Klasifikasi Review Baik dan Review Buruk dengan Metode Support Vector Machine (SVM) dan Bidirectional Encoder Representations from Transformers (BERT)

[1]Haikal Susanto, [2]Rafi Muhammad, [3]Gabriel Enrique
[123]Fakultas Ilmu Komputer, Universitas Indonesia, Indonesia
Email: [1]haikalsusanto21@gmail.com, [2]rafi.muhammad91@ui.ac.id,
[3]gabriel.enrique@ui.ac.id

Latar Belakang

Dalam bisnis perhotelan--ataupun bisnis jual-beli online, kuliner, dan sebagainya--review pelanggan sangatlah berguna. Review pelanggan bisa membantu pengelola bisnis dalam memperbaiki dan menjaga kualitas dari produk-produk mereka. Namun seiring dengan bertambah banyaknya pelanggan, review yang masuk akan semakin banyak pula. Sampai pada akhirnya tidak bisa ditangani oleh kemampuan fisik manusia.

Oleh karena itu, sebuah model machine learning yang bisa mendeteksi baik-buruknya sebuah review akan sangat membantu pengelola bisnis dalam mengembangkan bisnisnya.

Tujuan

Membuat model machine learning yang dapat mengklasifikasikan baik atau buruknya sebuah review berdasarkan kalimat review.

Manfaat

1. Pengelola bisnis dapat menentukan persentase kepuasan pelanggan berdasarkan review yang mereka tulis
2. Pengelola bisnis dapat memperbaiki dan menjaga kualitas bisnis yang mereka kelola

Metode Data Mining

Dataset

Prediksi yang dihasilkan berdasarkan input dari dataset yang setiap barisnya memuat review pelanggan hotel.

Secara keseluruhan, dataset yang kami gunakan adalah sebagi berikut:

- **train.csv** yang merupakan data yang berisi review beserta kategori baik-buruknya review tersebut.
- **test.csv** yang merupakan data untuk menguji model.
- **sample_submission.csv** yang merupakan contoh format submisi ke Kaggle.

Software

Dalam analisis kami menggunakan bahasa pemrograman Python dengan package-package analisis seperti Pandas, Numpy dan visualisasi data dengan menggunakan matplotlib dan seaborn.

Dalam proses preprocessing kami menggunakan library Sastrawi, Natural Language Toolkit (NLTK), dan langdetect untuk membersihkan data dari stopwords dan juga melakukan stemming pada data untuk memastikan bahwa hanya informasi yang benar - benar penting saja yang kami gunakan dalam pembuatan model

Dalam proses prediksi kami menggunakan bahasa pemrograman Python dengan package Scikit-Learn, TensorFlow dan Keras untuk menggunakan algoritma-algoritma yang dibutuhkan

Semua proses dilakukan di Kaggle dimana setiap anggota tim membuat notebook masing - masing yang kemudian hasilnya didiskusikan untuk mencapai performa model terbaik

Metode dan Algoritma

Metode yang kami gunakan diantaranya adalah menggunakan Bidirectional Encoder Representations from Transformers (BERT). Pada model bert ini menggunakan layer dari KerasLayer yang ada di TensorFlow Hub[1].

Jumlah maximum sequence length yang ditentukan adalah 128 dan batch size berjumlah 32. Epoch yang digunakan terdapat dua jenis yaitu 5 epoch dan 10 epoch. Hal ini dilakukan sebagai perbandingan. Kemudian menggunakan optimizer SGD dari Keras.

Algoritma lain yang digunakan sebagai layer awal prediksi adalah metode klasifikasi klasik seperti Support Vector Machine, K Nearest Neighbors, dan Random Forest yang digabungkan dengan TF-IDF Vectorizer

[1] https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1

Analisis

Deteksi dan Analisis Fitur-Fitur Teks Review

Dengan menggunakan Library langdetect Python, kami mengetahui bahwa bahasa review yang digunakan di dalam dataset train.csv maupun test.csv bukan hanya Bahasa Indonesia saja. Berikut adalah frekuensi bahasa di dataset train.csv. Kode bahasa yang digunakan adalah ISO 639-1.

Bahasa	Frekuensi		
id	13668	hr	15
en	430	lv	14
tl	235	nl	14
so	65	fr	11
sw	53	ro	10
de	50	cy	9
af	28	pl	9
fi	25	sv	9
et	23	tr	8
no	23	ca	8
da	23	pt	8
sl	20	it	8
sk	19	vi	7
hu	18	es	5
lt	15	sq	5

Kami juga menemukan beberapa emoji umum yang digunakan di dalam teks review. Emoji di dalam teks review berupa karakter unicode.

Ada pula emoticon-emoticon yang ada di dalam teks review. Emoticon di dalam teks review berupa karakter tanda baca (punctuations).

Data Cleaning

Metode data cleaning yang kami terapkan adalah menghapus tanda baca, menghapus stopwords, dan stemming.

Karena ada berbagai bahasa yang digunakan pada teks review, kami memutuskan untuk memproses setiap teks sesuai dengan bahasanya masing-masing. Kami menggunakan proses khusus untuk bahasa-bahasa yang paling banyak digunakan yaitu Bahasa Indonesia dan Bahasa Inggris.

Kami menggunakan library stopwords dari NLTK yang mempunyai daftar stopwords baik Bahasa Indonesia maupun Bahasa Inggris.

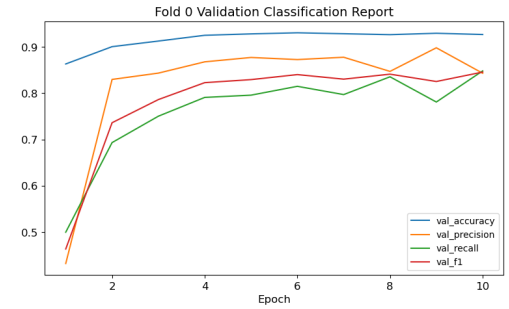
Untuk stemming, kami menggunakan Library NLTK untuk teks review Bahasa Inggris dan library Sastrawi untuk teks review Bahasa Indonesia.

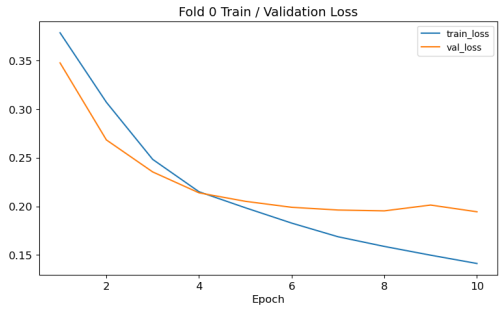
Hasil Prediksi Model

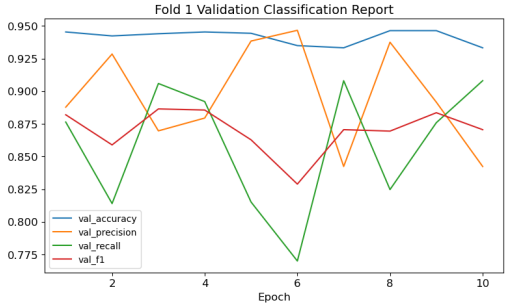
Dengan menggunakan metode-metode yang sudah kami uraikan di atas, kami bisa mendapatkan hasil tingkat persentase akurasi dari data train-test-split untuk train.csv. Tingkat persentase akurasi yang kami dapatkan adalah sebagai berikut:

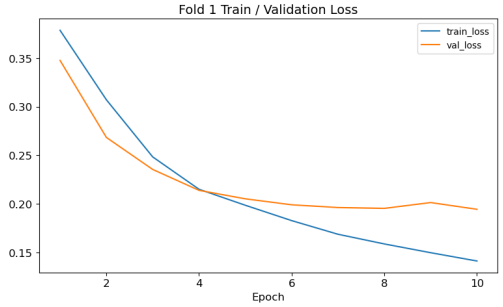
Model	**Akurasi**
SVM	0.9414644095451764
LinearSVC	0.9458277254374159

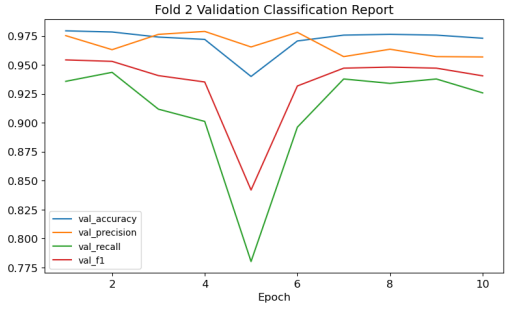
Ada pula hasil perhitungan grafik validation accuration, precision, recall, dan F1 dari setiap Fold model BERT yang digunakan.

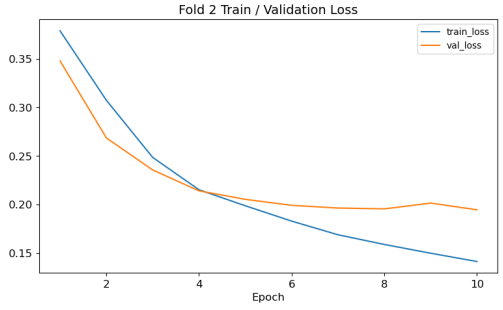


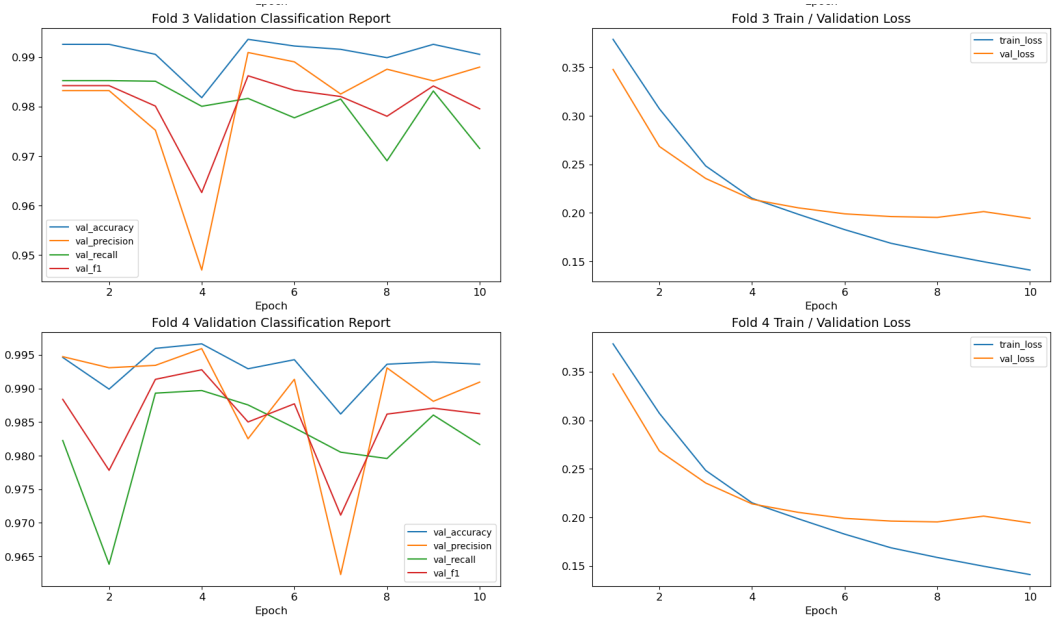












Setelah kami melakukan submisi ke Kaggle, tingkat akurasi yang kami dapatkan menurun. Berikut adalah tingkat akurasi yang kami dapatkan setelah melakukan submisi ke Kaggle:

Model	**Akurasi F1**
SVM	0.87001
LinearSVC	0.86059
BERT	0.83419

Kesimpulan

Berdasarkan hasil akurasi baik dari data train-test-split maupun Kaggle yang kami dapatkan, kami melihat bahwa model SVM adalah model yang paling efektif untuk mengklasifikasi baik-buruknya suatu review. Selain itu karena teks review tidak menggunakan gaya bahasa yang sama dan bahasa paling banyak digunakan adalah Bahasa Indonesia, proses preprocessing yang terbaik adalah dengan menghapus stopwords Bahasa Indonesia menggunakan stemming dari Sastrawi.

Dokumentasi

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv)

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
      for filename in filenames:
      print(os.path.join(dirname, filename))


# load data frame
df =
pd.read_csv("../input/penyisihan-datavidia-7-0/train.csv")
df


df.shape


# Packages
!pip install langdetect
!pip install PySastrawi


# Stemmer settings
from nltk.stem import SnowballStemmer
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

stemmer_eng = SnowballStemmer('english')

factory = StemmerFactory()
stemmer_id = factory.create_stemmer()


from langdetect import detect

def stem(x):
      try:
      bhs = detect(x)
```

```
      if bhs == 'id':
            res = stemmer_id.stem(x)
      elif bhs == 'en':
            res = stemmer_eng.stem(x)
      else:
            return x
      return res
      except LangDetectException:
      print("not detected")
      return x


from nltk.corpus import stopwords

words = stopwords.words("english") +
stopwords.words("indonesian")
words


import re

df['review_text'] = df['review_text'].apply(lambda x: "
".join([stem(x) for i in re.sub("[^a-zA-Z]", " ",
x).split() if i not in words]).lower())


df


from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(df['review_text'], df.category,
test_size=0.2)


from sklearn.feature_extraction.text import
TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.svm import LinearSVC
```

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([('vect',
TfidfVectorizer(ngram_range=(1, 2), stop_words="english",
sublinear_tf=True)),
                     ('chi',  SelectKBest(chi2,
k=10000)),
                     ('clf', LinearSVC(C=1.0,
penalty='l1', max_iter=3000, dual=False))])


model = pipeline.fit(X_train, y_train)


vectorizer = model.named_steps['vect']
chi = model.named_steps['chi']
clf = model.named_steps['clf']


print("accuracy: " + str(model.score(X_test, y_test)))


test_df =
pd.read_csv("../input/penyisihan-datavidia-7-0/test.csv")
test_df


test_df['review_text'] =
test_df['review_text'].apply(lambda x: " ".join([stem(x)
for i in re.sub("[^a-zA-Z]", " ", x).split() if i not in
words]).lower())


solve = lambda x : model.predict([x])
test_df["category"] = test_df["review_text"].map(solve)


unbox = lambda x : x[0]
test_df["category"] = test_df["category"].map(unbox)
test_df
```

```
output = pd.DataFrame({'review_id': test_df.review_id,
                       'category': test_df.category})
output.to_csv('submission_linearsvc.csv', index=False)
```

```
# This Python 3 envadironment comes with many helpful
analytics libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python



# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv)

# Input data files are available in the read-only
"../input/" directory
# For example, running this (by clicking run or pressing
Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory
(/kaggle/working/) that gets preserved as output when you
create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but
they won't be saved outside of the current session

import pandas as pd
import string, re
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import
```

```
pad_sequences
from sklearn.model_selection import train_test_split
from Sastrawi.StopWordRemover.StopWordRemoverFactory
import StopWordRemoverFactory
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
from Sastrawi.Dictionary.ArrayDictionary import
ArrayDictionary
from collections import Counter

df =
pd.read_csv("../input/penyisihan-datavidia-7-0/train.csv")

df.head()

df.shape

def cleaning(data) :
    ##case Folding
    data = data.lower()

    ##remove punctuation
    remove = string.punctuation
    translator = str.maketrans(remove," "*len(remove) )
    data = data.translate(translator)

    ##remove stopword
    factory = StopWordRemoverFactory()
    stopwords = factory.create_stop_word_remover()
    data = stopwords.remove(data)

    ##stemming
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()
    data  = stemmer.stem(data)

    return data

df['review_text_clean'] =
df['review_text'].apply(cleaning)
y = df['category']
```

```
X_train, X_test, y_train, y_test =
train_test_split(df['review_text_clean'], y,
test_size=0.33, random_state=42, stratify=y)


#Using SVM
from sklearn import svm
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.feature_extraction.text import
CountVectorizer, TfidfVectorizer



pipeline_tf = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('classifier', svm.SVC(kernel="linear",
probability=True))
])

pipeline_tf.fit(X_train, y_train)
pipeline_tf.score(X_test,y_test)




#Using Random Forest
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
pos_weights = (len(y_train) - sum(y_train)) /
(sum(y_train))
pipeline_dt = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('classifier',
RandomForestClassifier(max_depth=100,random_state=500,
class_weight={0: 1, 1: pos_weights}))
])
```

```
pipeline_dt.fit(X_train, y_train)
pipeline_dt.score(X_test,y_test)
```

```
#Using KNN
from sklearn.neighbors import KNeighborsClassifier


pipeline_nb = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('classifier', KNeighborsClassifier(n_neighbors=100)),
])

pipeline_nb.fit(X_train, y_train)
pipeline_nb.score(X_test,y_test)

pipeline_nb = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('classifier', MultinomialNB())
])

pipeline_nb.fit(X_train, y_train)
pipeline_nb.score(X_test,y_test)
```

```
#Using SVM since it has the best score

test_data =
pd.read_csv("../input/penyisihan-datavidia-7-0/test.csv")
test_data.head()
clean_test_data = test_data.review_text.apply(cleaning)
```

```
clean_test_data.tail()

res = pipeline_dt.predict(clean_test_data['review_text'])

test_data['category'] = res

submission = test_data.drop("review_text", axis=1)
submission.to_csv("submission_dt.csv", index=False)
```

```
def __init__(self, bert_layer, max_seq_length=128,
lr=0.0001, epochs=15, batch_size=32):

        # BERT and Tokenization params
        self.bert_layer = bert_layer

        self.max_seq_length = max_seq_length
        vocab_file =
self.bert_layer.resolved_object.vocab_file.asset_path.nump
y()
        do_lower_case =
self.bert_layer.resolved_object.do_lower_case.numpy()
        self.tokenizer =
tokenization.FullTokenizer(vocab_file, do_lower_case)

        # Learning control params
        self.lr = lr
        self.epochs = epochs
        self.batch_size = batch_size

        self.models = []
        self.scores = {}


    def encode(self, texts):

        all_tokens = []
        all_masks = []
        all_segments = []

       for text in texts:
```

```
            text = self.tokenizer.tokenize(text)
            text = text[:self.max_seq_length - 2]
            input_sequence = ['[CLS]'] + text + ['[SEP]']
            pad_len = self.max_seq_length -
len(input_sequence)

            tokens =
self.tokenizer.convert_tokens_to_ids(input_sequence)
            tokens += [0] * pad_len
            pad_masks = [1] * len(input_sequence) + [0] *
pad_len
            segment_ids = [0] * self.max_seq_length

            all_tokens.append(tokens)
            all_masks.append(pad_masks)
            all_segments.append(segment_ids)

        return np.array(all_tokens), np.array(all_masks),
np.array(all_segments)


def build_model(self):

        input_word_ids =
Input(shape=(self.max_seq_length,), dtype=tf.int32,
name='input_word_ids')
        input_mask = Input(shape=(self.max_seq_length,),
dtype=tf.int32, name='input_mask')
        segment_ids = Input(shape=(self.max_seq_length,),
dtype=tf.int32, name='segment_ids')

        pooled_output, sequence_output =
self.bert_layer([input_word_ids, input_mask, segment_ids])
        clf_output = sequence_output[:, 0, :]
        out = Dense(1, activation='sigmoid')(clf_output)

        model = Model(inputs=[input_word_ids, input_mask,
segment_ids], outputs=out)
        optimizer = SGD(learning_rate=self.lr,
momentum=0.8)
```

```
        model.compile(loss='binary_crossentropy',
optimizer=optimizer, metrics=['accuracy'])

        return model

  def train(self, X):

        for fold, (trn_idx, val_idx) in
enumerate(skf.split(X['stemmed'], X['category'])):

            print('\nFold {}\n'.format(fold))

            X_trn_encoded = self.encode(X.loc[trn_idx,
'stemmed'].str.lower())
            y_trn = X.loc[trn_idx, 'category']
            X_val_encoded = self.encode(X.loc[val_idx,
'stemmed'].str.lower())
            y_val = X.loc[val_idx, 'category']

            # Callbacks
            metrics =
ClassificationReport(train_data=(X_trn_encoded, y_trn),
validation_data=(X_val_encoded, y_val))

            # Model
            model = self.build_model()
            model.fit(X_trn_encoded, y_trn,
validation_data=(X_val_encoded, y_val),
callbacks=[metrics], epochs=self.epochs,
batch_size=self.batch_size)

            self.models.append(model)
            self.scores[fold] = {
                'train': {
                    'precision':
metrics.train_precision_scores,
                    'recall': metrics.train_recall_scores,
                    'f1': metrics.train_f1_scores
                },
                'validation': {
```

```
                    'precision':
metrics.val_precision_scores,
                    'recall': metrics.val_recall_scores,
                    'f1': metrics.val_f1_scores
                }
            }

def plot_learning_curve(self):

        fig, axes = plt.subplots(nrows=K, ncols=2,
figsize=(20, K * 6), dpi=100)

        for i in range(K):

            # Classification Report curve
            sns.lineplot(x=np.arange(1, self.epochs + 1),
y=text_classifier.models[i].history.history['val_accuracy'
], ax=axes[i][0], label='val_accuracy')
            sns.lineplot(x=np.arange(1, self.epochs + 1),
y=text_classifier.scores[i]['validation']['precision'],
ax=axes[i][0], label='val_precision')
            sns.lineplot(x=np.arange(1, self.epochs + 1),
y=text_classifier.scores[i]['validation']['recall'],
ax=axes[i][0], label='val_recall')
            sns.lineplot(x=np.arange(1, self.epochs + 1),
y=text_classifier.scores[i]['validation']['f1'],
ax=axes[i][0], label='val_f1')

            axes[i][0].legend()
            axes[i][0].set_title('Fold {} Validation
Classification Report'.format(i), fontsize=14)



           # Loss curve
            sns.lineplot(x=np.arange(1, self.epochs + 1),
y=text_classifier.models[0].history.history['loss'],
ax=axes[i][1], label='train_loss')
            sns.lineplot(x=np.arange(1, self.epochs + 1),
y=text_classifier.models[0].history.history['val_loss'],
```

```
ax=axes[i][1], label='val_loss')

            axes[i][1].legend()
            axes[i][1].set_title('Fold {} Train /
Validation Loss'.format(i), fontsize=14)

            for j in range(2):
                axes[i][j].set_xlabel('Epoch', size=12)
                axes[i][j].tick_params(axis='x',
labelsize=12)
                axes[i][j].tick_params(axis='y',
labelsize=12)

        plt.show()

    def predict(self, X):

        X_test_encoded =
self.encode(X['stemmed'].str.lower())
        y_pred = np.zeros((X_test_encoded[0].shape[0], 1))

        for model in self.models:
            y_pred += model.predict(X_test_encoded) /
len(self.models)

        return y_pred
```