Dita Dwi Sapitri – 24917034
Tugas Akhir Analitik Teks

Sumber Data : Kaggle "Locknlock Reviews on Tokopedia"

Dataset ini berisi review pembeli produk lock n lock di Tokopedia. Dataset ini berisi 2 file yaitu TokopediaLocknLock.csv berisi sekitar 10,276 review dan TokopediaLocknLock3k.csv berisi sekitar 3,000 review. Dalam tugas ini digunakan data TokopediaLocknLock.csv yang berisi 10,276 review.

```
df = pd.read_csv("/content/sample_data/TokopediaLocknLock.csv")
df = df.rename(columns={df.columns[0]: "text"})

print("Jumlah data mentah:", len(df))
df.head()
```

```
Jumlah data mentah: 10276
```

	text
0	bahan botol tipis. setauku pet 1 hanya diguna...
1	Materialnya bagus, tahan lama, ga gampang rusa...
2	packing aman ya dan botol ada plastik pembungk...
3	kualitas top. model easy grip sangat OK. packi...
4	Kaget pas datang paketnya dalam dus gede bange...

Data Preprocessing dan Tokenisasi

Data Preprocessing

```
#Hapus data kosong & duplikat
df = df.dropna(subset=["text"])
df = df.drop_duplicates()
```

```
def clean_text(text):
    text = str(text).lower()                    # lowercase
    text = re.sub(r"http\S+", "", text)         # hapus URL
    text = re.sub(r"@\w+", "", text)            # hapus mention
    text = re.sub(r"#\w+", "", text)            # hapus hashtag
    text = re.sub(r"[^a-zA-Z\s]", "", text)     # hapus simbol & angka
    text = re.sub(r"\s+", " ", text).strip()    # hapus spasi berlebih
    return text

#Filtering teks terlalu pendek
df = df[df["text"].str.len() >= 5]

print("Jumlah data setelah preprocessing:", len(df))
```

```
Jumlah data setelah preprocessing: 10258
```

Dalam pembersihan data, dilakukan menghilangkan data kosong dan duplikat. Selain itu juga dilakukan mengubah data menjadi lowercase, pembersihan URL, mention, hastag, karakter non-alfabet (simbol & angka) dan spasi berlebih. Hasil ulasan dengan teks terlalu pendek juga dihilangkan.

```python
# Simpan versi asli
df["raw_text"] = df["text"]

# Terapkan preprocessing ke kolom baru
df["clean_text"] = df["text"].astype(str).apply(clean_text)

# Membandingkan sebelum dan sesudah preprocessing
sample = df.sample(5, random_state=42)

for i, row in sample.iterrows():
    print("SEBELUM :", row["raw_text"])
    print("SESUDAH :", row["clean_text"])
    print("-" * 80)
```

```
SEBELUM : tutupnya kurang pas/tidak presisi
SESUDAH : tutupnya kurang pastidak presisi
--------------------------------------------------------------------------------
SEBELUM : manthap, sesuai ekspektasi
SESUDAH : manthap sesuai ekspektasi
--------------------------------------------------------------------------------
SEBELUM : produk bagusMaterial Produk  Bagus. Kualitas Produk Bagus. Warna Produk Sesuai. Ukuran Produk Sesuai.
SESUDAH : produk bagusmaterial produk bagus kualitas produk bagus warna produk sesuai ukuran produk sesuai
--------------------------------------------------------------------------------
SEBELUM : Produk ori, material OK, tahan panasnya manteb. Kata Bapak saya nyeduh kopi abis Shubuhan klo ditutup gelasnya tahan ampe Maghrib. 🤣👍🏻
SESUDAH : produk ori material ok tahan panasnya manteb kata bapak saya nyeduh kopi abis shubuhan klo ditutup gelasnya tahan ampe maghrib
--------------------------------------------------------------------------------
SEBELUM : bagus banget tumbler nya, build quality mantab..respon penjual cepat
SESUDAH : bagus banget tumbler nya build quality mantabrespon penjual cepat
--------------------------------------------------------------------------------
```

Berikut perbandingan data sebelum dan sesudai dilakukan data preprocessing.

```python
#Pemberian Label
def label_sentiment(text):
    positive_words = ["bagus", "mantap","murah","baik","awet","bonus","terima kasih","amanah","promo","puas","gratis","cocok", "asli","keren","sesuai","c
    negative_words = ["jelek","tipis","bau","sobek","lecet","lambat","parah","debu","kotor","plastik", "rusak", "kecewa","cacat", "salah","buruk", "retak

    pos = sum(w in text for w in positive_words)
    neg = sum(w in text for w in negative_words)

    if pos > neg:
        return "positive"
    elif neg > pos:
        return "negative"
    else:
        return "neutral"

df["label"] = df["text"].apply(label_sentiment)

print("\nDistribusi label:")
print(df["label"].value_counts())
```

```
Distribusi label:
label
positive    7054
neutral     2497
negative     707
Name: count, dtype: int64
```

```python
le = LabelEncoder()
df["label_id"] = le.fit_transform(df["label"])
```

Karena data yang ada belum ada pengklasifian label sentimen, maka dilakukan proses pelabelan untuk membedakan ulasan tersebut adalah positif, negatif atau netral hingga mendapatkan hasil positif sebanyak 7054, negative sebanyak 707 dan netral sebanyak 2497. Dimana diketahui

Distribusi label setelah optimasi label positive 68.8 %, neutral 24.3% dan negative 6.9%.

Kemudian hasil pelabelan diubah menjadi angka agar dapat digunakan pada model dengan fungsi LabelEncoder.

```
texts = df["clean_text"]
labels = df["label"]

X_train, X_temp, y_train, y_temp = train_test_split(
    texts, labels,
    test_size=0.30,
    random_state=42,
    stratify=labels
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp,
    test_size=0.50,
    random_state=42,
    stratify=y_temp
)

train_texts = X_train
val_texts = X_val
test_texts = X_test

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)

label_map = {idx: label for idx, label in enumerate(label_encoder.classes_)}

# Assign encoded labels back to y_train, y_val, y_test for subsequent steps
y_train = y_train_encoded
y_val = y_val_encoded
y_test = y_test_encoded
```

Dari data yang sudah di preprocessing tadi, kemudian dibagi menjadi data training, validation dan testing dengan pembagian 70:15:15. Pembagian ini terlihat dengan test_size = 0.30, artinya data training adalah 70%. Kemudian test_size=0.50 artinya membagi data menjadi 2 untuk data testing dan validasi. Hasilnya, data Training sebanyak 7180, Testing sebanyak 1539 dan Testing sebanyak 1539.

Tokenisasi

```
#Tokenisasi indoBert
model_name = "indobenchmark/indobert-base-p1"
tokenizer = AutoTokenizer.from_pretrained(model_name)

train_enc = tokenizer(list(train_texts), truncation=True)
val_enc   = tokenizer(list(val_texts), truncation=True)
test_enc  = tokenizer(list(test_texts), truncation=True)
```

Model yang digunakan adalah indoBERT. Setelah membagi data menjadi data training, validasi dan testing, kemudian dilakukan tahap tokenisasi yang berguna untuk mengubah teks mentah menjadi token ID yang dimengerti oleh model. Proses ini dilakukan karena Model Transformer tidak bisa langsung membaca teks mentah. Hasil tokenisasi akan dipakai untuk PyTorch Dataset dan dimasukkan ke model saat training/evaluasi.

```
class ReviewDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels.tolist()

    def __getitem__(self, idx):
        item = {k: torch.tensor(v[idx]) for k, v in self.encodings.items()}
        item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = ReviewDataset(train_enc, y_train)
val_dataset   = ReviewDataset(val_enc, y_val)
test_dataset  = ReviewDataset(test_enc, y_test)
```

Untuk membaca token dan label dari tokenisasi, dataset dibuat. Dalam proses ini dilakukan;

- __init__ Inisialisasi dataset, dengan menyimpan data encodings (hasil tokenisasi), labels (numerik hasil LabelEncoder) dan memastikan label menjadi list sehingga bisa diakses per index.
- __getitem__ mengambil satu data untuk training.
- _len__ untuk memberi tahu PyTorch berapa banyak data dalam dataset aga DataLoader tahu berapa batch yang bisa dibuat.

Kemudian membuat objek dataset yang siap digunakan dengan pembagian train_dataset digunakan untuk fine tuning, val_dataset digunakan untuk evaluasi selama training dan test_dataset digunakan untuk evaluasi akhir.

Handling Imbalance

```
#HHitung class weight
class_weights = compute_class_weight(
    class_weight="balanced",
    classes=np.unique(y_train),
    y=y_train
)

class_weights = torch.tensor(class_weights, dtype=torch.float)
print("Class weights:", class_weights)
```

```
Class weights: tensor([4.8350, 1.3692, 0.4848])
```

Kode ini menghitung bobot untuk setiap kelas agar model tidak bias terhadap kelas yang lebih banyak (imbalanced classes). Proses diterapkan saat training karena memengaruhi cara model dalam memperbaiki error untuk tiap kelas.

class_weight="balanced" artinya dilakukan menghitung bobot otomatis agar kelas minoritas memiliki bobot lebih besar. Proses ini dilakukan untuk memberikan penekanan lebih pada kelas minoritas agar model tidak bias ke kelas yang lebih banyak karena seperti review Tokopedia, jumlah review positif biasanya jauh lebih banyak daripada negatif atau netral.

Fine Tuning

Fine-tuning adalah proses menyesuaikan model yang sudah dilatih sebelumnya (pretrained) dengan dataset atau tugas spesifik yang ingin diselesaikan. Dalam proses ini, menggunakan model indoBert yang sudah pernah dilatih sebelumnya pada dataset besar dan umum.

```
#model + LoRA PEFT
model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=len(le.classes_)
)

lora_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=8,
    lora_alpha=32,
    lora_dropout=0.1
)

model = get_peft_model(model, lora_config)
model.print_trainable_parameters()
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at indobenchmark/indobert-base-p1 and are newly initialized:
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
trainable params: 297,219 || all params: 124,740,870 || trainable%: 0.2383
```

Code ini adalah implementasi fine-tuning IndoBERT dengan LoRA (PEFT). LoRA + PEFT dipilih karena memungkinkan fine-tuning IndoBERT secara efisien, hemat memori, cepat, dan tetap efektif untuk dataset ulasan Tokopedia yang relatif kecil. Dalam code ini menyesuaikan layer output dengan jumlah kelas sentimen (positif, netral dan negatif) yang sudah diberi label tadi sebelumnya.

LoRA (Parameter-Efficient Fine-Tuning) adalah teknik fine tuning efisien,yang hanya melatih parameter tambahan kecil daripada semua bobot model. Diketahui rank low-rank matriks LoRA (bobot adaptasi) adalah 8, skala pembobotan LoRA (lora_alpha=32) dan dropout untuk mencegah overfitting pada parameter LoRA adalah 0.1.

Dalam code ini, hanya parameter LoRA yang dilatih sedangkan bobot asli BERT tetap frozen untuk hemat memori dan mempercepat training.

```
class WeightedTrainer(Trainer):
    def compute_loss(self, model, inputs, return_outputs=False, num_items_in_batch=None):
        labels = inputs.pop("labels")
        outputs = model(**inputs)
        logits = outputs.logits
        loss_fn = torch.nn.CrossEntropyLoss(weight=class_weights.to(logits.device))
        loss = loss_fn(logits, labels)
        return (loss, outputs) if return_outputs else loss
```

Code ini membuat Trainer kustom untuk melatih model IndoBERT + LoRA dengan weighted loss, agar memperhitungkan imbalance kelas. Fungsi WeightedTrainer memungkinkan model memperhitungkan class imbalance saat training. Loss dihitung berat per kelas (class weight) , sehingga model belajar seimbang meskipun dataset tidak seimbang.

Hyperparameter Tuning

Hyperparameter adalah parameter yang ditentukan sebelum training dan tidak dipelajari dari data. Hyperparameter tuning adalah proses mencari kombinasi terbaik dari hyperparameter model agar performanya optimal pada tugas tertentu.

Tujuan Hyperparameter Tuning

- Meningkatkan performa model
- Menyeimbangkan bias dan variance untuk mencegah underfitting/overfitting.
- Memaksimalkan efisiensi untuk training lebih cepat, penggunaan memori optimal.

```
# Training Arguments
training_args = TrainingArguments(
    output_dir="./results",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=5,
    learning_rate=2e-5,
    logging_steps=50,
    save_strategy="no",
    report_to="none"
)

data_collator = DataCollatorWithPadding(tokenizer)
```

Dalam code ini menerapkannya dengan mengisikan seperti learning rate = 2e-5 yang bertujuan untuk mempengaruhi seberapa cepat LoRA belajar. Kemudian mengisikan num_train_epochs = 5 berapa lama model dilatih (sebanyak 5x). per_device_train_batch_size berarti jumlah sampel per batch saat training. Misal jika per_device_train_batch_size= 16, Dataset training = 1000 sampel dan num_train_epochs = 5
Maka Training: 1000 / 16 ≈ 63 batch per epoch → 5 epoch → 63 × 5 = 315 update LoRA. Begitupun juga perhitungannya dengan per_device_val_batch_size.

```
# Fungsi evaluasi
def compute_metrics(eval_pred):
    logits, labels = eval_pred.predictions, eval_pred.label_ids
    preds = np.argmax(logits, axis=-1)
    acc = accuracy_score(labels, preds)
    f1 = f1_score(labels, preds, average="weighted")
    return {"accuracy": acc, "f1": f1}
```

Code ini mendefinisikan fungsi evaluasi untuk digunakan oleh Hugging Face Trainer saat menghitung performa model pada dataset validation atau test. Fungsi ini mengambil logits dan label dari output Trainer, menentukan prediksi kelas (argmax) kemudian menghitung accuracy dan weighted F1-score. F1-score digunakan untuk memperhitungkan jumlah sampel tiap kelas. Jika dataset tidak seimbang, sehingga F1-score minoritas tetap diperhitungkan. Kemudian Trainer akan menampilkan dictionary sebagai hasil evaluasi.

```
trainer = WeightedTrainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=data_collator,
    compute_metrics=compute_metrics
)
```

Dalam code ini memiliki fungsi untuk melatih model IndoBERT + LoRA dengan beberapa fitur tambahan, termasuk weighted loss dan evaluasi custom.

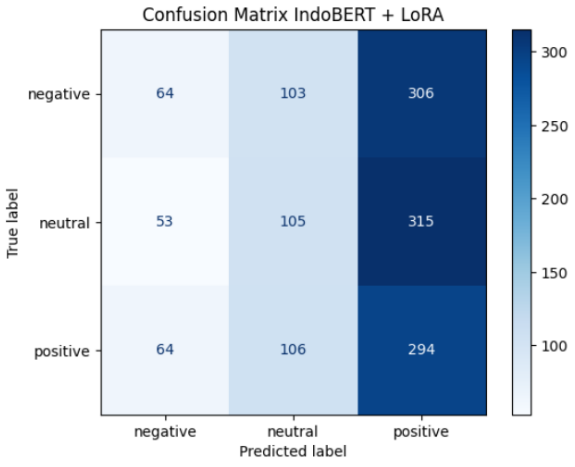
Dalam code ini sudah, Trainer sudah mengetahui model mana yang dilatih, pengunaan dataset untuk training dan validation, Hyperparameter (batch size, epochs, learning rate), weighted loss,metrik evaluasi yang di dapat dari proses sebelumnya.

Kemudian dilakukan proses training dari Hugging Face Trainer setelah fine-tuning IndoBERT + LoRA yang menghasilkan ;

```
TrainOutput(global_step=2245, training_loss=0.7713294005871881, metrics={'train_runtime': 135.2226, 'train_samples_per_second': 265.488,
'train_steps_per_second': 16.602, 'total_flos': 608038531816464.0, 'train_loss': 0.7713294005871881, 'epoch': 5.0})
```

Total waktu training adalah 135.22 detik, dengan kecepatan training 265.49 sampel/s. Training selesai selama 5 epoch dengan train_loss (rata-rata kesalahan model pada dataset training) sebanyak 0.7713. Karena dataset teks ulasan dan memiliki **3 kelas** dan jumlah data minoritas tidak seimbang, prediksi model cukup baik.

Confusion Matrix



Dari hasil Confusion Matrix, diketahui

1. Negative class (baris pertama)
   Dari total 64+103+306 = 473 sampel negative :
   Model prediksi negative sebanyak 64 = benar
   Model prediksi neutral sebanyak 103 = salah

Model prediksi positive sebanyak 306 = salah
Kesimpulan**:** model sulit membedakan negative dari positive, banyak false positive.

2. Neutral class (baris kedua)

   Dari total 53+105+315 = 473 sampel neutral:
   Model prediksi neutral sebanyak 105 = benar
   Model Prediksi negative sebanyak 53 = salah
   Model Prediksi positive sebanyak 315 = salah
   Kesimpulan: neutral juga sering salah klasifikasi sebagai positive.
3. Positive class (baris ketiga)
   Dari total 64+106+294 = 464 sampel positive:
   Model prediksi positive sebanyak 294 = benar
   Model Prediksi neutral sebanyak 106 = salah
   Model Prediksi negative sebanyak 64 = salah
   Kesimpulan: positive lebih sering dikenali, tapi masih ada kesalahan.

Model sangat bias ke prediksi positive karena sebagian besar sampel, termasuk negative dan neutral, diprediksi positive. Hal ini terlihat dari kolom positive yang paling gelap (306, 315, 294) sehingga tinggi, banyak prediksi positif. Akurasi untuk negative dan neutral rendah, sedangkan positive lebih tinggi.

Kemungkinan disebabkan Imbalanced dataset karena data positif lebih banyak sehingga model cenderung prediksi positive, Fitur teks kurang membedakan karena kata-kata negative & neutral mungkin mirip sehingga IndoBERT sulit membedakan dan Labeling rule based karena label neutral mungkin terlalu luas sehingga sulit bagi model belajar.