LAMPIRAN

Script hiperparameter *fine-tuning*

```
import pandas as pd
import re
import torch
import torch.nn as nn
from transformers import BertTokenizer, BertModel,
get_linear_schedule_with_warmup
from torch.optim import AdamW
from torch.utils.data import TensorDataset, DataLoader
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score,
classification_report, balanced_accuracy_score
from sklearn.model_selection import train_test_split
import numpy as np
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
from itertools import product
import time
import json


print("Running hyperparameter tuning with 8 combinations")


# Fungsi cleaning text
def clean_text(text):
    if pd.isna(text):
        return ""
    text = str(text).lower()
    text = re.sub(r'[^a-z0-9\s]', '', text)
```

```
    return ' '.join(text.split())


# Fungsi augmentasi sederhana
def augment_text(row, min_samples=20):
    r201, r202, r201b = row['R201'], row['R202'], row['R201B']
    augmented = [f"{r201} [SEP] {r202}"]  # Data asli

    # Hapus baris berikut agar tidak ada augmentasi tukar R201 dan R202
    # augmented.append(f"{r202} [SEP] {r201}")

    # Augmentasi 2: Variasi berbasis sinonim atau rephrasing
    r201_variations = [
        r201,
        r201.replace("menjual", "berdagang") if "menjual" in r201 else r201,
        r201.replace("jual", "menjual") if "jual" in r201 else r201,
        r201.replace("membuat", "memproduksi") if "membuat" in r201 else r201,
        r201.replace("industri", "pembuatan") if "industri" in r201 else r201,
        r201.replace("pembuatan", "industri") if "pembuatan" in r201 else r201,
        r201.replace("reparasi", "memperbaiki") if "reparasi" in r201 else r201,
        r201.replace("memperbaiki", "service") if "memperbaiki" in r201 else r201,
        r201.replace("untuk hewan", "untuk binatang") if "untuk hewan" in r201 else r201,
        f"aktivitas {r201}",
        f"usaha {r201}" if f"usaha" not in r201 else r201,
        f"melakukan {r201}",
        f"pekerjaan {r201}",
        f"bisnis {r201}",
    ]
```

```
    r202_variations = [
        r202,
        f"hasil {r202}",
        f"produk {r202}",
        f"barang {r202}",
        f"item {r202}",
        f"keluaran {r202}",
    ]

    # Tambahkan kombinasi variasi hingga mencapai min_samples
    for vr201 in r201_variations:
        for vr202 in r202_variations:
            if len(augmented) < min_samples:
                augmented.append(f"{vr201} [SEP] {vr202}")
            else:
                break
        if len(augmented) >= min_samples:
            break

    # Jika masih kurang dari min_samples, ulangi data asli dengan sedikit modifikasi
    while len(augmented) < min_samples:
        augmented.append(f"{r201} ke-{len(augmented) + 1} [SEP] {r202}")

    return augmented

# Load data
excel_file = '/kaggle/input/dataset/kbli.xlsx'
```

```
    df = pd.read_excel(excel_file).dropna(subset=['R201', 'R202', 'R201B'])


# Clean data awal
df['R201'] = df['R201'].apply(clean_text)
df['R202'] = df['R202'].apply(clean_text)


# Hitung frekuensi kelas
class_counts = df['R201B'].value_counts()
print("Jumlah kelas sebelum augmentasi:", len(class_counts))
print("Jumlah data sebelum augmentasi:", len(df))


# Augmentasi untuk kelas < 20
df_augmented = []
for r201b, group in df.groupby('R201B'):
  if len(group) >= 20:
    # Kelas >= 20, gunakan data asli
    for _, row in group.iterrows():
      df_augmented.append({'input_text': f"{row['R201']} [SEP] {row['R202']}",
'R201B': r201b})
  else:
    # Kelas < 20, augmentasi hingga minimal 20
    for _, row in group.iterrows():
      augmented_texts = augment_text(row)
      for text in augmented_texts:
        df_augmented.append({'input_text': text, 'R201B': r201b})


# Konversi ke DataFrame
df_filtered = pd.DataFrame(df_augmented)
```

```
 # Konversi ke DataFrame
df_filtered = pd.DataFrame(df_augmented)
df_filtered['input_text'] = df_filtered['input_text'].fillna("")


# Verifikasi
print("Jumlah kelas setelah augmentasi:", df_filtered['R201B'].nunique())
print("Jumlah data setelah augmentasi:", len(df_filtered))
print("Distribusi minimal setelah augmentasi:",
df_filtered['R201B'].value_counts().min())


# Split dataset menjadi train, validation, dan test
from sklearn.model_selection import train_test_split


# Split awal: train+val dan test (20% test)
trainval_df, test_df = train_test_split(
    df_filtered, test_size=0.2, stratify=df_filtered['R201B'], random_state=42
)


# Split train+val menjadi train dan val (20% val dari sisa, yaitu 16% dari total)
train_df, val_df = train_test_split(
    trainval_df, test_size=0.2, stratify=trainval_df['R201B'], random_state=42
)


print("Jumlah data train:", len(train_df))
print("Jumlah data val:", len(val_df))
print("Jumlah data test:", len(test_df))


# Load tokenizer dan model
```

```
tokenizer = BertTokenizer.from_pretrained("indobenchmark/indobert-base-p2")


# Preprocessing
def preprocess(texts, max_length=128):
    if isinstance(texts, str):
        texts = [texts]
    texts = ["" if pd.isna(t) or not isinstance(t, str) else t for t in texts]
    if not texts:
        texts = [""]
    inputs = tokenizer(texts, max_length=max_length, padding=True, truncation=True,
return_tensors="pt")
    return inputs


# Encode labels
le = LabelEncoder()
train_labels_encoded = le.fit_transform(train_df['R201B'])
val_labels_encoded = le.transform(val_df['R201B'])
test_labels_encoded = le.transform(test_df['R201B'])  # Tambahan untuk test set


# Model
class IndoBERTClassifier(nn.Module):
    def __init__(self, bert_model, num_classes, hidden_size=768, dropout=0.1):
        super(IndoBERTClassifier, self).__init__()
        self.bert = bert_model
        self.dropout = nn.Dropout(dropout)
        self.fc1 = nn.Linear(hidden_size, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, num_classes)
```

```
    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        cls_output = outputs.last_hidden_state[:, 0, :]
        x = self.dropout(cls_output)
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = torch.relu(self.fc2(x))
        x = self.dropout(x)
        logits = self.fc3(x)
        return logits

# Fungsi evaluasi lengkap dengan metrik weighted dan balanced
def evaluate_model(model, val_loader, device):
    model.eval()
    all_preds = []
    all_labels = []
    total_loss = 0
    loss_fn = nn.CrossEntropyLoss()

    with torch.no_grad():
        for batch in val_loader:
            input_ids, attention_mask, labels = [x.to(device) for x in batch]
            logits = model(input_ids, attention_mask)
            loss = loss_fn(logits, labels)
            total_loss += loss.item()

            preds = torch.argmax(logits, dim=1)
```

```
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    # Hitung semua metrik dengan fokus pada weighted, macro, dan micro
    accuracy = accuracy_score(all_labels, all_preds)
    balanced_accuracy = balanced_accuracy_score(all_labels, all_preds)
    precision_weighted = precision_score(all_labels, all_preds, average='weighted',
zero_division=0)
    precision_macro = precision_score(all_labels, all_preds, average='macro',
zero_division=0)
    precision_micro = precision_score(all_labels, all_preds, average='micro',
zero_division=0)
    recall_weighted = recall_score(all_labels, all_preds, average='weighted',
zero_division=0)
    recall_macro = recall_score(all_labels, all_preds, average='macro', zero_division=0)
    recall_micro = recall_score(all_labels, all_preds, average='micro', zero_division=0)
    f1_weighted = f1_score(all_labels, all_preds, average='weighted', zero_division=0)
    f1_macro = f1_score(all_labels, all_preds, average='macro', zero_division=0)
    f1_micro = f1_score(all_labels, all_preds, average='micro', zero_division=0)
    avg_loss = total_loss / len(val_loader)

    return {
        'accuracy': accuracy,
        'balanced_accuracy': balanced_accuracy,
        'precision_weighted': precision_weighted,
        'precision_macro': precision_macro,
        'precision_micro': precision_micro,
        'recall_weighted': recall_weighted,
        'recall_macro': recall_macro,
```

```
        'recall_micro': recall_micro,
        'f1_weighted': f1_weighted,
        'f1_macro': f1_macro,
        'f1_micro': f1_micro,
        'loss': avg_loss
    }

# Fungsi training dengan metrik terbaru
def train_model(hyperparams, train_df, val_df, le, experiment_id):
    print(f"\n{'='*50}")
    print(f"Experiment {experiment_id}: {hyperparams}")
    print(f"{'='*50}")

    # Setup data
    train_texts = train_df['input_text'].tolist()
    val_texts = val_df['input_text'].tolist()
    train_labels = le.transform(train_df['R201B'])
    val_labels = le.transform(val_df['R201B'])

    train_inputs = preprocess(train_texts, max_length=128)
    val_inputs = preprocess(val_texts, max_length=128)

    train_labels_tensor = torch.tensor(train_labels)
    val_labels_tensor = torch.tensor(val_labels)

    # DataLoader
    train_dataset = TensorDataset(train_inputs['input_ids'],
train_inputs['attention_mask'], train_labels_tensor)
```

```
val_dataset = TensorDataset(val_inputs['input_ids'], val_inputs['attention_mask'], val_labels_tensor)

    train_loader = DataLoader(train_dataset, batch_size=hyperparams['batch_size'], shuffle=True)

    val_loader = DataLoader(val_dataset, batch_size=hyperparams['batch_size'])


    # Model setup

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    bert_model = BertModel.from_pretrained("indobenchmark/indobert-base-p2")

    num_classes = len(le.classes_)

    model = IndoBERTClassifier(bert_model, num_classes, dropout=0.1)


    if torch.cuda.device_count() > 1:

        print(f"Using {torch.cuda.device_count()} GPUs!")

        model = nn.DataParallel(model)


    model.to(device)


    # Optimizer dan scheduler

    optimizer = AdamW(model.parameters(), lr=hyperparams['learning_rate'], weight_decay=0.01)

    total_steps = len(train_loader) * hyperparams['epochs']

    scheduler = get_linear_schedule_with_warmup(

        optimizer,

        num_warmup_steps=100,

        num_training_steps=total_steps

    )


    loss_fn = nn.CrossEntropyLoss()
```

```
# Training loop
best_f1 = 0.0
patience = 2
early_stop_counter = 0
history = {'train_loss': [], 'val_metrics': []}

start_time = time.time()

for epoch in range(hyperparams['epochs']):
    # Training
    model.train()
    total_train_loss = 0

    for batch in train_loader:
        input_ids, attention_mask, labels = [x.to(device) for x in batch]

        optimizer.zero_grad()
        logits = model(input_ids, attention_mask)
        loss = loss_fn(logits, labels)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()
        scheduler.step()

        total_train_loss += loss.item()

    avg_train_loss = total_train_loss / len(train_loader)
```

```
history['train_loss'].append(avg_train_loss)

# Validation
val_metrics = evaluate_model(model, val_loader, device)
history['val_metrics'].append(val_metrics)

print(f"Epoch {epoch+1}/{hyperparams['epochs']}")
print(f"  Train Loss: {avg_train_loss:.4f}")
print(f"  Val Loss: {val_metrics['loss']:.4f}")
print(f"  Val Accuracy: {val_metrics['accuracy']:.4f}")
print(f"  Val Balanced Accuracy: {val_metrics['balanced_accuracy']:.4f}")
print(f"  Val Precision (Weighted): {val_metrics['precision_weighted']:.4f}")
print(f"  Val Recall (Weighted): {val_metrics['recall_weighted']:.4f}")
print(f"  Val F1 (Weighted): {val_metrics['f1_weighted']:.4f}")
print(f"  Val F1 (Macro): {val_metrics['f1_macro']:.4f}")
print(f"  Val Precision (Micro): {val_metrics['precision_micro']:.4f}")
print(f"  Val Recall (Micro): {val_metrics['recall_micro']:.4f}")
print(f"  Val F1 (Micro): {val_metrics['f1_micro']:.4f}")

# Early stopping berdasarkan F1 weighted
if val_metrics['f1_weighted'] > best_f1:
    best_f1 = val_metrics['f1_weighted']
    # Save best model
    torch.save(model.state_dict(), f'best_model_exp_{experiment_id}.pth')
    early_stop_counter = 0
else:
    early_stop_counter += 1
```

```
            if early_stop_counter >= patience:
                print(f"Early stopping at epoch {epoch+1}")
                break

    training_time = time.time() - start_time

    # Load best model untuk evaluasi final
    model.load_state_dict(torch.load(f'best_model_exp_{experiment_id}.pth'))
    final_metrics = evaluate_model(model, val_loader, device)

    # --- Tambahan: Evaluasi pada test set ---
    test_texts = test_df['input_text'].tolist()
    test_labels = le.transform(test_df['R201B'])
    test_inputs = preprocess(test_texts, max_length=128)
    test_labels_tensor = torch.tensor(test_labels)
    test_dataset = TensorDataset(test_inputs['input_ids'], test_inputs['attention_mask'],
test_labels_tensor)
    test_loader = DataLoader(test_dataset, batch_size=hyperparams['batch_size'])
    test_metrics = evaluate_model(model, test_loader, device)
    # ----------------------------------------


    result = {
        'experiment_id': experiment_id,
        'hyperparams': hyperparams,
        'final_metrics': final_metrics,
        'test_metrics': test_metrics,  # Tambahan
        'best_f1': best_f1,
```

```
    'training_time': training_time,
    'epochs_trained': epoch + 1,
    'history': history
}

print(f"Final Results (Validation):")
print(f"  Accuracy: {final_metrics['accuracy']:.4f}")
print(f"  Balanced Accuracy: {final_metrics['balanced_accuracy']:.4f}")
print(f"  Precision (Weighted): {final_metrics['precision_weighted']:.4f}")
print(f"  Recall (Weighted): {final_metrics['recall_weighted']:.4f}")
print(f"  F1 (Weighted): {final_metrics['f1_weighted']:.4f}")
print(f"  F1 (Macro): {final_metrics['f1_macro']:.4f}")
print(f"  Training Time: {training_time:.2f}s")
print(f"Final Results (Test):")
print(f"  Accuracy: {test_metrics['accuracy']:.4f}")
print(f"  Balanced Accuracy: {test_metrics['balanced_accuracy']:.4f}")
print(f"  Precision (Weighted): {test_metrics['precision_weighted']:.4f}")
print(f"  Recall (Weighted): {test_metrics['recall_weighted']:.4f}")
print(f"  F1 (Weighted): {test_metrics['f1_weighted']:.4f}")
print(f"  F1 (Macro): {test_metrics['f1_macro']:.4f}")
print(f"  Precision (Micro): {final_metrics['precision_micro']:.4f}")
print(f"  Recall (Micro): {final_metrics['recall_micro']:.4f}")
print(f"  F1 (Micro): {final_metrics['f1_micro']:.4f}")
print(f"  Precision (Micro): {test_metrics['precision_micro']:.4f}")
print(f"  Recall (Micro): {test_metrics['recall_micro']:.4f}")
print(f"  F1 (Micro): {test_metrics['f1_micro']:.4f}")
```

Script uji prediksi KBLI GPT-4.1

```
import openai
import pandas as pd
import time

# SETUP
openai.api_key = " "  # Ganti dengan API key Anda
MODEL = "gpt-4.1"  # atau "gpt-4" jika tersedia

# Baca dataset
df = pd.read_csv("kbli_dataset_v2_test.csv")

# Fungsi untuk menanyakan kode KBLI ke OpenAI
def ask_kbli(input_text):
    prompt = f"""
Berdasarkan deskripsi berikut, berikan hanya kode KBLI yang paling sesuai (hanya kode, tanpa penjelasan):

"{input_text}"

Jawab hanya dengan kode KBLI (misal: 56101).
"""
    try:
        response = openai.chat.completions.create(
            model=MODEL,
            messages=[
                {"role": "system", "content": "Kamu adalah asisten yang ahli dalam klasifikasi KBLI. Jawab hanya dengan kode KBLI yang paling sesuai."},
                {"role": "user", "content": prompt}
```

```
            {"role": "user", "content": prompt}
        ],
        max_tokens=10,
        temperature=0
    )
    answer = response.choices[0].message.content.strip()
    # Ambil hanya angka (kode KBLI) dari jawaban
    answer = ''.join(filter(str.isdigit, answer))
    return answer
  except Exception as e:
    print(f"Error: {e}")
    return None

# Looping dan evaluasi
correct = 0
total = 0
results = []

for idx, row in df.iterrows():
  input_text = row['input_text']
  true_kbli = str(row['R201B']).strip()
  pred_kbli = ask_kbli(input_text)
  is_correct = pred_kbli == true_kbli
  results.append({
    "input_text": input_text,
    "true_kbli": true_kbli,
    "pred_kbli": pred_kbli,
    "is_correct": is_correct
  ``
```

```
    })
    total += 1
    if is_correct:
        correct += 1
    print(f"[{idx+1}/{len(df)}] True: {true_kbli}, Pred: {pred_kbli}, Correct: {is_correct}")
    time.sleep(1.2)  # Hindari rate limit

# Akurasi
accuracy = correct / total * 100
print(f"\nAkurasi ChatGPT dalam memprediksi kode KBLI: {accuracy:.2f}%")

# Simpan hasil jika perlu
pd.DataFrame(results).to_csv("kbli_gpt_results.csv", index=False)
```