada folder yang berisi dokumen berlabel dan folder yang berisi dokumen tidak berlabel. Adapun kode program untuk membaca folder yang berisi dokumen berekstensi txt seperti yang dapat dilihat pada Gambar 4.2.

```
1  trainlabeled = glob.glob(os.path.join(os.getcwd(), "E:/kiki/Bismillah
2  ya Allah/TAfix/TrainingA/Labeled", "*.txt"))
3  trainunlabeled=glob.glob(os.path.join(os.getcwd(),
4  "E:/kiki/Bismillah ya Allah/TAfix/TrainingA/Unlabeled", "*.txt"))
```

Gambar 4. 2 Kode program membaca folder dokumen training berlabel dan tidak berlabel

Berdasarkan Gambar 4.2 menunjukan baris 1-2 adalah kode program untuk membaca seluruh dokumen berlabel dalam folder. Baris 3-4 adalah kode program untuk membaca seluruh dokumen tidak berlabel dalam folder.

Kemudian, setelah membaca dokumen berlabel dan dokumen tidak berlabel, proses membaca folder yang berisi dokumen txt juga dilakukan pada dokumen testing. Gambar 4.3 adalah kode program untuk membaca folder yang berisi dokumen berekstensi txt pada folder dokumen testing.

```
1  testing = glob.glob(os.path.join(os.getcwd(), "E:/kiki/Bismillah ya
2  Allah/TAfix/Testing", "*.txt"))
```

Gambar 4. 3 Kode program membaca folder dokumen testing

Setelah membaca data, maka data siap untuk dilakukan proses preprocessing. Berikut adalah proses preprocessing yang dilakukan.

a. Membersihkan tanda baca, angka, tanda *hashtag*, *white space*, karakter non ASCII

Proses ini dilakukan untuk menghilangkan simbol – simbol, tanda baca, spasi yang berlebihan dan karakter *non ASCII* dengan menggunakan *regex* yang modulnya sudah disediakan oleh *python.* Gambar 4.4 adalah kode program untuk *import* modul regex.

```
import re
```

Gambar 4. 4 *Import* modul *re*

Modul yang sudah diimpor tersebut dapat digunakan untuk membantu proses pengahapusan *noise*. Gambar 4.5 adalah proses pembersihan karakter yang fungsinya diberi nama *cleaning.*

```
1   def cleaning(text):
2       #remove non-ascii
3       text = unicodedata.normalize('NFKD', text).encode('ascii',
4   'ignore').decode('utf-8', 'ignore')
5      #remove URLs
6           text = re.sub(r'(?i)\b((?:https?://|www\d{0,3}[.]|
7      [a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+
8      |(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]+|(\([^\s()<>]+
9   \)))*\)|[^\s`!()\[\]{};:\'".,<>?«»“”‘’]))', ' ', text)
10      #remove punctuations
11      text = re.sub(r'[^\w]|_',' ',text)
12      #remove digit from string
13      text = re.sub("\S*\d\S*", " ", text).strip()
14      #remove digit or numbers
15      text = re.sub(r"\b\d+\b", "", text)
16      #Remove additional white spaces
17      #txt = re.sub(r"\s+", " ",text)
18      text = re.sub('[\s]+', ' ', text)
19      #return " ".join(text.split())
20      return text
```

Gambar 4. 5 Kode program *cleaning*

Berdasarkan Gambar 4.5 proses untuk menghilangkan *noise* dalam data melalui tahap-tahap seperti pada baris 3-4 adalah kode program yang digunakan untuk menghilangkan karakter non-ASCII, baris 6-9 adalah kode program yang digunakan untuk menghapus *URL*, baris 14 untuk menghapus tanda baca, baris 13 untuk menghapus angka dalam karakter *string*, baris 15 untuk menghapus angka atau nomor, dan baris 18 kode program untuk menghapus spasi yang berlebihan (*white spaces*).

b. Mengubah semua huruf menjadi huruf kecil

Mengubah semua huruf menjadi huruf kecil atau yang sering dikenal dengan *casefolding* dibutuhkan dalam mengkonversi keseluruhan teks dalam dokumen menjadi suatu bentuk standar (biasanya huruf kecil atau lowercase). Gambar 4.6 adalah kode program untuk melakukan *casefolding* yang ditunjukan pada baris 2.

```
1  def casefolding(text):
2      new_str = text.lower()
3      return new_str
```

Gambar 4. 6 Kode program melakukan *casefolding*

c. Mengubah kata berimbuhan menjadi kata dasar

Mengubah kata berimbuhan menjadi kata dasar atau yang sering disebut dengan *stemming* dilakukan untuk mengelompokan kata-kata lain yang memiliki kata dasar dan arti yang serupa. Proses *stemming* Bahasa Indonesia dapat menggunakan modul yang telah disediakan oleh python. Modul *stemming* Bahasa Indonesia yang digunakan adalah Sastrawi. Gambar 4.7 adalah kode program untuk *import* modul *stemming* Bahasa Indonesia menggunakan sastrawi.

```
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
```

Gambar 4. 7 *Import* modul *stemming* sastrawi

Modul yang sudah di *import* dapat digunakan untuk mengubah kata menjadi kata dasar atau *stemming*. Gambar 4.8 adalah kode program untuk melakukan stemming Bahasa Indonesia menggunakan modul sastrawi yang sudah di import sebelumnya. Untuk melakukan stemming bahas Indonesia, kode program pada Gambar 4.8 yang berperan adalah baris 2-3.

```
1  def stemmingIndo(text):
2      factory = StemmerFactory()
3      stemmer = factory.create_stemmer()
4      return stemmer.stem(text)
```

Gambar 4. 8 Kode program *stemming* bahasa indonesia

d. Menghilangkan *stopwords*

Stopwords adalah kumpulan kata yang tidak atau kurang penting sehingga kata tersebut harus dihilangkan. Sebelum menghilangkan *stopwords*, terlebih dahulu harus mengimport modul nltk dimana dokumen stop words ini disimpan dengan nama '*stopwordnew*'. Gambar 4.5 adalah kode program untuk *import* modul *nltk* yang digunakan untuk menghapus *stopword*.

```
from nltk.corpus import stopwords
```

Gambar 4. 9 *Import nltk* untuk *stopword*

Untuk menghilangkan *stopwords* dibuat kamus yang berisi kumpulan *stopwords* dari korpus yang dimiliki. Gambar 4.10 berikut ini adalah kode program untuk menghilangkan *stopwords.*

```
1  def removeStopword(text):
2      stop_words = set(stopwords.words('stopwordnew'))
3      word_tokens = word_tokenize(text)
4      filtered_sentence = [w for w in word_tokens if not w in stop_words]
5      return ' '.join(filtered_sentence)
```

Gambar 4. 10 Kode program untuk menghilangkan *stopwords*

Untuk menghapus *stopword*, berdasarkan Gambar 4.10 kode program yang digunakan untuk menghapus *stopword* adalah baris 2-4. Pada baris 2, kata yang dihapus berdasarkan kata yang ada pada *corpus* dengan format nama '*stopwordnew*'.

e. Menyimpan hasil *preprocessing* ke dalam dokumen baru

Setelah proses *preprocessing* dilakukan maka hasil *preprocessing* akan disimpan ke dalam dokumen yang baru dengan ekstensi yang sama yaitu txt. Hasil *preprocessing* akan disimpan sebagai dokumen baru baik itu pada dokumen berlabel dan dokumen tidak berlabel. Gambar 4.11 adalah kode program untuk melakukan *preprocessing* dan menyimpan dokumen berlabel ke dalam dokumen baru.

```
1     for file_path in trainlabeled:
2         # a.txt => a-out.txt, b.txt => b-out.txt, etc.
3         with  open(file_path.replace('.txt',  '-out.txt'),  'w')  as
4  outfile:
5             # read a line at time from input file and
6             # write a line at a time to output file
7             with open(file_path) as infile:
8                 for text in infile:
9                  print(preprocessingWithStopWord(text),    end='    ',
10 file=outfile)
```

Gambar 4. 11 Kode program untuk *preprocessing* dan menyimpan dokumen berlabel ke dalam dokumen baru

Selain melakukan preprocessing dan menyimpan ke dalam dokumen baru untuk dokumen berlabel. Preprocessing dan menyimpan ke dalam dokumen baru juga dilakukan pada dokumen yang tidak berlabel. Gambar 4.12 adalah kode program untuk melakukan *preprocessing* dan menyimpan dokumen tidak berlabel ke dalam dokumen baru.

```
for file_path in trainunlabeled:
    # a.txt => a-out.txt, b.txt => b-out.txt, etc.
    with open(file_path.replace('.txt', '-out.txt'), 'w') as outfile:
        # read a line at time from input file and
        # write a line at a time to output file
        with open(file_path) as infile:
            for text in infile:
                print(preprocessing(text), end=' ', file=outfile)
```

Gambar 4. 12 Kode program untuk *preprocessing* dan menyimpan dokumen tidak berlabel ke dalam dokumen baru

Hal yang sama juga dilakukan pada dokumen *testing* yaitu melakukan *preprocessing* dan menyimpan dokumen hasil *preprocessing* tersebut ke dalam dokumen baru. Gambar 4.13 adalah kode program untuk melakukan *preprocessing* dan menyimpan dokumen *testing* ke dalam dokumen baru.

```
for file_path in testing:
    # a.txt => a-out.txt, b.txt => b-out.txt, etc.
    with open(file_path.replace('.txt', '-out.txt'), 'w') as outfile:
        # read a line at time from input file and
        # write a line at a time to output file
        with open(file_path) as infile:
            for text in infile:
                print(preprocessing(text), end=' ', file=outfile)
```

Gambar 4. 13 Kode program untuk *preprocessing* dan menyimpan dokumen *testing* ke dalam dokumen baru

Setelah proses *preprocessing*, maka secara otomatis data hasil preprocessing akan di simpan ke dalam dokumen baru. Untuk melalui proses training, dokumen hasil preprocessing akan digabungkan dan disimpan menjadi satu dokumen baru berkekstensi csv. Kode program untuk melakukan proses *converting* dokumen *training* yang berlabel ditunjukan pada Gambar 4.14.

```
1   header = ["Text", "Category"]
2
3   with open('Labeled.csv', 'w', newline='') as f_output:
4       csv_output = csv.writer(f_output)
5       csv_output.writerow(header)
6       for x in range(1, 401):
7           filepath     =     os.path.normpath(r'E:\kiki\Bismillah     ya
8   Allah\TAfix\TrainingA\Labeled\{}-out.txt'.format(x))
9           with open(filepath, 'r', newline='') as f_text:
10              csv_text     =     csv.reader(f_text,     delimiter=',',
11  skipinitialspace=True)
12              csv_output.writerow(tuple(next(csv_text)))
```

Gambar 4. 14 *Converting* dokumen *training* berlabel

Berdasarkan Gambar 4.14 proses untuk menggabungkan dokumen bersih hasil preprocessing dan mengubahnya menjadi satu dokumen baru berekstensi csv ditunjukan pada kode program baris 3-12. Mengabungkan dan menyimpan data menjadi dokumen baru juga dilakukan pada dokumen *training* yang tidak berlabel yang mana folder dari dokumen *training* untuk data berlabel dan data tidak berabel dipisah. Gambar 4.15 adalah *converting* dokumen *training* pada data yang tidak berlabel.

```
1   header = ["Text"]
2
3   with open('Unlabeled.csv', 'w', newline='') as f_output:
4       csv_output = csv.writer(f_output)
5       csv_output.writerow(header)
6       for x in range(1, 301):
7           filepath     =     os.path.normpath(r'E:\kiki\Bismillah     ya
8   Allah\TAfix\TrainingA\Unlabeled\{}-out.txt'.format(x))
9           with open(filepath, 'r', newline='') as f_text:
10              csv_text = csv.reader(f_text, skipinitialspace=True)
11              csv_output.writerow(tuple(next(csv_text)))
```

Gambar 4. 15 *Converting* dokumen *training* tidak berlabel

Berdasarkan Gambar 4.15 proses untuk menggabungkan dokumen bersih hasil preprocessing dan mengubahnya menjadi satu dokumen baru berekstensi csv ditunjukan pada kode program baris 3-11. Selain itu, proses ini juga diimplementasikan pada dokumen *testing*, tidak hanya

pada dokumen yang akan digunakan untuk melakukan *training*. Maka, gambar 4.16 menunjukan kode program untuk melakukan *convert* dokumen *testing*.

```
1     header = ["Text", "Category"]
2
3     with open('Testing.csv', 'w', newline='') as f_output:
4         csv_output = csv.writer(f_output)
5         csv_output.writerow(header)
6         for x in range(1, 101):
7             filepath = os.path.normpath(r'E:\kiki\Bismillah ya
8 Allah\TAfix\TrainingA\Testing\{}-out.txt'.format(x))
9             with open(filepath, 'r', newline='') as f_text:
10                csv_text = csv.reader(f_text, delimiter=',',
11 skipinitialspace=True)
12                csv_output.writerow(tuple(next(csv_text)))
```

Gambar 4. 16 *Converting* dokumen *testing*

Baris yang menunjukan kode program untuk menggabungkan data hasil preprocessing dan menyimpannya ke dalam dokumen baru berkektensi csv seperti pada Gambar 4.16 ditunjukan pada baris 3-12.

4.3 Training menggunakan *Semi-Supervised Learning*

Klasifikasi adalah proses untuk melakukan pembagian atau kategorisasi sesuatu kedalam kelas-kelas tertentu. Pada penelitian ini yang menjadi objek klasifikasi adalah dokumen medis yang terdapat pada website kesehatan yang ada di Indonesia dan berbahasa Indonesia. Penelitian ini menggunaan teknik *Semi-Supervised Learning* dengan menerapkan metode *Pseudo Labeling* dan menggunakan algoritma *Multinomial Naive Bayes* untuk melakukan klasifikasi pada dokumen medis. Proses klasifikasi yang dilakukan menggunakan *labeled data* (data berlabel) dan *unlabeled data* (data tidak berlabel) secara bersamaan sebagai data training. Pada data berlabel pemberian label dilakukan dengan memberi label pada data *training* dengan cara *self-labeling* pada dokumen medis. Sedangkan data tidak berlabel nantinya akan di klasifikasi dan menghasilkan *pseudo-label*.

Data *training* yang digunakan pada penelitian ini adalah data artikel kesehatan dari *website* kesehatan di Indonesia. Data yang digunakan berjumlah 700 data yang dibagi menjadi dua jenis yaitu data berlabel dan data tidak berlabel. Masing-masing jenis data yang ada

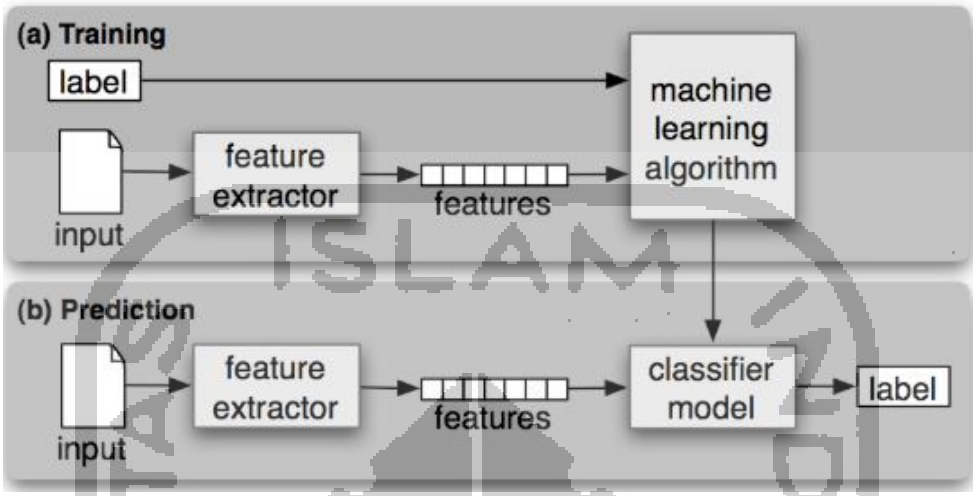
kemudian dilakukan *preprocessing* dan hasil dari *preprocessing* tersebut di simpan kedalam dokumen baru. Dokumen baru hasil dari *preprocessing* kemudian di gabungkan menjadi satu dokumen *csv* yang siap untuk di proses. Pada proses *training* dilakukan dengan membagi proses pengujian ke dalam empat tahap. Pengujian pertama menggunakan data acak untuk 400 data berlabel dan 300 data tidak berlabel (model 4:3) sebagai data training dan data acak untuk 100 data berlabel sebagai data testing, pengujian kedua menggunakan data acak untuk 300 data berlabel dan 400 data tidak berlabel (model 3:4) sebagai data training dan data acak untuk 100 data berlabel sebagai data testing, pengujian ketiga menggunakan data acak untuk 200 data berlabel dan 500 data tidak berlabel (model 2:5) sebagai data training dan data acak untuk 100 data berlabel sebagai data testing, dan yang terkahir dilakukan adalah pengujian keempat yaitu menggunakan data acak untuk 100 data berlabel dan 600 data tidak berlabel (model 1:6) sebagai data training dan data acak untuk 100 data berlabel sebagai data testing. Proses klasifikasi yang dilakukan pada data *training* dengan masing-masing porsi dokumen berlabel dan dokumen tidak berlabel ini terdapat 10 kategori yang berbeda pada bidang kesehatan yaitu kesehatan bayi, diabetes, diet, jantung, kecantikan, kehamilan, kesehatan gigi dan mulut, kolesterol, kulit, mata. Berikut ini ditunjukan pada Tabel 4.1 adalah percobaan yang dilakukan pada penelitian ini.

Tabel 4. 1 Pengujian yang akan dilakukan

	Data berlabel	**Data tak berlabel**	**Data kombinasi**
Pengujian 1	400	300	400:300/4:3
Pengujian 2	300	400	300:400/3:4
Pengujian 3	200	500	200:500/2:5
Pengujian 4	100	600	100:600/1:6

Model *classifier* pada masing-masing model dibangun berdasarkan 3 jenis data yang berbeda yaitu data berlabel, data tidak berlabel, dan data kombinasi dari data berlabel dan tidak berlabel. Pengujian pada setiap model akan dilakukan percobaan berdasarkan 3 jenis data tersebut. Percobaan akan dilakukan pada data testing untuk melakukan uji validitas terhadap hasil akurasi yang diperoleh dari proses klasifikasi dokumen medis. Data *testing* pada penelitain ini berjumlah 100 data acak berlabel dengan 10 kategori yang berbeda dan dibagi menjadi 10 dokumen berlabel yang berbeda. Data tersebut kemudian dilakukan *preprocessing* dan hasil dari *preprocessing* tersebut di simpan kedalam dokumen baru dan di simpan menjadi

satu dokumen *csv*. Data inilah yang akan dipakai sebagai data testing terhadap model *classifier* yang dibangun. Alur proses klasifikasi dapat dilihat pada gambar 4.17.



Gambar 4. 17 Alur proses klasifikasi

Berdasarkan gambar 4.17 maka langkah yang dilakukan dalam proses klasifikasi adalah sebagai berikut :

4.3.1 Membaca Data

Pada penelitian ini input data yang digunakan adalah kumpulan data dari data berlabel dan data tidak berlabel. Percobaan ini dilakukan adalah untuk menghasilkan model classifier dari masing-masing percobaan pada 3 jneis data yang dilakukan.

Hal pertama yang dilakukan adalah membaca data *input* dalam hal ini adalah data *training*/data latih Sebelum membaca data yang harus dilakukan adalah *import* modul *pandas* yang sudah disediakan oleh *python* seperti pada Gambar 4.18.

```
import pandas as pd
```

Gambar 4. 18 *Import* modul *pandas*

Import modul pandas pada python digunakan agar dapat membaca dokumen yang akan dilakukan pada proses selanjutnya. Setelah import selesai dilakukan, proses untuk membaca data dapat dilakukan. Awal percobaan ini akan membaca data latih belabel. Langkah-langkah yang dilakukan adalah seperti seperti pada Gambar 4.19.

```
df = pd.read_csv('Labeled.csv')
```

Gambar 4. 19 Membaca data latih berlabel

4.3.2 *Training Data*

Untuk melakukan proses *training data*, data berlabel yang dibaca akan digunakan untuk menghitung nilai *tf-idf*. Training data awal dilakukan pada data berlabel yang mana dari data berlabel tersebut akan dibangun model *classifier* yang digunakan untuk melakukan prediksi pada data tidak berlabel. Hal tersebut dilakukan untuk menerapkan teknik *pseudo-labeling* pada seluruh percobaan. Setiap percobaan yang dilakukan akan di *training* dengan cara yang sama namun dengan jumlah porsi data yang berbeda seperti yang sudah dijelaskan pada Tabel 4.1. Pertama, yang dilakukan untuk melakukan training data adalah menghitung nilai *tf-idf*. Pada bahasa pemrograman *Python*, modul untuk melakukan *tf-idf* sudah disediakan. Untuk itu, modul tersebut harus di *import* terlebih dahulu. Kode program untuk mengimpor modul *tf-idf* dapat dilihat pada Gambar 4.20.

```
1  from sklearn.feature_extraction.text import CountVectorizer,
2  TfidfVectorizer, TfidfTransformer
```

Gambar 4. 20 *Import* modul *tf-idf*

Setelah modul untuk mengaplikasikan *tf-idf* berhasil diimpor, selanjutnya adalah melakukan *tf-idf vectorizer* hal ini dilakukan untuk mengumpulkan kata unik yang muncul dalam dokumen yang dimiliki. Sebelum dilakukan proses perhitungan *tf-idf* data teksnya akan di transformasi ke dalam vektor. Kode program untuk langkah di atas dapat dilihat pada Gambar 4.21.

```
1  count_vect = CountVectorizer()
2  X_train_counts = count_vect.fit_transform(df['Text'])
3  tfidf_transformer = TfidfTransformer()
4  X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

Gambar 4. 21 Transformasi *tf-idf*

Selanjutnya, langkah yang dilakukan adalah membangun model *classifier* dari data berlabel. Hal yang sama dilakukan pada setiap percobaan pada Tabel 4.1. Model klasifikasi yang digunakan pada penelitian ini adalah menggunakan *Multinomial Naive Bayes classifier*. Untuk

membangun model ini maka hal yang harus dilakukan pertama kali adalah dengan mengimpor modul yang dibutuhkan seperti yang dapat dilihat pada Gambar 4.22.

```
from sklearn.naive_bayes import MultinomialNB
```

Gambar 4. 22 *Import* modul *classifier*

Setelah impor modul dilakukan, selanjutnya adalah mulai menerapkan klasifikasi *Multinomial Naive Bayes* dengan menggunakan variabel vektor *tf-idf*, klasifikasi menggunakan *Multinomial Naive Bayes* dapat dilakukan, seperti pada gambar 4.23.

```
clf = MultinomialNB().fit(X_train_tfidf, df['Category'])
```

Gambar 4. 23 Model *Multinomial Naive Bayes*

4.3.3 Prediksi Data Tidak Berlabel

Data tidak berlabel pada setiap percobaan akan diproses menggunakan metode *pseudo-labeling*. Caranya adalah dengan menggunakan model *classifier* dari data berlabel yang telah dibangun sebelumnya, model *classifier* tersebut kemudian digunakan untuk melakukan prediksi pada setiap percobaan menggunakan data tidak berlabel. Hasil data tersebut kemudian disimpan sebagai *pseudo-label data* yang dianggap sebagai label sebenarnya dari data tersebut.

Pertama yang harus dilakukan untuk melakukan klasifikasi menggunakan metode *pseudo-labeling* adalah membaca data tidak berlabel menggunakan kode program seperti pada Gambar 4.24.

```
dfun = pd.read_csv('Unlabeled.csv')
```

Gambar 4. 24 Membaca data latih tidak berlabel

Menggunakan masing-masing model *classifier* yang dibangun pada setiap percobaan menggunakan data berlabel, prediksi dilakukan pada data tidak berlabel. Gambar 4.25 menunjukan kode program untuk melakukan prediksi terhadap data tidak berlabel dengan menggunakan model *classifier* data berlabel.

```
1  textunlabeled = dfun["Text"]#tolist()
2  unlabeledclass=clf.predict(count_vect.transform(textunlabeled))
3  lihat = unlabeledclass.tolist()
```

Gambar 4. 25 Kode program untuk melakukan prediksi

Berdasarkan Gambar 4.25 kode program pada baris 2 digunakan untuk melakukan prediksi pada tidak berlabel menggunakan model yang dibangun dengan data berlabel. Hasil dari klasifikasi data tidak berlabel menggunakan model *classifier* dari data berlabel yang dilakukan dapat dilihat pada Gambar 4.26, Gambar 4.27, Gambar 4.28, dan Gambar 4.29.

Category	
bayi	29
diabetes	31
diet	31
gigi dan mulut	30
jantung	29
kecantikan	12
kehamilan	27
kolesterol	35
kulit	45
mata	31

Gambar 4. 26 Hasil prediksi percobaan 1 pada data tidak berlabel

Category	
bayi	35
diabetes	37
diet	42
gigi dan mulut	40
jantung	43
kecantikan	14
kehamilan	42
kolesterol	44
kulit	62
mata	41

Gambar 4. 27 Hasil prediksi percobaan 2 pada data tidak berlabel

Category	
bayi	48
diabetes	47
diet	49
gigi dan mulut	50
jantung	51
kecantikan	20
kehamilan	50
kolesterol	64
kulit	70
mata	51

Gambar 4. 28 Hasil prediksi percobaan 3 pada data tidak berlabel

Category	
bayi	68
diabetes	85
diet	63
gigi dan mulut	70
jantung	66
kecantikan	49
kehamilan	69
kolesterol	79
kulit	78
mata	73

Gambar 4. 29 Hasil prediksi percobaan 4 pada tidak berlabel

Gambar 4.26, Gambar 4.27, Gambar 4.28, dan Gambar 4.29 adalah hasil prediksi pada data tidak berlabel yang dilakukan pada percobaan 1,2,3, dan 4. Hasil dari prediksi yang dilakukan disebut dengan *pseudo-label data* yang mana jika digunakan untuk membangun model dari kombinasi data, *pseudo-label* tersebut akan dianggap sebagai label sebenarnya dari data.

Proses selanjutnya digunakan untuk menyimpan hasil prediksi sebagai *pseudo-label* yang akan disimpan sebagai label dari dokumen tidak berlabel. Hasil tersebut dianggap sebagai label sebenarnya dari data. Pertama yang dilakukan dalam proses ini adalah dengan menyimpan hasil dari klasifikasi yang berupa kategori hasil prediksi yang sudah dilakukan sebelumnya. Gambar 4.30 adalah kode program untuk menyimpan kategori hasil klasifikasi ke dalam dokumen csv yang ditunjukan pada baris 1-2.

```
1  dflabel = pd.DataFrame(lihat, columns=["Category"])
2  dflabel.to_csv('label.csv', index=False)
3  dfcategory = pd.read_csv('label.csv')
```

Gambar 4. 30 Menyimpan kategori hasil prediksi

Selanjutnya adalah menggabungkan kategori hasil prediksi yang dianggap sebagai *pseudo-label* dengan dokumen teks dari data. Gambar 4.31 adalah kode program untuk menggabungkan *pseudo-label* dengan dokumen teks tidak berlabel yang mana dokumen teks tersebut adalah dokumen yang diklasifikasi sebelumnya.

```
1  C = pd.merge(left=dfun, right=dfcategory, how='outer',
2  left_index=True, right_index=True, suffixes=['_a', '_b'])
3  C.to_csv('Unlabeled-label.csv', index=False)
```

Gambar 4. 31 Menggabungkan label dan teks data

Baris 1-2 adalah kode program yang digunakan untuk menggabungkan *pseudo-label* dengan teks data. Kemudian, pada baris ke 3 kode program tersebut digunakan untuk menyimpan hasil penggabungan data ke dalam dokumen baru. Setelah menyimpan hasil klasifikasi data tidak berlabel kedalam satu dokumen. Langkah selanjutnya adalah membaca data tersebut yang digunakan untuk *training* data tidak berlabel. Gambar 4.29 adalah kode program untuk membaca dokumen hasil klasifikasi.

```
dftrainun = pd.read_csv('Unlabeled-label.csv')
```

Gambar 4. 32 Kode program untuk membaca dokumen hasil klasifikasi

Selanjutnya adalah menerapkan fitur *tf-idf* pada data yang sudah dibaca sebelumnya. Gambar 4.33 adalah kode program untuk menghitung *tf-idf* pada data tidak berlabel.

```
1  count_vecttrain = CountVectorizer()
2  X_train_countstrain=count_vecttrain.fit_transform(dftrainun['Text'])
3  tfidf_transformertrain = TfidfTransformer()
4  X_train_tfidftrain=tfidf_transformertrain.fit_transform(X_train_counts
5  train)
```

Gambar 4. 33 *tf-idf* pada data tidak berlabel

Setelah menghitung nilai *tf-idf*, langkah selanjutnya adalah membangun model *classifier* dengan kode program yang tinjunjukan pada gambar 4.31.

```
1  clftrain=MultinomialNB().fit(X_train_tfidftrain,
2  dftrainun['Category'])
```

Gambar 4. 34 Model *classifier* data tidak berlabel

4.3.4 Menggabungkan Data Berlabel & Data Tidak Berlabel

Teknik *Semi-Supervised Learning* adalah teknik yang memanfaatkan kombinasi dari kedua jenis label data baik itu data berlabel maupun data tidak berlabel. Untuk menerapkan teknik *Semi-Supervised Learning* data berlabel dan data tidak berlabel akan digabungkan menjadi satu dokumen baru. Langkah ini adalah langkah yang digunakan untuk membangun model *classifier* dari kedua jenis data training yaitu data kombinasi. Proses ini akan dilakukan pada setiap percobaan yang ada pada Tabel 4.1. Dari proses training data kombinasi ini model 4:3, model 3:4, model 2:5, dan model 1:6 akan dihasilkan.

Pertama yang dilakukan untuk membangun model *classifier* dari data kombinasi adalah membaca data. *Training* diawali dengan membaca kedua dokumen data berlabel dan data tidak berlabel/data kombinasi ditunjukan pada kode program Gambar 4.35.

```
1  reader = pd.read_csv('Labeled.csv')
2  reader1 = pd.read_csv('Unlabeled-label.csv')
```

Gambar 4. 35 Membaca data kombinasi

Berdasarkan Gambar 4.35 kode program pada baris 1 digunakan untuk membaca data berlabel sedangkan pada baris 2 adalah kode program yang digunakan untuk membaca data tidak berlabel yang sudah digabungkan dengan *pseudo-label*.

Selanjutnya adalah menggabungkan dokumen yang sudah dibaca sebelumnya yaitu data berlabel dan data tidak berlabel menjadi satu dokumen baru yang nantinya akan di training. Gambar 4.36 adalah kode program untuk menggabungkan dokumen dari data berlabel dan data tidak berlabel menjadi satu dokumen baru.

```
1  #combine all files in the list
2  combined_csv = pd.concat([reader, reader1], ignore_index=True)
3  combined_csv.to_csv(  "Combined.csv",  index=False,  encoding='utf-8-
4  sig')
```

Gambar 4. 36 Menggabungkan dokumen kombinasi

Baris 2 pada Gambar 4.36 adalah kode program yang digunakan untuk menggabungkan data berlabel dan data tidak berlabel. Kemudian, pada baris 3 data yang sudah digabungkan sebelumnya disimpan ke dalam satu dokumen baru. Setelah dokumen baru hasil dari penggabungan data kombinasi berhasil dilakukan, maka langkah selanjutnya adalah membaca dokumen gabungan tersebut agar dapat di *training* sesuai dengan kode program yang ditunjukan pada Gambar 4.37.

```
dfcombined = pd.read_csv("Combined.csv")
```

Gambar 4. 37 Membaca dokumen gabungan

Proses membaca dokumen dilakukan agar dapat diproses pada tahap-tahap selanjutnya, seperti menerapkan fitur *tf-idf* yang ditunjukan pada gambar 4.38.

```
1  count_vectun = CountVectorizer()
2  X_train_countsun=count_vectun.fit_transform(dfcombined['Text'])
3  tfidf_transformerun = TfidfTransformer()
4  X_train_tfidfun=tfidf_transformerun.fit_transform(X_train_countsun)
```

Gambar 4. 38 *tf-idf* pada dokumen data kombinasi

Setelah menghitung nilai *tf-idf* dari data kombinasi, selanjutnya adalah membangun model *classifier* dari data kombinasi pada setiap percobaan pada Tabel 4.1. Model *classifier* ini dibangun seperti yang ditunjukan pada gambar 4.39.

```
1  clfun = MultinomialNB().fit(X_train_tfidfun, dfcombined['Category'])
```

Gambar 4. 39 Model classifier dari data kombinasi

Seluruh proses untuk melakukan training menggunakan teknik *Semi-Supervised Learning* akan dilakukan pada setiap percobaan yang ditunjukan pada Tabel 4.1. Setiap percobaan yang membangun model *classifier* akan di uji cobakan pada tahap selanjutnya menggunakan data

testing. Model pada seluruh percobaan yang uji akan dilihat performanya berdasarkan hasil nilai akurasi yang ditunjukan pada proses pengujian.

4.4 Analisis dan evaluasi

Analisis dilakukan untuk mengetahui apakah model *classifier* yang sudah dibangun dengan menerapkan metode *Pseudo Labeling* menggunakan algoritma *Multinomial Naïve Bayes* mampu menghasilkan akurasi yang cukup baik dalam melakukan klasifikasi dokumen medis. Analisis dilakukan pada data *testing* berjumlah 100 data berlabel yang diberikan secara acak dan analisis juga menggunakan model *classifier* dari data *training* berlabel yang acak. Klasifikasi dilakukan untuk melihat apakah kategori dari data berlabel sesuai dengan hasil klasifikasi yang dilakukan dengan menerapkan model *classifier* menggunakan percobaan yang ditunjukan pada Tabel 4.1 yang dibangun sebelumnya. Kemudian setelah melakukan klasifikasi akan dilihat hasil dari proses tersebut dan akan dihitung berapa banyak kategori prediksi yang tidak sesuai dengan kategori asli dokumen. Semakin sedikit nilai kategori prediksi yang tidak sesuai dengan kategori asli maka semakin tinggi pula akurasi yang diperoleh. Untuk melakukan analisis berdasarkan data testing beberapa langkah-langkah akan dilakukan sebagai berikut :

4.4.1 Membaca data

Hal pertama yang dilakukan untuk melakukan uji validitas pada data testing adalah dengan membaca data. Gambar 4.40 adalah kode program untuk membaca data testing.

```
dftest = pd.read_csv("Testing.csv")
```

Gambar 4. 40 Membaca data testing

Setelah membaca data, proses yang akan dilakukan adalah membuat data yang terdiri dari teks dan kategori ini ke dalam sebuah list. Hal ini dilakukan karena data teks adalah data yang nantinya akan di prediksi kategorinya. Gambar 4.41 adalah kode program untuk menyimpan data testing yang berisi data teks dan kategori di simpan kedalam list.

```
1  testingtext = dftest["Text"]#tolist()
2  testingcategory = dftest["Category"]#tolist()
```

Gambar 4. 41 Menyimpan data ke dalam *list*

Pada Gambar 4.41, data yang disimpan ke dalam list yang berupa text data adalalah baris 1, sedangkan data yang berupa kategori dari teks ditunjukan pada baris 2.

4.4.2 Analisis model *classifier* data berlabel

Pengujian pada data *testing* digunakan untuk melakukan analisis terhadap model *classifier* yang dibangun. Analisis dilakukan dengan cara memprediksi seluruh data *testing* menggunakan model *classifier* data berlabel. Prediksi dilakukan pada data *testing* yang menggunakan model *classifier* berdasarkan beberapa percobaan pada Tabel 4.1 dengan data berlabel. Berikut Gambar 4.42 menunjukan kode program untuk melakukan prediksi pada data *testing* menggunakan model *classifier* data berlabel.

```
testlabeled = clf.predict(count_vect.transform(testingtext))
```

Gambar 4. 42 Prediksi data testing menggunakan model *classifier* berlabel

Dari seluruh hasil prediksi yang diperoleh bisa menjelaskan berapa data yang sesuai dan berapa data yang tidak sesuai berdasarkan *true label* dan *predicted label*. Untuk melihat akurasi dari hasil yang sudah diperoleh maka Gambar 4.43 menunjukan kode program untuk *import* modul yang bisa digunakan untuk menghitung akurasi dari hasil klasifikasi yang sudah dilakukan sebelumnya.

```
import numpy
```

Gambar 4. 43 *Import* modul *numpy*

Untuk mengetahui akurasi dari hasil klasifikasi yang diperoleh dari uji validitas yang dilakukan Gambar 4.44 menunjukan kode program untuk menghitung akurasi yang dihasilkan dari hasil klasifikasi.

```
1  print(str('%.3f'%(numpy.mean(testlabeled == testingcategory)*100)) +
2  "%")
```

Gambar 4. 44 Menghitung akurasi dari model *classifier* data berlabel

a. Model 4:3

Dari gambar 4.44 hasil percobaan yang dilakukan pada porsi 4:3 menggunakan 400 data acak berlabel untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 4:3 pada lima kali percobaan menggunakan 400 data acak berlabel seperti pada Tabel 4.2.

Tabel 4. 2 Akurasi model 4:3 data berlabel

Percobaan	**Akurasi**
Percobaan 1	91.00%
Percobaan 2	92.00%
Percobaan 3	90.00%
Percobaan 4	91.00%
Percobaan 5	89.00%

b. Model 3:4

Dari gambar 4.44 hasil percobaan yang dilakukan pada porsi 3:4 menggunakan 300 data acak berlabel untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 3:4 pada lima kali percobaan menggunakan 300 data acak berlabel seperti pada Tabel 4.3.

Tabel 4. 3 Akurasi model 3:4 data berlabel

Percobaan	**Akurasi**
Percobaan 1	87.00%
Percobaan 2	89.00%
Percobaan 3	92.00%
Percobaan 4	91.00%
Percobaan 5	93.00%

c. Model 2:5

Dari gambar 4.44 hasil percobaan yang dilakukan pada porsi 2:5 menggunakan 200 data acak berlabel untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 2:5 pada lima kali percobaan menggunakan 200 data acak berlabel seperti pada Tabel 4.4.

Tabel 4. 4 Akurasi model 2:5 data berlabel

Percobaan	**Akurasi**
Percobaan 1	86.00%
Percobaan 2	93.00%
Percobaan 3	93.00%
Percobaan 4	92.00%
Percobaan 5	85.00%

d. Model 1:6

Dari gambar 4.44 hasil percobaan yang dilakukan pada porsi 1:6 menggunakan 100 data acak berlabel untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 1:6 pada lima kali percobaan menggunakan 100 data acak berlabel seperti pada Tabel 4.5.

Tabel 4. 5 Akurasi model 1:6 data berlabel

Percobaan	**Akurasi**
Percobaan 1	91.00%
Percobaan 2	92.00%
Percobaan 3	80.00%
Percobaan 4	89.00%
Percobaan 5	89.00%

Berdasarkan model 4:3, model 3:4, model 2:5, dan model 1:6 menghasilkan akurasi *max* dari model *classifier* pada seluruh percobaan menggunakan data acak berlabel. Hasil nilai akurasi

yang tertinggi pada masing-masing percobaan menggunakan data berlabel ditampilkan pada Tabel 4.6.

Tabel 4. 6 Hasil akurasi data *testing* dari model *classifier* data berlabel

Percobaan (Data)	Akurasi Tertinggi	Akurasi Terendah	Rata-rata Akurasi
Percobaan 1 (400)	92.00%	89.00%	90.60%
Percobaan 2 (300)	93.00%	87.00%	90.40%
Percobaan 3 (200)	93.00%	85.00%	89.80%
Percobaan 4 (100)	92.00%	80.00%	88.20%

Akurasi yang diperoleh pada Tabel 4.2 adalah hasil dari klasifikasi yang dilakukan menggunakan model *classifier* yang dibangun dengan metode *Multinomial Naïve Bayes* yang memanfaatkan seluruh data berlabel yang di acak. Model *classifier* yang sudah dibangun kemudian digunakan untuk melakukan prediksi terhadap 100 data acak *testing* yang memiliki label, dimana label tersebut dianggap sebabagi *true label* dari data. setelah melakukan prediksi, hasil prediksi akan dianggap sebagai *predict label* dari data. Langkah selanjutnya adalah akan dilihat apakah *predict label* sama dengan *true label*. Kesesuaian antara *true label* dan *predict label* akan dihitung sebagai nilai akurasi menggunakan kode program seperti pada Gambar 4.44 dan menghasilkan nilai akurasi seperti yang ditunjukan pada Tabel 4.6.

4.4.3 Analisis model *classifier* data tidak berlabel

Pengujian pada data *testing* digunakan untuk melakukan analisis terhadap model *classifier* yang dibangun. Analisis dilakukan dengan cara memprediksi seluruh data *testing* menggunakan model *classifier* data tidak berlabel. Prediksi dilakukan pada data *testing* yang menggunakan model *classifier* berdasarkan beberapa percobaan pada Tabel 4.1 dengan data berlabel. Berikut Gambar 4.45 menunjukan kode program untuk melakukan prediksi pada data *testing* menggunakan model *classifier* data tidak berlabel.

```
testunlabeled = clftrain.predict(count_vecttrain.transform(testingtext))
```

Gambar 4. 45 Prediksi data *testing* menggunakan model *classifier* tidak berlabel

Dari seluruh hasil yang diperoleh bisa menjelaskan berapa data yang sesuai dan berapa data yang tidak sesuai berdasarkan *true label* dan *predicted label*. Untuk mengetahui akurasi

dari hasil klasifikasi yang diperoleh dari uji validitas yang dilakukan pada seluruh percobaan pada Tabel 4.1 menggunakan data tidak berlabel, Gambar 4.46 menunjukan kode program untuk menghitung akurasi yang dihasilkan dari hasil klasifikasi.

```
1  print(str('%.3f'%(numpy.mean(testunlabeled == testingcategory)*100))
2  + "%")
```

Gambar 4. 46 Menghitung akurasi dari model *classifier* data tidak berlabel

a. Model 4:3

Dari gambar 4.46 hasil percobaan yang dilakukan pada porsi 4:3 menggunakan 300 data tidak berlabel untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 4:3 pada lima kali percobaan menggunakan 300 data tidakberlabel seperti pada Tabel 4.7.

Tabel 4. 7 Akurasi model 4:3 data tidak berlabel

Percobaan	**Akurasi**
Percobaan 1	82.00%
Percobaan 2	85.00%
Percobaan 3	80.00%
Percobaan 4	82.00%
Percobaan 5	79.00%

b. Model 3:4

Dari gambar 4.46 hasil percobaan yang dilakukan pada porsi 3:4 menggunakan 400 data tidak berlabel untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 3:4 pada lima kali percobaan menggunakan 400 data tidak berlabel seperti pada Tabel 4.8.

Tabel 4. 8 Akurasi model 3:4 data tidak berlabel

Percobaan	Akurasi
Percobaan 1	80.00%
Percobaan 2	80.00%
Percobaan 3	84.00%
Percobaan 4	80.00%
Percobaan 5	84.00%

c. Model 2:5

Dari gambar 4.46 hasil percobaan yang dilakukan pada porsi 2:5 menggunakan 500 data tidak berlabel untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 2:5 pada lima kali percobaan menggunakan 500 data tidak berlabel seperti pada Tabel 4.9.

Tabel 4. 9 Akurasi model 2:5 data tidak berlabel

Percobaan	Akurasi
Percobaan 1	80.00%
Percobaan 2	80.00%
Percobaan 3	78.00%
Percobaan 4	82.00%
Percobaan 5	79.00%

d. Model 1:6

Dari gambar 4.46 hasil percobaan yang dilakukan pada porsi 1:6 menggunakan 600 data tidak berlabel untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 1:6 pada lima kali percobaan menggunakan 600 data tidak berlabel seperti pada Tabel 4.10.

Tabel 4. 10 Akurasi model 1:6 data tidak berlabel

Percobaan	Akurasi
Percobaan 1	82.00%
Percobaan 2	81.00%
Percobaan 3	72.00%
Percobaan 4	83.00%
Percobaan 5	81.00%

Berdasarkan Gambar 4.46, menghasilkan akurasi dari model *classifier* pada seluruh percobaan menggunakan data tidak berlabel. Hasil perhitungan akurasi pada masing-masing percobaan menggunakan data tidak berlabel ditampilkan pada Tabel 4.11.

Tabel 4. 11 Hasil akurasi data *testing* dari model *classifier* data tidak berlabel

Percobaan (Data)	Akurasi Tertinggi	Akurasi Terendah	Rata-rata Akurasi
Percobaan 1 (300)	85.00%	79.00%	81.60%
Percobaan 2 (400)	84.00%	80.00%	81.60%
Percobaan 3 (500)	82.00%	78.00%	79.80%
Percobaan 4 (600)	83.00%	72.00%	79.90%

Akurasi yang diperoleh pada Tabel 4.11 adalah hasil dari klasifikasi yang dilakukan menggunakan model *classifier* yang dibangun dengan metode *Pseudo-labeling* yang memanfaatkan seluruh data yang tidak berlabel. Data tidak berlabel dapat digunakan untuk melakukan klasifikasi karena menggunakan *pseudo label data* untuk membangun sebuah model *classifier*. Model *classifier* yang sudah dibangun kemudian digunakan untuk melakukan prediksi terhadap 100 data acak *testing* yang memiliki label, dimana label tersebut dianggap sebabagi *true label* dari data. Setelah melakukan prediksi, hasil prediksi akan dianggap sebagai *predict label* dari data. Langkah selanjutnya adalah akan dilihat apakah *predict label* sama dengan *true label*. Kesesuaian antara *true label* dan *predict label* akan dihitung sebagai nilai akurasi menggunakan kode program seperti pada Gambar 4.46 dan menghasilkan nilai akurasi seperti yang ditunjukan pada Tabel 4.11.

4.4.4 Analisis model *classifier* data kombinasi

Pengujian pada data *testing* digunakan untuk melakukan analisis terhadap model *classifier* yang dibangun. Analisis dilakukan dengan cara memprediksi seluruh data *testing* menggunakan model *classifier* data kombinasi. Prediksi dilakukan pada data *testing* yang menggunakan model *classifier* berdasarkan beberapa percobaan pada Tabel 4.1 dengan data kombinasi. Berikut Gambar 4.67 menunjukan kode program untuk melakukan prediksi pada data *testing* menggunakan model *classifier* data kombinasi.

```
test = clfun.predict(count_vectun.transform(testingtext))
```

Gambar 4. 47 Prediksi data *testing* menggunakan model *classifier* kombinasi data

Untuk mengetahui akurasi dari hasil klasifikasi yang diperoleh dari uji validitas yang dilakukan sebelumnya, gambar 4.48 menunjukan kode program untuk menghitung akurasi yang dihasilkan dari hasil klasifikasi menggunakan data kombinasi.

```
print(str('%.3f'%(numpy.mean(test == testingcategory)*100)) + "%")
```

Gambar 4. 48 Menghitung akurasi dari model *classifier* data kombinasi

a. Model 4:3

Dari gambar 4.48 hasil percobaan yang dilakukan pada porsi 4:3 menggunakan 4:3 data kombinasi untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 4:3 pada lima kali percobaan menggunakan 4:3 data kombinasi seperti pada Tabel 4.12.

Tabel 4. 12 Akurasi model 4:3 data kombinasi

Percobaan	**Akurasi**
Percobaan 1	84.00%
Percobaan 2	84.00%
Percobaan 3	81.00%
Percobaan 4	85.00%
Percobaan 5	82.00%

b. Model 3:4

Dari gambar 4.48 hasil percobaan yang dilakukan pada porsi 3:4 menggunakan 3:4 data kombinasi untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 3:4 pada lima kali percobaan menggunakan 3:4 data kombinasi seperti pada Tabel 4.13.

Tabel 4. 13 Akurasi model 3:4 data kombinasi

Percobaan	**Akurasi**
Percobaan 1	82.00%
Percobaan 2	81.00%
Percobaan 3	85.00%
Percobaan 4	82.00%
Percobaan 5	87.00%

c. Model 2:5

Dari gambar 4.48 hasil percobaan yang dilakukan pada porsi 2:5 menggunakan 2:5 data kombinasi untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 2:5 pada lima kali percobaan menggunakan 2:5 data kombinasi seperti pada Tabel 4.14.

Tabel 4. 14 Akurasi model 2:5 data kombinasi

Percobaan	**Akurasi**
Percobaan 1	82.00%
Percobaan 2	80.00%
Percobaan 3	79.00%
Percobaan 4	83.00%
Percobaan 5	81.00%

d. Model 1:6

Dari gambar 4.48 hasil percobaan yang dilakukan pada porsi 1:6 menggunakan 1:6 data kombinasi untuk membangun model classifier yang berbeda pada setiap percobaannya. Percobaan ini dilakukan lima kali pada 100 data testing berlabel yang di acak. Setiap percobaan yang dihasilkan akan digunakan hasil nilai akurasi yang tertinggi. Berikut ini hasil akurasi yang diperoleh model 1:6 pada lima kali percobaan menggunakan 1:6 data kombinasi seperti pada Tabel 4.15.

Tabel 4. 15 Akurasi model 1:6 data kombinasi

Percobaan	**Akurasi**
Percobaan 1	82.00%
Percobaan 2	83.00%
Percobaan 3	74.00%
Percobaan 4	85.00%
Percobaan 5	81.00%

Berdasarkan gambar 4.48, menghasilkan akurasi dari model *classifier* data kombinasi pada masing-masing percobaan yang dilakukan. Hasil perhitungan akurasi pada masing-masing pengujian ditampilkan pada Tabel 4.16.

Tabel 4. 16 Hasil akurasi data *testing* dari model *classifier* data kombinasi

Percobaan (Data)	**Akurasi Tertinggi**	**Akurasi Terendah**	**Rata-rata Akurasi**
Percobaan 1 (4:3)	85.00%	81.00%	83.20%
Percobaan 2 (3:4)	87.00%	81.00%	83.40%
Percobaan 3 (2:5)	82.00%	79.00%	81.00%
Percobaan 4 (1:6)	85.00%	74.00%	81.00%

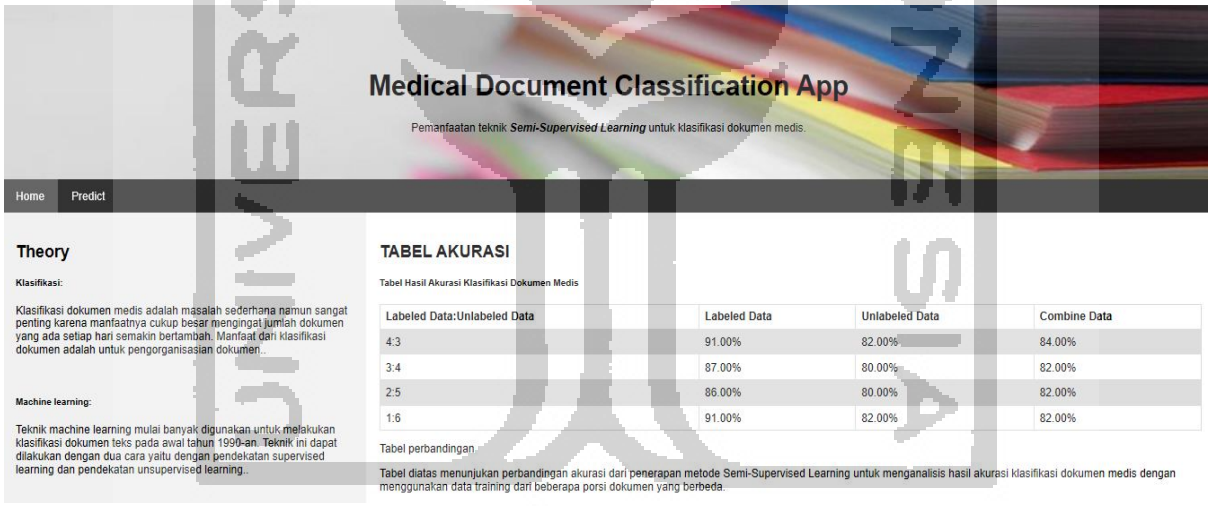
Hasil uji validitas menggunakan model *classifier* pada data tidak berlabel dan model *classifier* pada kombinasi data dengan meguji *validation set* sebagai data *testing* menunjukan hasil yang cukup baik di keempat pembagian porsi dokumen berlabel dan tidak berlabel. Hal ini menunjukan bahwa model yang dibangun cukup baik untuk diimplementasikan.

Hal tersebut ditunjukan dengan nilai akurasi pada model *classifier* menggunakan data kombinasi mencapai nilai akurasi yang cukup tinggi yaitu diatas 80%. Nilai akurasi yang dihasilkan dari klasifikasi menggunakan model yang dibangun ini juga menunjukan bahwa

model yang dibangun masih bisa digunakan untuk melakukan klasifikasi dokumen medis yang lainnya. Oleh karna itu, model *classifier* yang dibangun mampu melakukan klasifikasi pada *testing data* dengan cukup baik yang ditunjukan dengan nilai akurasi yang diperoleh dari hasil klasifikasi.

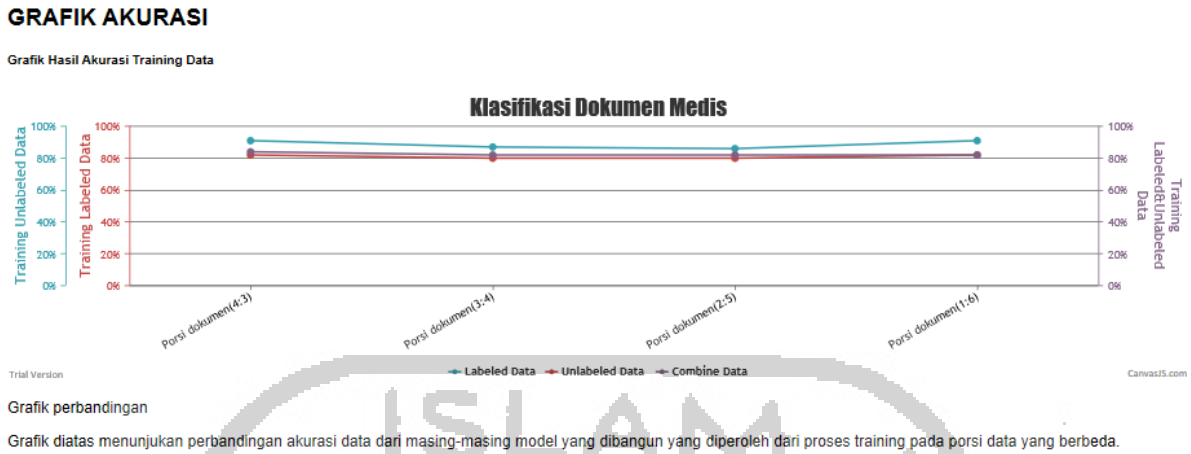
4.5 Implementasi Aplikasi

Website dibangun untuk melakukan klasifikasi teks dokumen medis menggunakan model 4:3, model 3:4, model 2:5, dan model 1:6. Pada penelitian ini, pengembangan *website* menggunakan *flask* sebagai *framework* untuk bahasa pemrograman *python*. Halaman awal *website* atau halaman *home*, pada website ditampilkan beberapa informasi yang berkaitan dengan model *classifier* yang dibangun sebelumnya. Informasi tersebut berupa tabel dan grafik hasil akurasi model *classifier* pada *validation set*. Gambar 4.49 menunjukan tabel hasil akurasi model *classifier* yang telah dilakukan sebelumnya.



Gambar 4. 49 Tabel hasil akurasi klasifikasi dokumen medis

Selain menyajikan informasi hasil akurasi ke dalam tabel, informasi juga disajikan ke dalam grafik yang menampilkan hasil akurasi klasifikasi dokumen medis pada gambar 4.50.

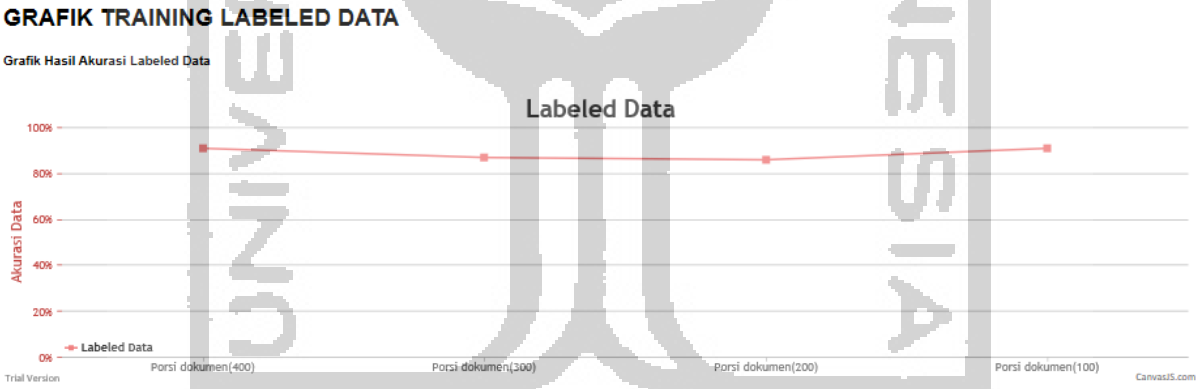


Grafik perbandingan

Grafik diatas menunjukan perbandingan akurasi data dari masing-masing model yang dibangun yang diperoleh dari proses training pada porsi data yang berbeda.

Gambar 4. 50 Grafik hasil akurasi klasifkasi dokumen medis

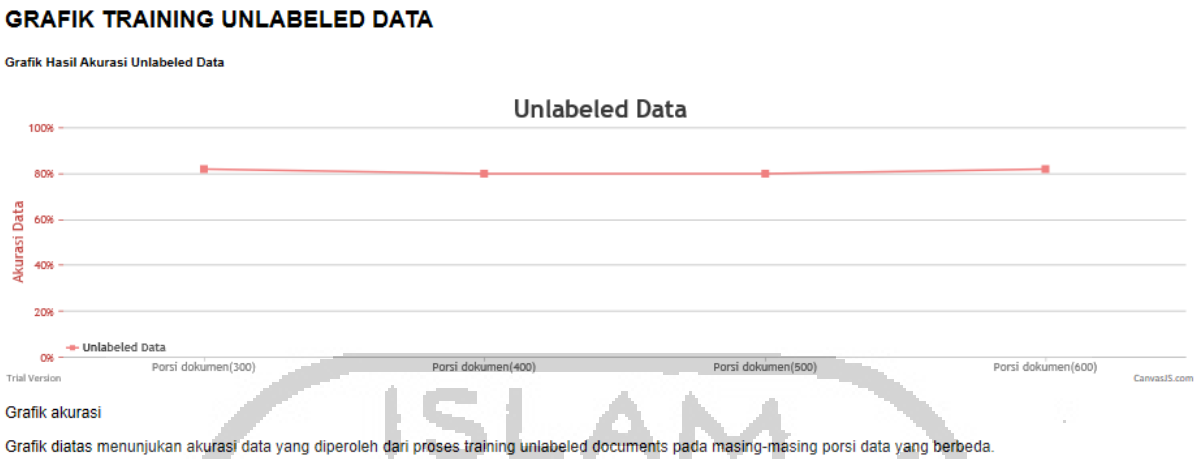
Informasi yang disajikan juga menampilkan grafik yang menunjukan hasil akurasi klasifikasi dokumen medis berdasarkan jenis data yaitu data berlabel, data tidak berlabel, dan data kombinasi dari keduanya. Grafik masing-masing jenis data ditampilkan secara terpisah yaitu pada Gambar 4.51, Gambar 4.52, dan Gambar 4.53.



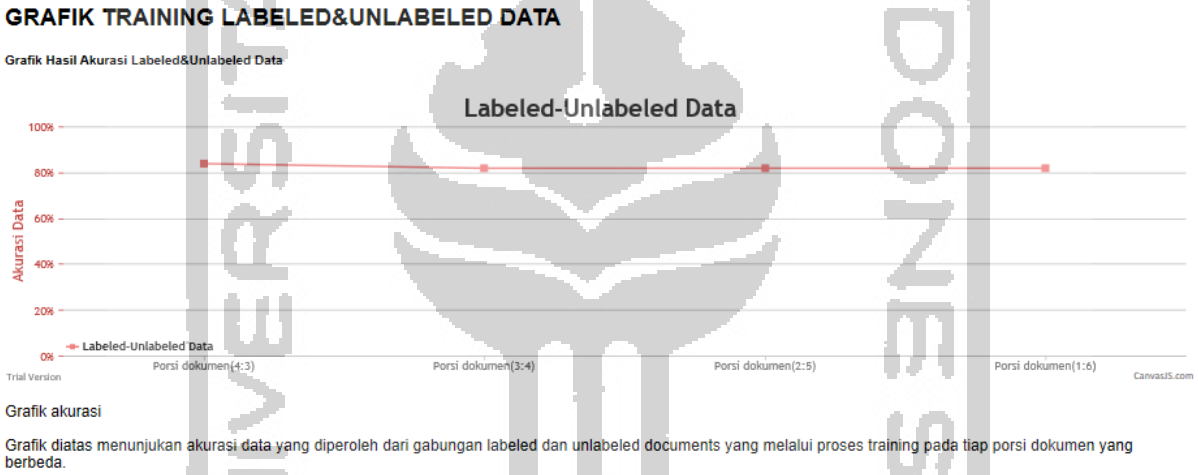
Grafik akurasi

Grafik diatas menunjukan akurasi data yang diperoleh dari proses training labeled documents pada masing-masing porsi data yang berbeda.

Gambar 4. 51 Grafik hasil *training* data berlabel



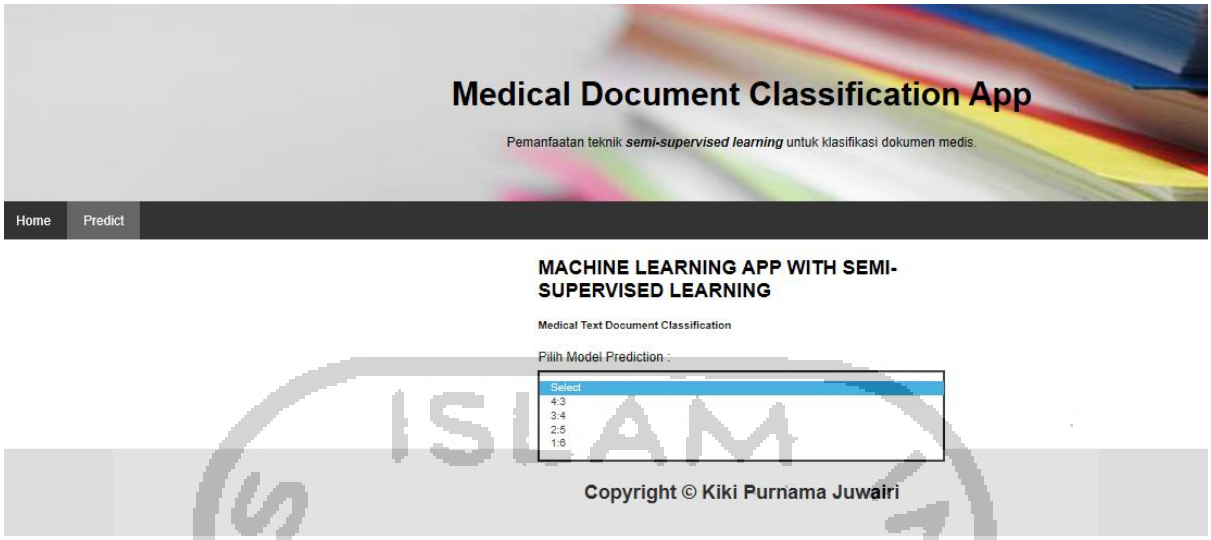
Gambar 4. 52 Grafik hasil *training* data tidak berlabel



Gambar 4. 53 Grafik hasil *training* data kombinasi

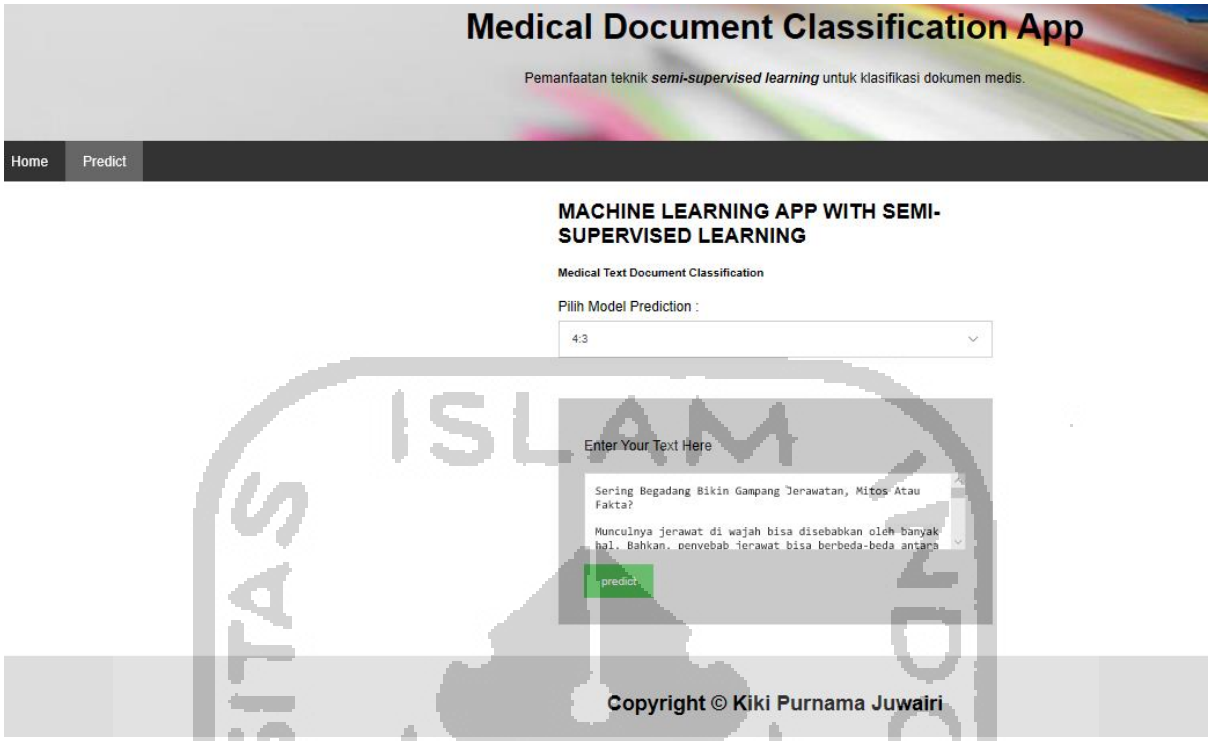
Pada halaman *predict*, teks dokumen medis diinputkan untuk melakukan proses klasifikasi. Untuk melakukan klasifikasi, *user* harus memilih satu dari empat model *classifier* yang ada pada *website*. Model *classifier* pilihan digunakan untuk melakukan prediksi pada *text input*. Model *classifier* pada *website* adalah model *classifier* yang dibangun pada percobaan 1, 2, 3, dan 4 menggunakan data kombinasi. Jadi, proses klasifikasi teks dokumen medis ini melibatkan empat model *classifier* yaitu model 4:3, 3:4, 2:5, dan 1:6.

Berdasarkan empat pilihan model classifier tersebut selanjutnya dibuat menu *dropdown* untuk memilih model *classifier* yang tersedia pada proses klasifikasi seperti pada Gambar 4.54.

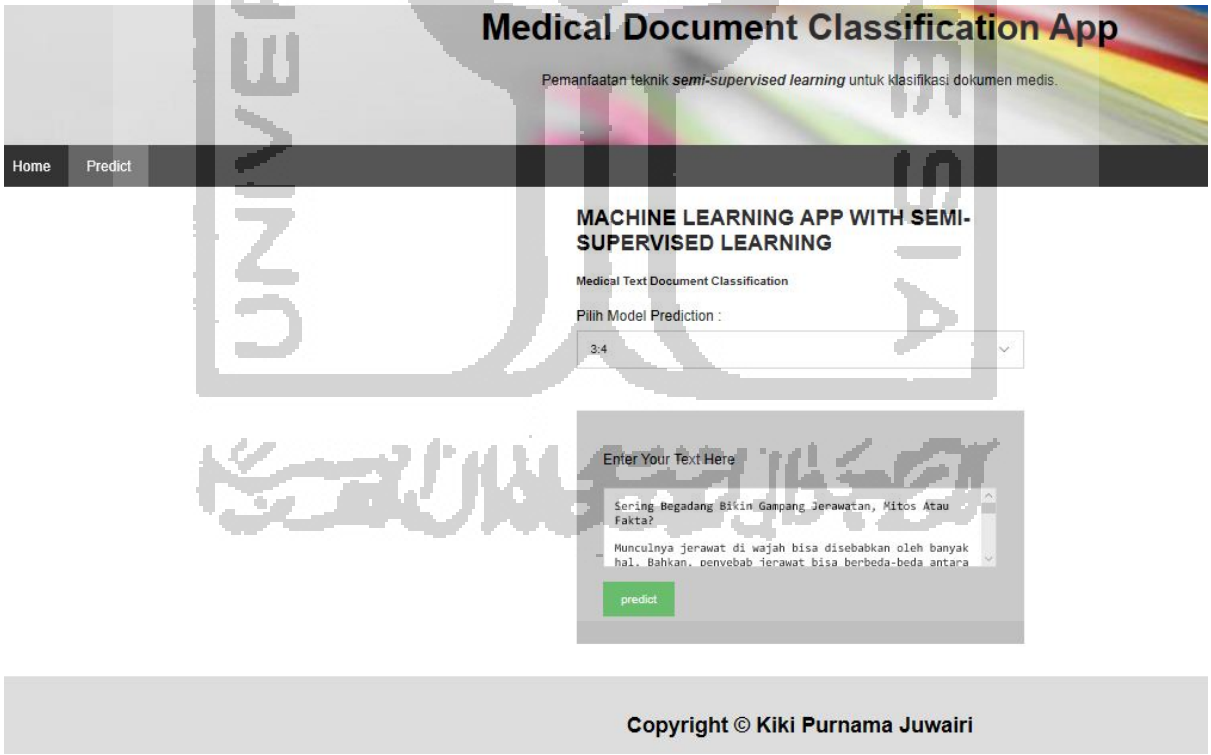


Gambar 4. 54 Menu *dropdown* pilihan model *classifier*

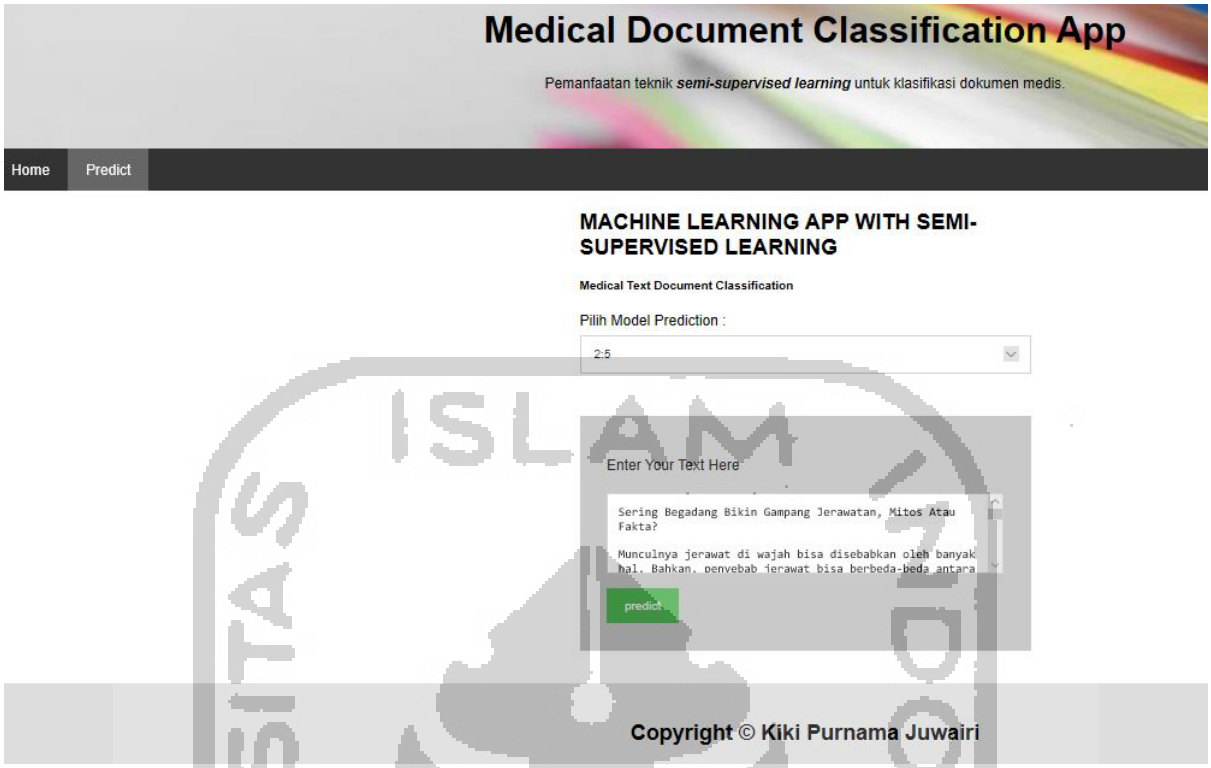
Setelah memilih model *classifier* yang akan digunakan untuk melakukan klasifikasi seperti yang ditunjukan oleh Gambar 4.54 akan muncul *text box*. *Text box* tersebut digunakan untuk menginputkan teks dokumen medis yang akan di klasifikasi menggunakan model *classifier* yang dipilih. Seperti pada Gambar 4.55, Gambar 4.56, Gambar 4.57, dan Gambar 4.58 adalah *text box* berupa input teks dokumen medis yang siap untuk di klasifikasi menggunakan model *classifier* yang dipilih *user*.



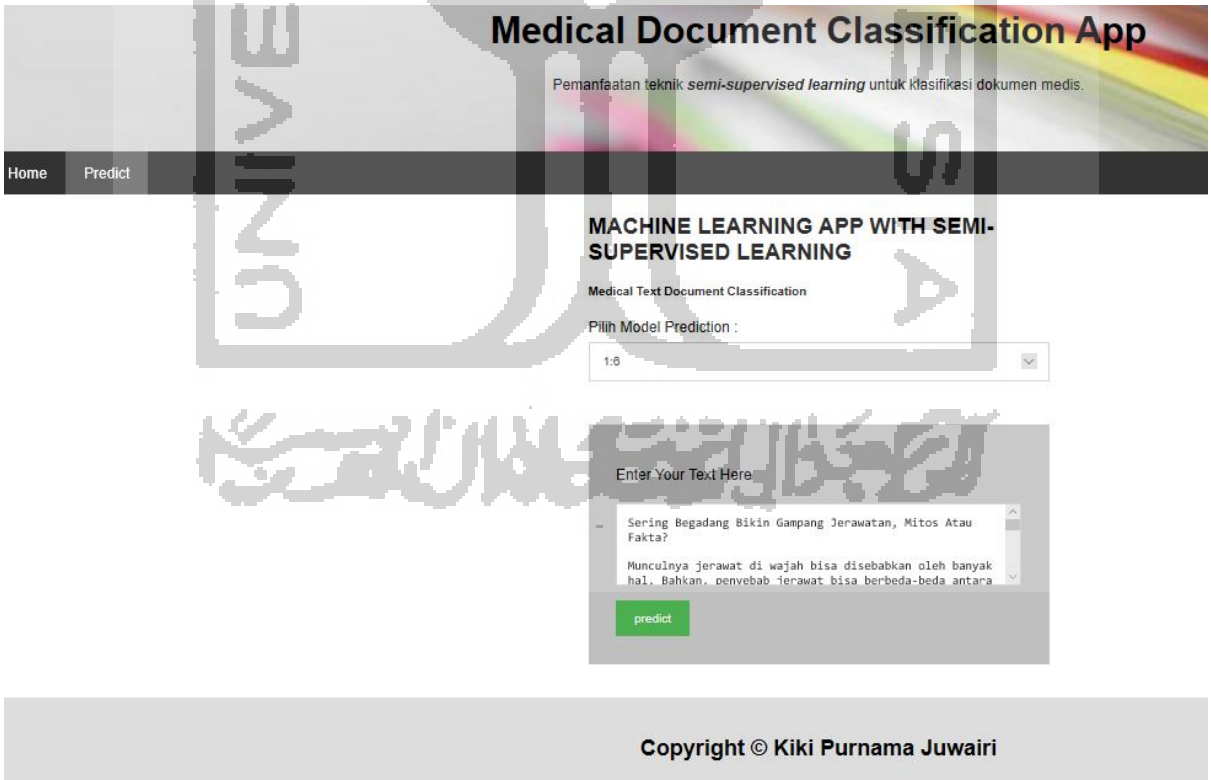
Gambar 4. 55 *Text input* menggunakan model 4:3



Gambar 4. 56 *Text input* menggunakan model 3:4

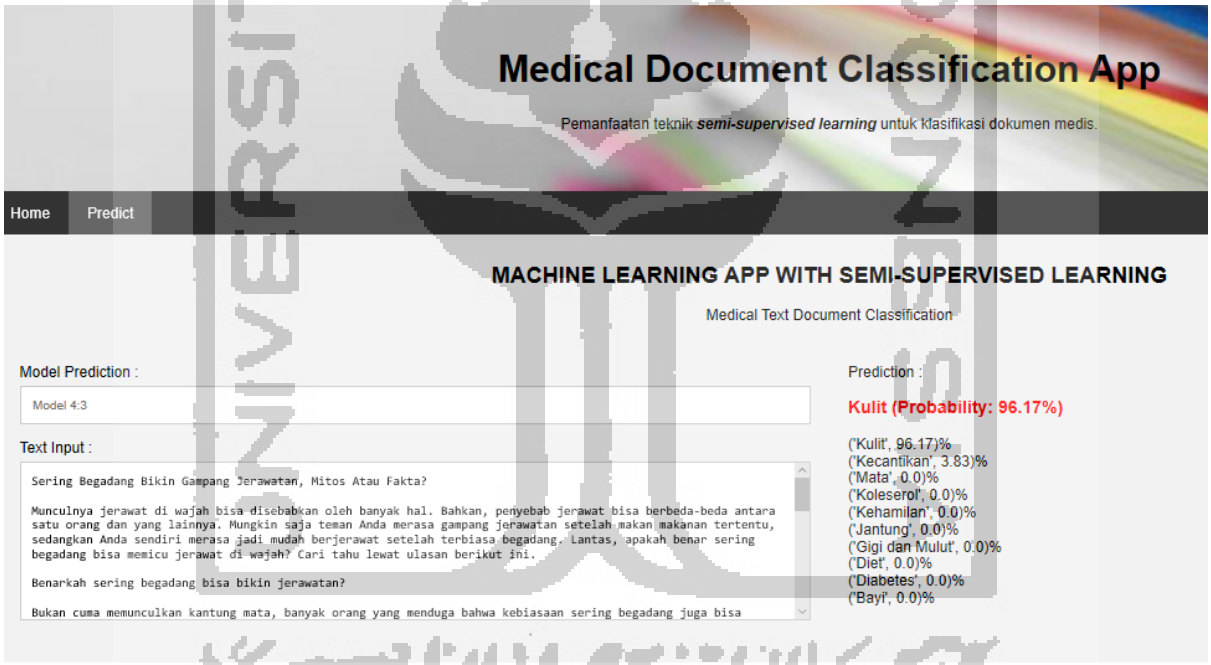


Gambar 4. 57 *Text input* menggunakan model 2:5

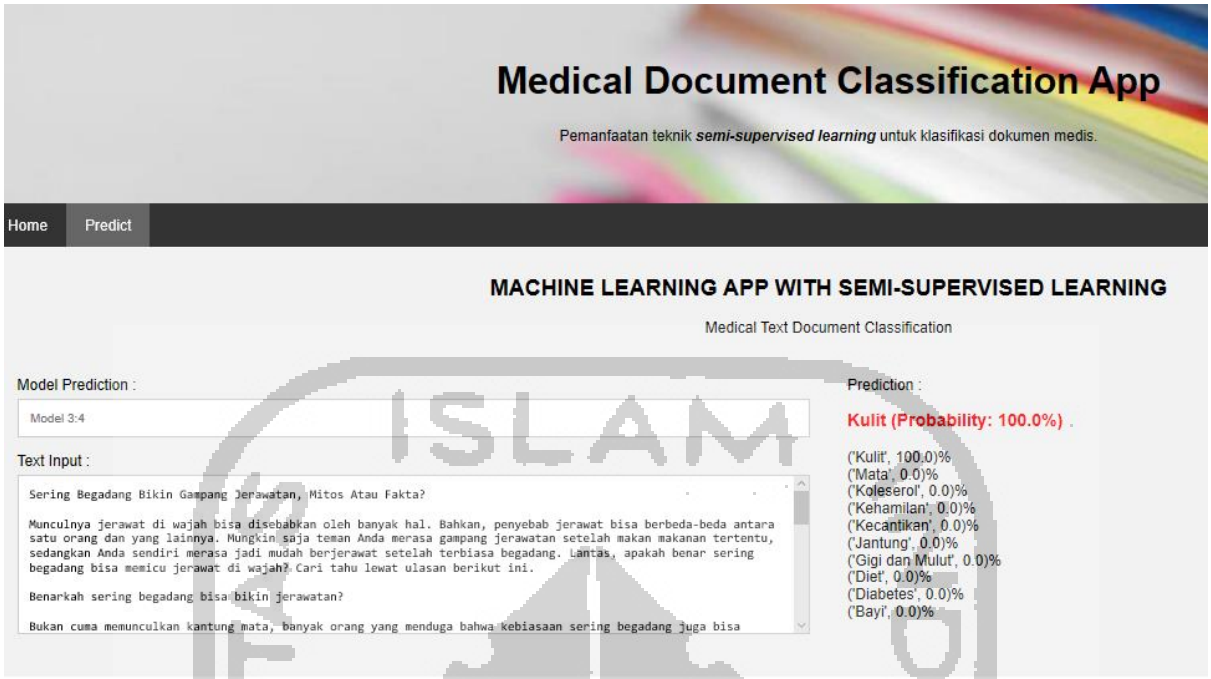


Gambar 4. 58 *Text input* menggunakan model 1:6

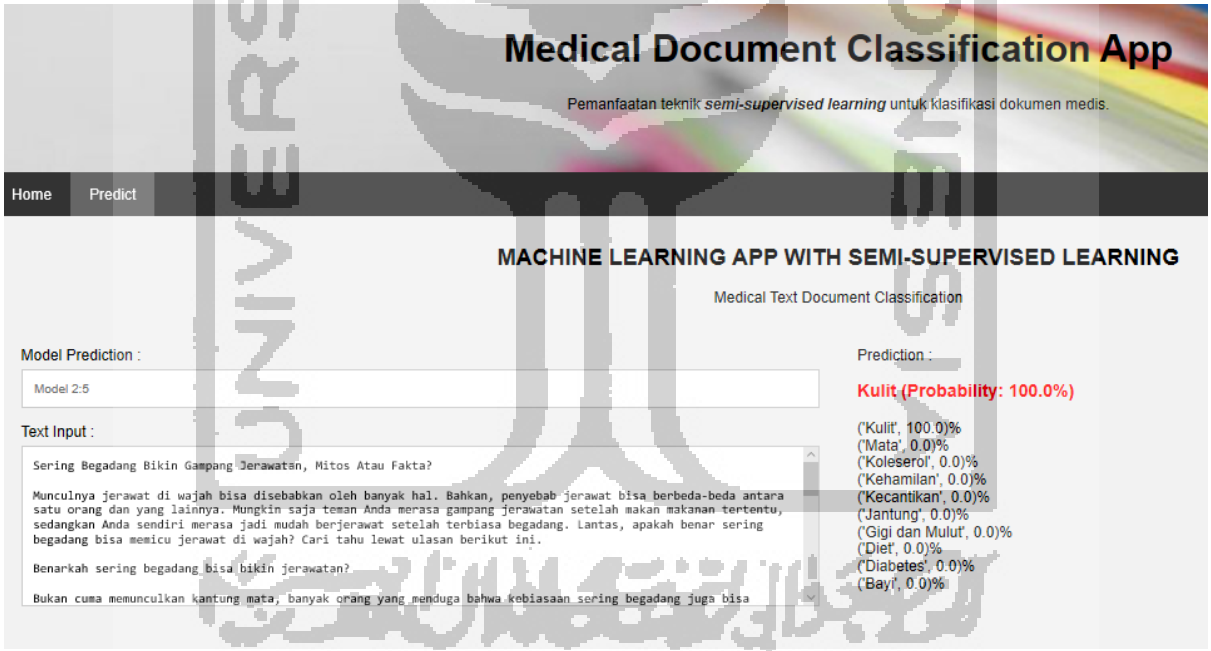
Setelah memasukan teks berupa dokumen medis ke dalam *text box* dan memilih model *classifier* yang akan digunakan untuk klasifikasi pada *website*, langkah selanjutnya adalah melakukan prediksi teks yang sudah diinputkan. Prediksi dilakukan untuk mengetahui kategori dari teks yang diinputkan. Selain mengetahui kategori/kelas dari teks dokumen tersebut, *user* juga akan mengetahui nilai probabilitas dari setiap kategori dokumen yang ada pada sistem. Untuk memulai proses klasifikasi dan mengetahui kategori dari teks dokumen yang diinputkan, maka *user* harus menekan tombol *predict* yang berwarna hijau. Setelah menekan tombol *predict* maka prediksi akan dilakukan oleh sistem dan hasil dari proses prediksi tersebut akan ditampilkan pada *website*. Berikut ini adalah tampilan hasil dari masing-masing proses klasifikasi menggunakan seluruh model *classifier* ditunjukan pada Gambar 4.59, Gambar 4.60, Gambar 4.61, dan Gambar 4.62.



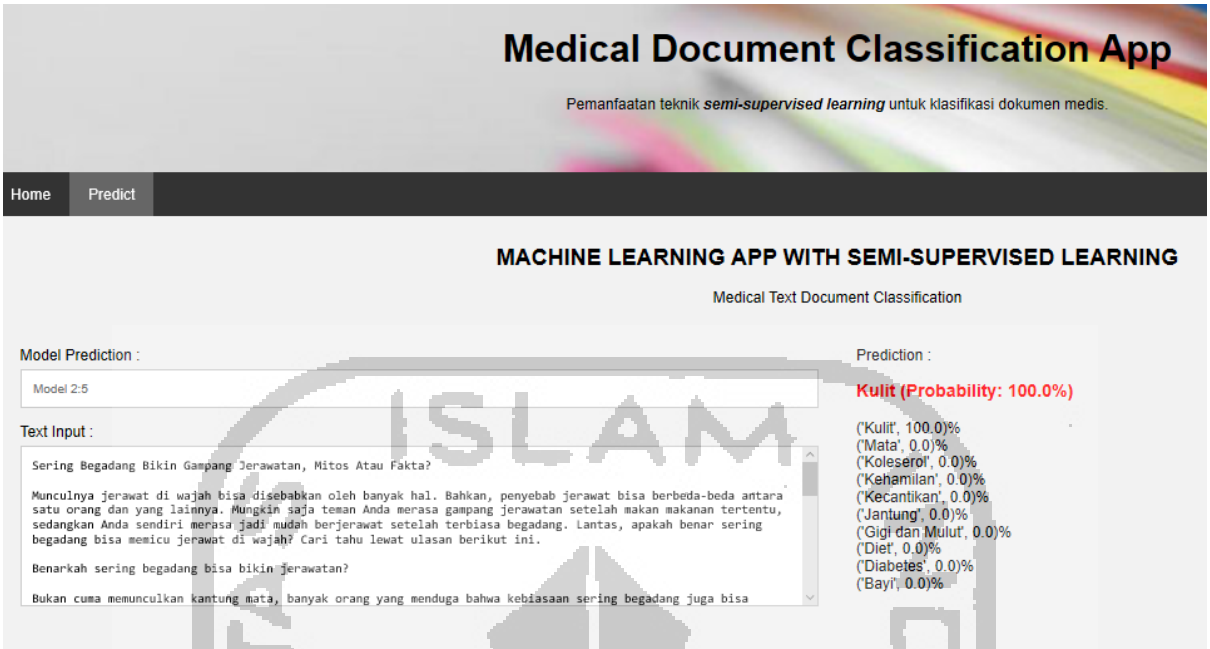
Gambar 4. 59 Hasil prediksi menggunakan model 4:3



Gambar 4. 60 Hasil prediksi menggunakan model 3:4



Gambar 4. 61 Hasil prediksi menggunakan model 2:5



Gambar 4. 62 Hasil prediksi menggunakan model 1:6

Berdasarkan Gambar 4.59, Gambar 4.60, Gambar 4.61 dan Gambar 4.62 menunjukan hasil klasifikasi yang diperoleh dari teks dokumen medis. Untuk setiap proses klasifikasi menggunakan setiap model akan mengeluarkan *output* hasil prediksi yang dilakukan model *classifier* yang dipilih. *Output* yang ditampilkan oleh *website* berupa informasi *text input* yaitu teks dokumen medis yang di klasifikasi. Kemudian, ada hasil prediksi dan nilai probabilitas dari perhitungan klasifikasi yang dilakukan. Terakhir, adalah informasi perhitungan nilai prbabilitas setiap kategori yang ditampilkan berdasarkan nilai tertinggi hingga terendah dari probabilitas yang diperoleh. Urutan nilai probabilitas yang tertinggi hingga terendah ditampilkan dari kiri ke kanan.

4.6 Hasil pengujian *website* untuk klasifikasi dokumen medis

Pengembangan *website* dilakukan untuk menguji model *classifier* yang sudah dibangun sebelumnya. Pengujian ini dilakukan untuk melihat apakah model yang dibangun sebelumnya layak untuk digunakan sebagai acuan untuk melakukan klasifikasi pada dokumen medis yang lain. Berdasarkan data *training* yang diperoleh, model *classifier* dibangun berdasarkan dua jenis data yaitu data berlabel dan data tidak berlabel. Proses *training* dilakukan pada setiap percobaan. Model pada setiap data kombinasi yang dibangun pada proses training akan digunakan untuk melakukan klasifikasi pada website. Sehingga, *website* dapat menampilkan menu prediksi yang mana menu tersebut digunakan untuk melakukan klasifikasi pada teks

dokumen medis. Berdasarkan website yang dibangun maka dilakukan uji fungsionalitas sistem yang ditunjukan pada Tabel 4.18.

Tabel 4. 17 Pengujian fungsionalitas sistem

Skenario Pengujian	Hasil
Menampilkan halaman home	Sukses
Menampilkan tabel hasil pengujian validitas	Sukses
Menampilkan grafik hasil pengujian validitas	Sukses
Menampilkan halaman predict	Sukses
Dapat memilih model classifier pada menu dropdown	Sukses
Menambahkan teks dokumen ke dalam text box	Sukses
Melakukan prediksi menggunakan model 4:3	Sukses
Melakukan prediksi menggunakan model 3:4	Sukses
Melakukan prediksi menggunakan model 2:5	Sukses
Melakukan prediksi menggunakan model 1:6	Sukses
Menampilkan hasil prediksi	Sukses
Menampilkan probabilitas hasil prediksi	Sukses
Menampilkan probabilitas masing-masing kategori	Sukses