IF5153 Pemrosesan Bahasa Alami

Tugas Kode Program

Text Classification

Disusun Oleh:

Husnia Munzayana / 13521077

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

A. Penjelasan Kode Program

Tautan Kode Program:

https://github.com/munzayanahusn/IF5153-13521077-TextClassification.git

Berikut merupakan kumpulan library yang akan digunakan serta proses mengunduh *package* `punkt` yang nantinya akan digunakan untuk tokenisasi dengan `nltk`.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import nltk
import re
import string
from collections import Counter
from nltk.tokenize import word_tokenize
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
from sklearn.feature_extraction.text import TfidfVectorizer
from IPython.display import display
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, precision_score,
recall_score, accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.feature_extraction.text import TfidfVectorizer

nltk.download('punkt')
```

Langkah pertama adalah melakukan *data loading* untuk tiga dataset, yaitu *data train*, *data validation*, dan *data test*. *Data train* digunakan dalam proses pelatihan model, di mana model akan mempelajari pola-pola dari data tersebut. *Data validation* berfungsi untuk mengevaluasi kinerja setiap model atau setiap skenario eksperimen yang diterapkan, guna memilih model dengan performa terbaik. Setelah model terbaik terpilih, *data test* akan digunakan untuk mengukur performa *final model*.

```
file_path_train = '../data/train_preprocess.tsv'
train_data = pd.read_csv(file_path_train, sep='\t', header=None)

file_path_val = '../data/valid_preprocess.tsv'
val_data = pd.read_csv(file_path_val, sep='\t', header=None)

file_path_test = '../data/test_preprocess.tsv'
test_data = pd.read_csv(file_path_test, sep='\t', header=None)
```

Setelah proses *loading data* selesai, langkah selanjutnya adalah *data understanding*, yaitu memahami ukuran data serta distribusi sentimen yang terdapat dalam dataset. Proses ini bertujuan untuk memperoleh wawasan awal mengenai karakteristik data, seperti jumlah total data dan proporsi tiap kategori sentimen, yang akan membantu dalam menentukan pendekatan analisis dalam pemodelan nantinya.

```
# Data Understanding
print('Data Train')
print('Data Size:', train_data.shape)
print('Sample Data:')
display(train_data.head())
print('\nSentiment Distribution:', train_data[1].value_counts())

print('\nData Validation')
print('Data Size:', val_data.shape)
print('Sample Data:')
display(val_data.head())

print('\nData Test')
print('Data Size:', test_data.shape)
print('Sample Data:')
display(test_data.head())
```

Kemudian, dilakukan proses *splitting data* untuk memisahkan fitur (data_x) dan label atau target (data_y). *Splitting data* dilakukan agar model dapat mengenali hubungan antara fitur-fitur dalam data_x dan target dalam data_y.

```
# Label Splitting
x_train = train_data[0]
y_train = train_data[1]

print('x_train:\n')
```

```
display(x_train.head())
print('y_train:\n')
display(y_train.head())

x_val = val_data[0]
y_val = val_data[1]
print('x_val:\n')
display(x_val.head())
print('y_val:\n')
display(y_val.head())

x_test = test_data[0]
y_test = test_data[1]
print('x_test:\n')
display(x_test.head())
print('y_test:\n')
display(y_test.head())
```

Setelah itu, dilakukan label encoding untuk mengubah representasi label atau target menjadi bentuk numerik. Dalam proses ini, label sentimen yang awalnya berupa teks seperti "negative", "neutral", dan "positive" diubah menjadi angka, di mana "negative" direpresentasikan sebagai 0, "neutral" sebagai 1, dan "positive" sebagai 2.

```
# Label Encoding
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_val_encoded = label_encoder.transform(y_val)
y_test_encoded = label_encoder.transform(y_test)

print('Encoded labels train:', y_train_encoded)
print('Encoded labels validation:', y_val_encoded)
print('Encoded labels test:', y_test_encoded)
print('Classes:', label_encoder.classes_)
```

Pada tahap preprocessing data, dilakukan beberapa tahapan preprocessing untuk mempersiapkan data teks agar dapat digunakan secara efektif dalam pemodelan, antara lain:

a. Lowercase: Mengubah seluruh teks menjadi huruf kecil agar seragam dan menghindari perbedaan kata akibat penggunaan huruf kapital yang tidak konsisten.

b. Entity Masking: Menggantikan entitas penting seperti alamat email dan nomor telepon dengan default string: `_email_` dan `_phone_` untuk melindungi informasi sensitif dan menyederhanakan pemrosesan.
c. Tokenization: Memecah teks menjadi kata-kata menggunakan `word_tokenize` dari *library* NLTK agar lebih mudah dianalisis pada level kata.
d. Spelling Correction: Memperbaiki kesalahan ejaan dalam data. *Spelling correction* dilakukan dengan *library* Sastrawi karena teks berbahasa Indonesia. Sastrawi menggunakan pendekatan stemming untuk mengoreksi kata yang salah eja, yaitu dengan mengembalikan kata-kata ke bentuk dasarnya.
e. Penghapusan Tanda Baca: Menghapus tanda baca yang tidak diperlukan agar fokus analisis hanya pada kata-kata penting dalam teks.

```
# X Label Preprocessing
def lowercase (data):
    return data.str.lower()

def entity_masking (data):
    # Email masking
    email = re.compile(r'\w+@\w+\.[a-z]{3}')
    data_mask = data.apply(lambda x: email.sub('_email_', x))

    # Phone Number Masking
    phone =
re.compile(r'(\+?\d{1,2}[-.\s]?)?(\(?\d{3}\)?[-.\s]?)?\d{3}[-.\s]
?\d{4}')
    data_mask = data_mask.apply(lambda x: phone.sub('_phone_',
x))

    return data_mask

def tokenization (data):
    data_tokenize = data.apply(lambda x: word_tokenize(x))
    return data_tokenize

def idn_spelling_correction (data):
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()

    data_stem = []
    for sentence in data:
```

```
        stemmed_words = [stemmer.stem(word) for word in sentence]
        data_stem.append(stemmed_words)

    return data_stem

def remove_punctuation (data):
    data_nopunc = []

    for sentence in data:
        cleaned_sentence = [word for word in sentence if word not
in string.punctuation]
        data_nopunc.append(cleaned_sentence)

    return data_nopunc

def preprocess (data):
    data = lowercase(data)
    data = entity_masking(data)
    data = tokenization(data)
    data = idn_spelling_correction(data)
    data = remove_punctuation(data)
    return data
```

```
# Preprocess Data
x_train_preprocessed = preprocess(x_train)
x_val_preprocessed = preprocess(x_val)
x_test_preprocessed = preprocess(x_test)
```

Pada proses ini, dilakukan *feature extraction* menggunakan teknik Bag of Words berbasis TF-IDF (Term Frequency-Inverse Document Frequency). Sebelumnya, dokumen yang sebelumnya berbentuk list of words diubah kembali menjadi bentuk *string*. Hal ini dilakukan karena proses *vectorization* akan menggunakan TfidfVectorizer yang menerima input berupa teks (string).

```
# Feature Extraction: Bag of Words TF-IDF
x_train_preprocessed = [' '.join(doc) for doc in
x_train_preprocessed]
x_val_preprocessed = [' '.join(doc) for doc in
x_val_preprocessed]
x_test_preprocessed = [' '.join(doc) for doc in
x_test_preprocessed]
```

Dilakukan inisiasi objek `TfidfVectorizer` dengan menggunakan *library* scikit-learn. Pada proses ini, beberapa kata yang jarang muncul dalam data akan dieliminasi untuk mengurangi fitur yang kurang relevan. Eliminasi dilakukan pada kata-kata yang muncul di kurang dari 100 dokumen (`min_df=100`). Parameter `token_pattern=r'[a-zA-Z]+'` menunjukkan bahwa hanya kata-kata yang terdiri dari huruf yang akan dilibatkan dalam proses pelatihan. Setelah itu, dilakukan

```
tfidf_vec = TfidfVectorizer(min_df=100,
token_pattern=r'[a-zA-Z]+')
```

Setelah inisialisasi objek TfidfVectorizer, langkah berikutnya adalah menerapkan transformasi pada data. Pertama, `fit_transform()` diterapkan pada data train untuk membangun kamus kata dan menghitung nilai TF-IDF, mengubah data teks menjadi representasi numerik berbentuk matriks. Selanjutnya, transform() digunakan pada data validation dan test untuk menghitung nilai TF-IDF berdasarkan kata-kata yang sudah ada dari data train.

```
x_train_bow = tfidf_vec.fit_transform(x_train_preprocessed)
x_val_bow = tfidf_vec.transform(x_val_preprocessed)
x_test_bow = tfidf_vec.transform(x_test_preprocessed)
```

Selanjutnya, dilakukan pendefinisian fungsi untuk melakukan prediksi menggunakan model yang telah dilatih dan evaluasi kinerja model dengan menggunakan akurasi, presisi, recall, dan F1-Score. Setiap metrik evaluasi dihitung dengan metode *weighted* untuk menangani *imbalance data*.

```
# Evaluation Metrics: Accuracy, Precision, Recall, F1-Score
def model_predict(model, x):
    y_pred = model.predict(x)

    return y_pred

def model_evaluation(y_pred, y_actual):
    accuracy = accuracy_score(y_actual, y_pred)
    precision = precision_score(y_actual, y_pred,
average='weighted')
    recall = recall_score(y_actual, y_pred, average='weighted')
    f1 = f1_score(y_actual, y_pred, average='weighted')
```

```
    return accuracy, precision, recall, f1
```

Pada proses *model training*, akan digunakan tiga algoritma *machine learning* yaitu Logistic Regression, Random Forest, dan SVM. Setiap model akan dilatih dengan *data train*, kemudian dilakukan prediksi pada *data validation.* Setelah itu, hasil prediksi akan dievaluasi menggunakan metrik akurasi, presisi, recall, dan F1-Score.

```
# Model Training
# Logistic Regression
print('Logistic Regression')
logreg = LogisticRegression(max_iter=1000)
logreg.fit(x_train_bow, y_train_encoded)

# Predictions validation data
y_pred = model_predict(logreg, x_val_bow)
print('Predictions:', y_pred)
print('Actual:', y_val_encoded)
print()

evaluation_metrics = model_evaluation(y_pred, y_val_encoded)
print('Accuracy:', evaluation_metrics[0])
print('Precision:', evaluation_metrics[1])
print('Recall:', evaluation_metrics[2])
print('F1-Score:', evaluation_metrics[3])
```

Pada setiap model, dilakukan proses hyperparameter tuning menggunakan metode Grid Search untuk mencari kombinasi parameter terbaik yang dapat meningkatkan performa model.

```
# Hyperparameter Tuning
print("\nHyperparameter Tuning for Logistic Regression")
param_grid_logreg = {
    'C': [0.1, 1, 10],
    'solver': ['newton-cg', 'lbfgs', 'liblinear']
}
grid_search_logreg =
GridSearchCV(LogisticRegression(max_iter=1000),
param_grid_logreg, cv=5)
grid_search_logreg.fit(x_train_bow, y_train_encoded)
print('Best Parameters (LogReg):',
grid_search_logreg.best_params_)
```

```
print('Best Score (LogReg):', grid_search_logreg.best_score_)

# Predictions validation data
y_pred = model_predict(grid_search_logreg, x_val_bow)
print('Predictions:', y_pred)
print('Actual:', y_val_encoded)
print()

evaluation_metrics = model_evaluation(y_pred, y_val_encoded)
print('Accuracy:', evaluation_metrics[0])
print('Precision:', evaluation_metrics[1])
print('Recall:', evaluation_metrics[2])
```

Hal yang sama dilakukan terhadap model Random Forest dan SVM.

```
# Random Forest Classifier
print('Random Forest')
rf = RandomForestClassifier()
rf.fit(x_train_bow, y_train_encoded)

# Prediction on validation set
y_pred = model_predict(rf, x_val_bow)
print('Predictions:', y_pred)
print('Actual:', y_val_encoded)
print()

evaluation_metrics = model_evaluation(y_pred, y_val_encoded)
print('Accuracy:', evaluation_metrics[0])
print('Precision:', evaluation_metrics[1])
print('Recall:', evaluation_metrics[2])
print('F1-Score:', evaluation_metrics[3])

# Hyperparameter Tuning
print("\nHyperparameter Tuning for Random Forest")
param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20]
}
grid_search_rf = GridSearchCV(RandomForestClassifier(),
param_grid_rf, cv=5)
grid_search_rf.fit(x_train_bow, y_train_encoded)
print('Best Parameters (RandomForest):',
```

```
grid_search_rf.best_params_)
print('Best Score (RandomForest):', grid_search_rf.best_score_)

# Prediction on validation data
y_pred = model_predict(grid_search_rf, x_val_bow)
print('Predictions:', y_pred)
print('Actual:', y_val_encoded)
print()

evaluation_metrics = model_evaluation(y_pred, y_val_encoded)
print('Accuracy:', evaluation_metrics[0])
print('Precision:', evaluation_metrics[1])
print('Recall:', evaluation_metrics[2])
print('F1-Score:', evaluation_metrics[3])
```

```
# Support Vector Machine
svm = SVC()
svm.fit(x_train_bow, y_train_encoded)

# Prediction on validation data
y_pred = model_predict(svm, x_val_bow)
print('Predictions:', y_pred)
print('Actual:', y_val_encoded)
print()

evaluation_metrics = model_evaluation(y_pred, y_val_encoded)
print('Support Vector Machine')
print('Accuracy:', evaluation_metrics[0])
print('Precision:', evaluation_metrics[1])
print('Recall:', evaluation_metrics[2])
print('F1-Score:', evaluation_metrics[3])

# Hyperparameter Tuning
print("\nHyperparameter Tuning for Support Vector Machine")
param_grid_svm = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf']
}

grid_search_svm = GridSearchCV(SVC(), param_grid_svm, cv=5)
grid_search_svm.fit(x_train_bow, y_train_encoded)
```

```
print('Best Parameters (SVM):', grid_search_svm.best_params_)
print('Best Score (SVM):', grid_search_svm.best_score_)

# Prediction on validation data
y_pred = model_predict(grid_search_svm, x_val_bow)
print('Predictions:', y_pred)
print('Actual:', y_val_encoded)
print()

evaluation_metrics = model_evaluation(y_pred, y_val_encoded)
print('Accuracy:', evaluation_metrics[0])
print('Precision:', evaluation_metrics[1])
print('Recall:', evaluation_metrics[2])
print('F1-Score:', evaluation_metrics[3])
```

Setelah menganalisis hasil evaluasi pada setiap model serta hasil tuningnya, ditemukan bahwa model Support Vector Machine (SVM) Hyperparameter Tuning memiliki nilai F1 Score terbaik. Oleh karena itu, model tersebut akan digunakan untuk melakukan prediksi pada data test untuk mengukur performa final model pada *unknown data*.

```
# Data Test Prediction
y_test_pred = model_predict(grid_search_svm, x_test_bow)
print('Predictions:\n', y_test_pred)
print('Actual:\n', y_test_encoded)
print()

# Confusion Matrix
conf_matrix = confusion_matrix(y_test_encoded, y_test_pred)
plt.figure(figsize=(5, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=label_encoder.classes_,
yticklabels=label_encoder.classes_)
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title('Confusion Matrix')
plt.show()

# Evaluation Metrics
evaluation_metrics = model_evaluation(y_test_pred,
y_test_encoded)
```

```
print('Accuracy:', evaluation_metrics[0])
print('Precision:', evaluation_metrics[1])
print('Recall:', evaluation_metrics[2])
print('F1-Score:', evaluation_metrics[3])
```

B. Skenario Eksperimen

Pada eksperimen ini, dilakukan *model training* dengan tiga algoritma *machine learning* untuk memprediksi sentimen dari data teks. Tahapan dalam eksperimen ini meliputi:

1. *Preprocessing Data*
    a. Mengonversi seluruh teks menjadi huruf kecil (*lowercase*).
    b. *Entity masking* untuk mengubah informasi sensitif menjadi *default string*
    c. Tokenization
    d. Spelling Correction
    e. Penghapusan tanda baca

2. *Feature Extraction*: Setelah data di-*preprocess*, digunakan metode Bag of Words dengan TF-IDF untuk mengubah teks menjadi representasi numerik.

3. *Model Training*
    a. Tiga algoritma *machine learning* yang digunakan antara lain:
        1) Logistic Regression
        2) Random Forest
        3) Support Vector Machine (SVM)

        Setiap model dilatih menggunakan *data train*, dan prediksi dievaluasi pada *data validation.*
    b. Hyperparameter Tuning: Dilakukan *hyperparameter tuning* menggunakan Grid Search pada setiap model untuk mencari kombinasi parameter terbaik. Grid Search dilakukan dengan *5-fold cross-validation* untuk mendapatkan parameter paling optimal.

4. Evaluasi Model: Setiap model dievaluasi berdasarkan metrik akurasi, presisi, recall, dan F1-Score. Model dengan nilai F1-Score terbaik dipilih sebagai model final.

5. Prediksi pada Data Test: Model yang memiliki performa terbaik digunakan untuk melakukan prediksi pada *data test*.

C. Hasil Eksperimen

1. Logistic Regression

```
Logistic Regression
Predictions: [0 2 2 ... 1 0 2]
Actual: [1 0 2 ... 0 0 2]

Accuracy: 0.823015873015873
Precision: 0.8216928033208631
Recall: 0.823015873015873
F1-Score: 0.822074686802914
```

2. Logistic Regression - Hyperparameter Tuning

```
Hyperparameter Tuning for Logistic Regression
Best Parameters (LogReg): {'C': 1, 'solver': 'lbfgs'}
Best Score (LogReg): 0.8291818181818181
Predictions: [0 2 2 ... 1 0 2]
Actual: [1 0 2 ... 0 0 2]

Accuracy: 0.823015873015873
Precision: 0.8216928033208631
Recall: 0.823015873015873
F1-Score: 0.822074686802914
```

3. Random Forest

```
Random Forest
Predictions: [0 0 2 ... 0 0 2]
Actual: [1 0 2 ... 0 0 2]

Accuracy: 0.8301587301587302
Precision: 0.8255855611250349
Recall: 0.8301587301587302
F1-Score: 0.8260421591490895
```

4. Random Forest - Hyperparameter Tuning

```
Hyperparameter Tuning for Random Forest
Best Parameters (RandomForest): {'max_depth': 20,
'n_estimators': 200}
Best Score (RandomForest): 0.796
Predictions: [0 0 2 ... 0 0 2]
Actual: [1 0 2 ... 0 0 2]

Accuracy: 0.8031746031746032
Precision: 0.8153691313645595
Recall: 0.8031746031746032
F1-Score: 0.7884396291653665
```

5. Support Vector Machine (SVM)

```
Support Vector Machine
Predictions: [0 2 2 ... 1 0 2]
Actual: [1 0 2 ... 0 0 2]

Accuracy: 0.8468253968253968
Precision: 0.8468943206791202
Recall: 0.8468253968253968
F1-Score: 0.8447536477080531
```

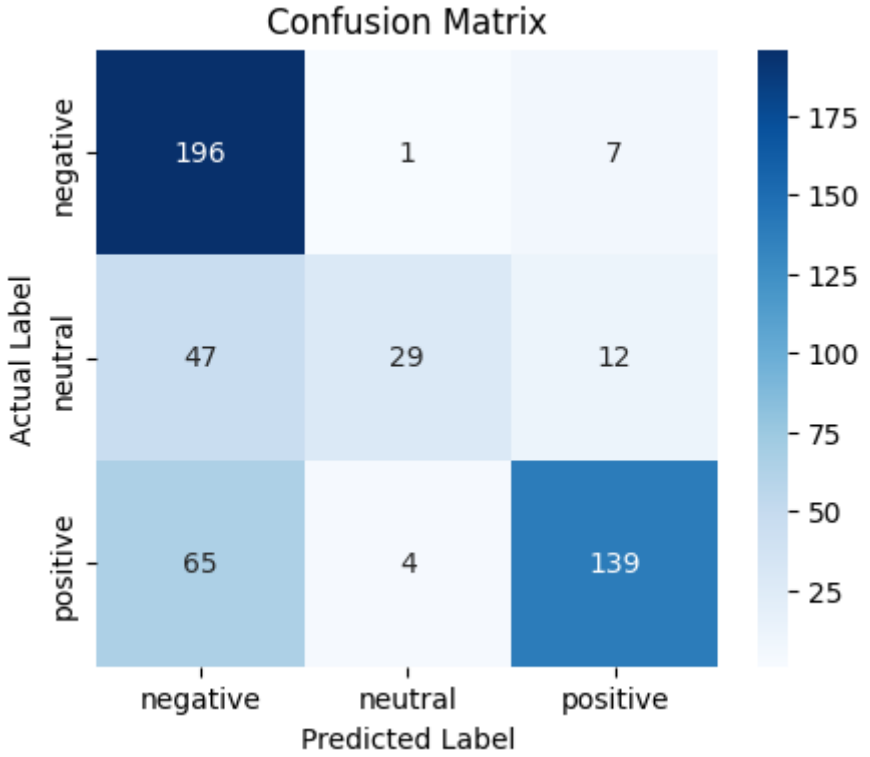
6. Support Vector Machine (SVM) - Hyperparameter Tuning

```
Hyperparameter Tuning for Support Vector Machine
Best Parameters (SVM): {'C': 10, 'kernel': 'rbf'}
Best Score (SVM): 0.8569090909090908
Predictions: [0 0 2 ... 1 0 2]
Actual: [1 0 2 ... 0 0 2]

Accuracy: 0.8539682539682539
Precision: 0.8526444791544027
Recall: 0.8539682539682539
F1-Score: 0.8527648498618379
```

D. Error Analysis

Berikut adalah hasil analisis kesalahan (error analysis) terhadap prediksi data test menggunakan model Support Vector Machine (SVM) setelah dilakukan Hyperparameter Tuning:




```
Accuracy: 0.728
Precision: 0.775728694239491
Recall: 0.728
F1-Score: 0.7120252732240437
```