Pertarungan Paradigma: KLASIK vs. MODERN

WHICH IS BETTER?

The Modern or The Classical Method

ZERO-SHOT

Gemini API

Fine-Tuned Two-Stage

IndoBERT

Studi Kasus Klasifikasi Data Teks (Komentar) ke dalam 24 Kategori di SEVEN Retail Group

SEVEN

Pertarungan Paradigma: KLASIK vs. MODERN

Company Profile

SEVEN

SOZO dental

SOZO SKIN

Sparks
Education Made Easier

Golden Lamian

Kenapa tagging manual dihindari?

Lambat: Butuh waktu berminggu-minggu untuk membaca dan mengkategorikan.
Subjektif: Interpretasi bisa berbeda antar analis.
Tidak Real-time: Wawasan baru didapat saat sudah terlambat.

Database Sparks

Sparks English Databases

Data Komentar customer Itu Berharga loh ya!

Net Promoter Score Data

Nama Student, NPS Score, Komentar, Kategori, dll.

Harta Karun Terpendam

Berupa Insight akademik berbentuk inovasi produk dan layanan

Pertarungan Paradigma: KLASIK vs. MODERN

TUJUAN UTAMA:

Secara sistematis mengevaluasi dan membandingkan efektivitas dua metode AI terdepan untuk tugas klasifikasi komentar pelanggan Sparks English.

ZERO-SHOT
Gemini
API

Fine-Tuned
IndoBERT

Sasaran Spesifik:

- **Mengukur Kinerja:** Menganalisis kinerja kedua model secara kuantitatif (dengan metrik F1-Score, Precision, Recall) dan kualitatif (melalui studi kasus).
- **Menghasilkan Wawasan:** Mengidentifikasi topik utama yang dibicarakan pelanggan.
- **Memberikan Rekomendasi:** Memberikan rekomendasi berbasis data tentang pendekatan mana yang paling cocok untuk kebutuhan operasional Sparks English, dengan mempertimbangkan trade-off antara kinerja, biaya, dan kecepatan implementasi.

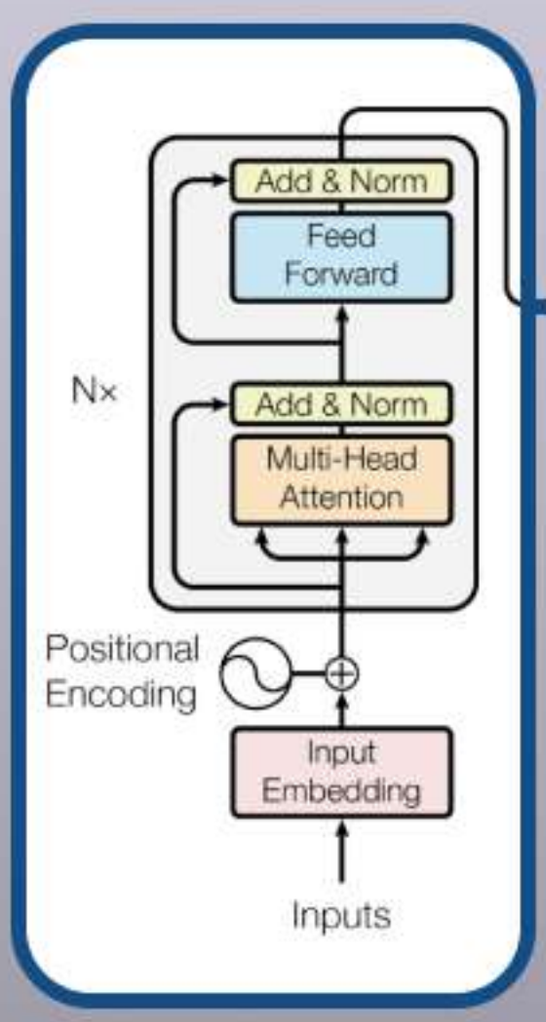
Fine-Tuned Two-Stage IndoBERT

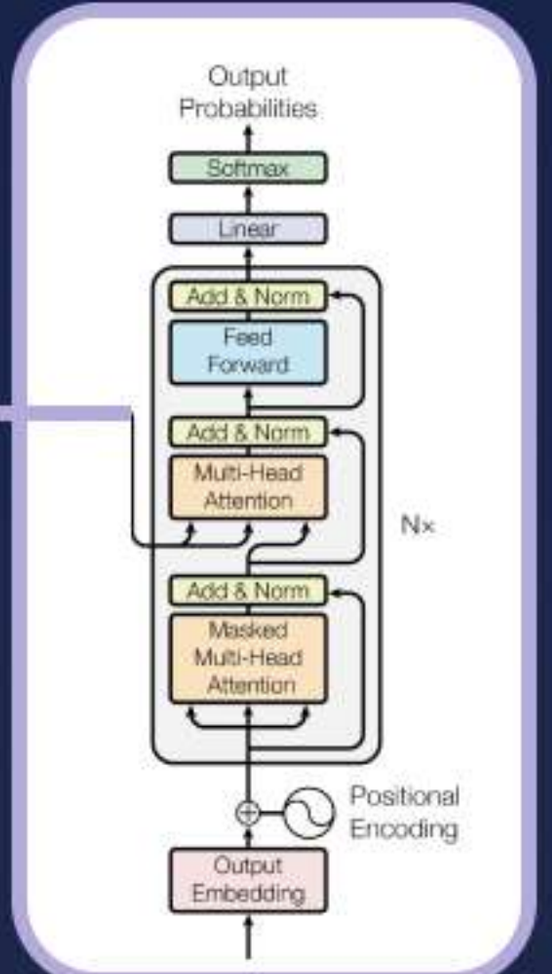
Pendekatan yang membangun dua model spesialis yang bekerja secara berurutan.

Tujuannya adalah memaksimalkan akurasi dengan memecah masalah kompleks menjadi dua subtugas yang lebih sederhana.

Membutuhkan fine-tuning dan kontrol penuh atas proses training.

BERT ENCODER





Gemini DECODER

Pendekatan yang memanfaatkan Large Language Model (LLM).

Tidak memerlukan training, melainkan menggunakan prompt engineering untuk melakukan klasifikasi secara langsung.

Menawarkan kecepatan dan fleksibilitas pengembangan yang luar biasa.

ZERO-SHOT Gemini API

Pertarungan Paradigma: KLASIK vs. MODERN

Arsitektur Two-Stage IndoBERT

Database Sparks

Sparks English Databases

4866 Comments

24 Categories

1st STAGE

Feedback Positive

1 Category

DONE

Suggestion

22 Categories

2nd STAGE

Uncategorized

1 Category

DONE

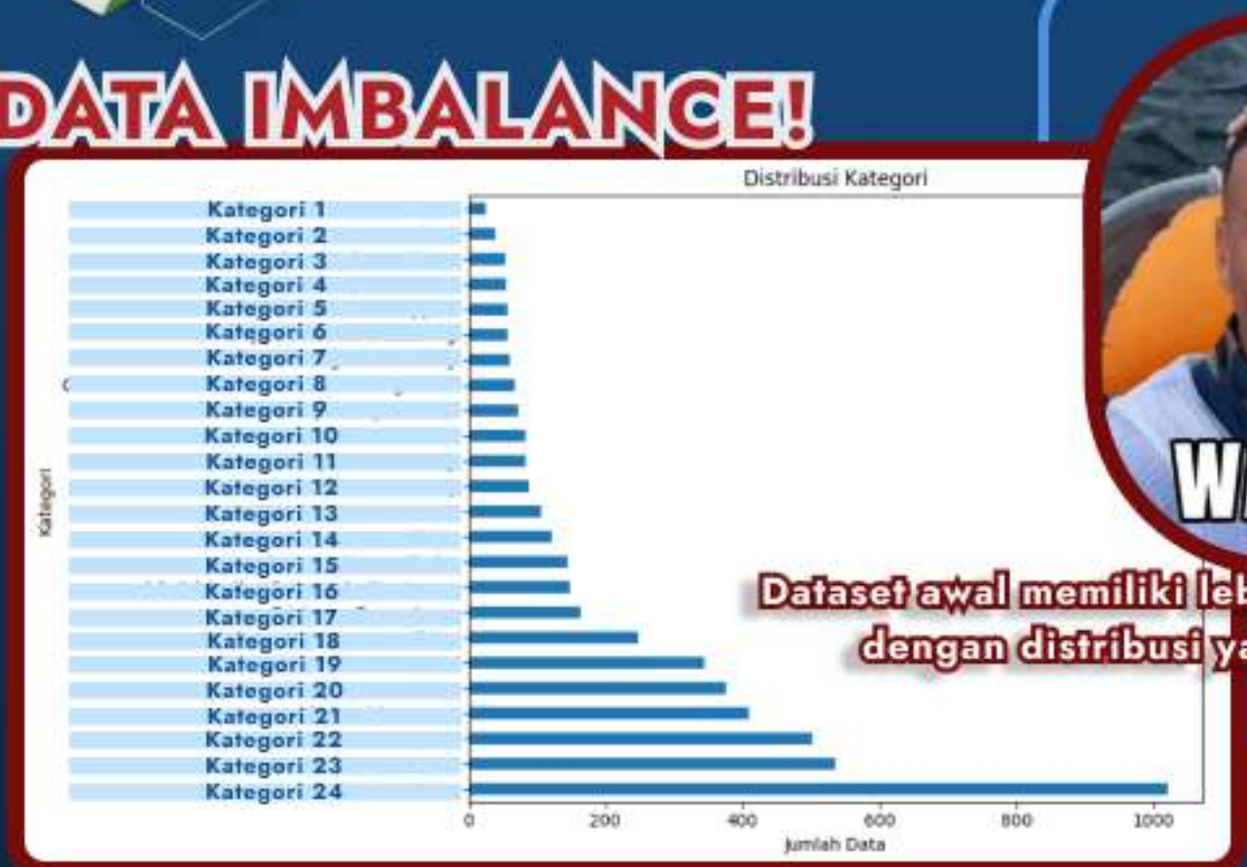
Category 1, Category 2, Category 3, ... , Category 22

DONE

Pertarungan Paradigma: KLASIK vs. MODERN

Data Problem gimana mengakalinya?

DATA IMBALANCE!



Dataset awal memiliki lebih dari 20 kategori dengan distribusi yang sangat timpang.

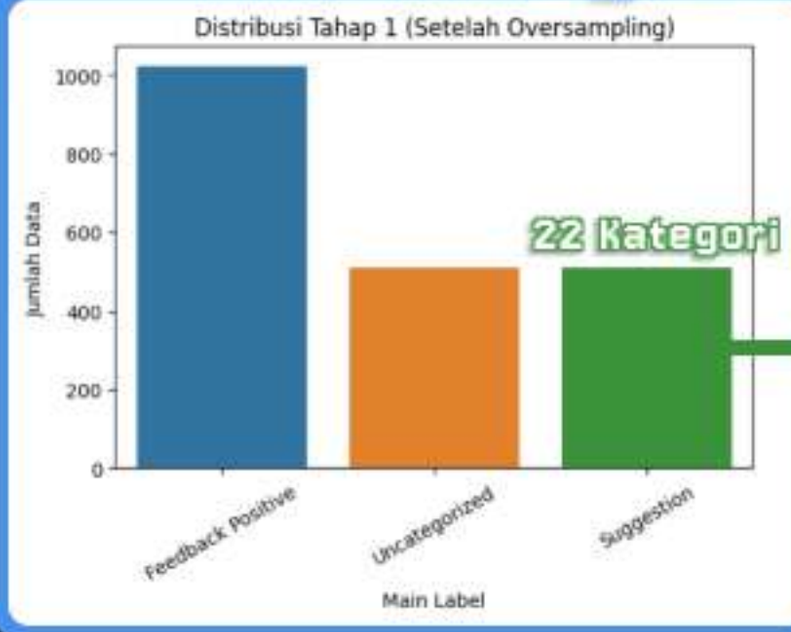
Menggandakan Data dengan **Oversampling & Undersampling**

24 Kategori

4866 Data

11770 Data

3 Kategori



22 Kategori

Pertarungan Paradigma: KLASIK vs. MODERN

Data Problem gimana mengakalinya?

Komentar Multilabel!

(Komentar ini hanya imitasi)

Tolong dong untuk Pak X ngajarnya lebih interaktif lagi, jangan terlalu kaku. Terus juga tolong toilet di lantai 2 dibersihkan yaa supaya anak lebih nyaman pas ke toilet.

Teacher Issue | Toilet Cleanliness

Komentar Dipecah Per-Kategori

Tolong dong untuk Pak X ngajarnya lebih interaktif lagi, jangan terlalu kaku

Teacher Issue

Tolong toilet di lantai 2 dibersihkan yaa supaya anak lebih nyaman pas ke toilet.

Toilet Cleanliness

Komentar pada data train yang memiliki multilabel dipecah ke masing-masing kategori untuk memperjelas konteks komentar saat model dilatih

Encoding Label

Sparks English NPS Comment Data

Fine-Tuned Two-Stage IndoBERT

Proses mengubah label kategori yang berupa teks (contoh: "Pujian Pengajar") menjadi representasi angka (contoh: 0).

Original Data

Team	Points
A	25
A	12
B	15
B	14
B	19
B	23
C	25
C	29

Label Encoded Data

Team	Points
0	25
0	12
1	15
1	14
1	19
1	23
2	25
2	29

Python Code:

```
[3]: label2id_main

[3]: {'Feedback Positive': 0, 'Suggestion': 1, 'Uncategorized': 2}

     + Code  + Markdown

     label2id_22

[4]: {'Additional Class/Schedule Request': 0,
      'Additional Operational Staff Support': 1,
      'Administrative Communication Issues': 2,
      'CCTV for Parents': 3,
      'Class Timing & Punctuality': 4,
      'Class Transition & Flow/mobilisasi Management': 5,
      'Crowded Waiting Room': 6,
      'Curriculum or Learning Method': 7,
      'Facility Safety Concern': 8,
      'Inadequate Classroom Facilities': 9,
      'Inadequate Common Facilities': 10,
      'Inadequate Prayer Room': 11,
      'Learning Distraction': 12,
      'Learning Material & Study Tools': 13,
      'Lesson Documentation Sharing': 14,
      'Parent Participation Flexibility': 15,
      'Parking Issue': 16,
      'Pricing or Promo Issue': 17,
      'Progress Report Clarity': 18,
      'Support for Home Learning': 19,
      'Teacher Issue': 20,
      'Toilet Cleanliness': 21}

     + Code  + Markdown
```

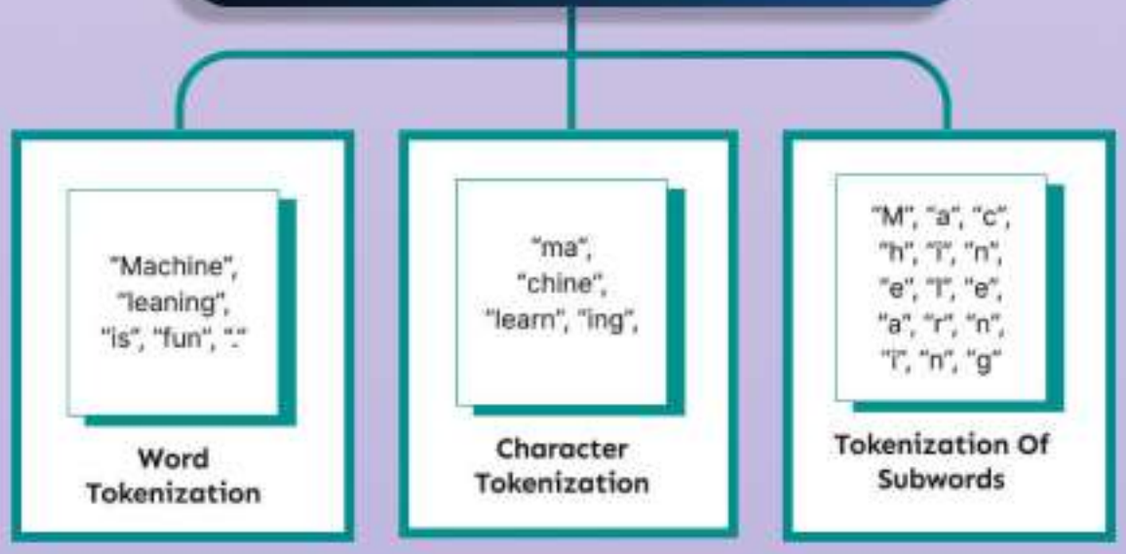
Data Tokenization

Sparks English NPS Comment Data

Fine-Tuned Two-Stage **IndoBERT**

Proses inti untuk mengubah teks komentar yang dapat dibaca manusia menjadi serangkaian angka yang dapat diproses oleh model **IndoBERT**. **tokenizer** digunakan spesifik dari model **indobenchmark/indobert-base-p1** untuk memastikan kesesuaian.

Tipe-Tipe Tokenisasi

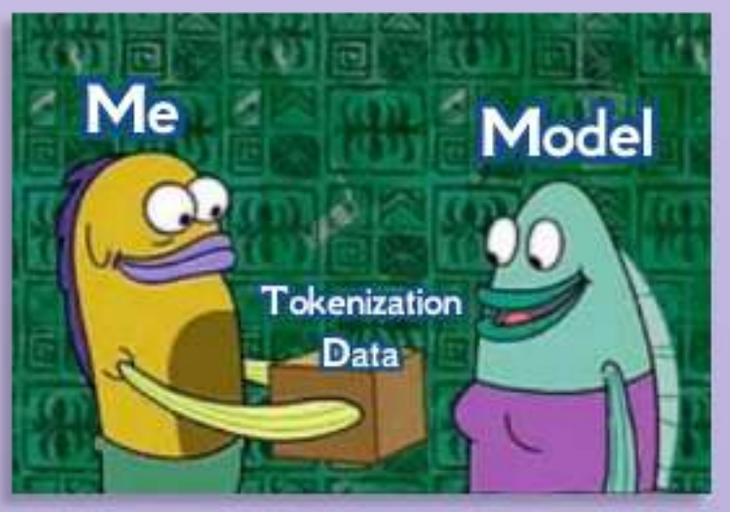


Python Code:

```python
# Tokenizer & model
model_name = "indobenchmark/indobert-base-p1"
tokenizer = AutoTokenizer.from_pretrained(model_name)

def tokenize_function(example):
    return tokenizer(example["Comment"], truncation=True)

tokenized_train = train_dataset.map(tokenize_function, batched=True)
tokenized_test = test_dataset.map(tokenize_function, batched=True)
```



	Comment	label	input_ids	token_type_ids	attention_mask
0	[redacted]	12	[2, 1604, 12519, 14790, 1899, 2198, 41, 119, 1...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
1	[redacted]	9	[2, 119, 176, 3]	[0, 0, 0, 0]	[1, 1, 1, 1]
2	[redacted]	8	[2, 3854, 4561, 4439, 30469, 4439, 34, 171, 27...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
3	[redacted]	11	[2, 991, 515, 3876, 26, 2828, 752, 310, 1057, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
4	[redacted]	7	[2, 12712, 176, 9508, 3744, 90, 436, 34, 1160,...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Train-Test Data Split

Sparks English NPS Comment Data

Fine-Tuned Two-Stage **IndoBERT**

"Untuk memastikan model dievaluasi secara adil, dataset dibagi menjadi dua bagian, seperti yang digambarkan. Sebagian besar data **(80%)** digunakan sebagai **Training Set**. Sisa datanya **(20%)** disimpan sebagai **Test Set**.

4866 Data

Upsampling & Downsampling

80% Train Data | 20% Test Data

Train-Test 1st Stage

```
df_train_main, df_test_main = train_test_split(
    df_balanced,
    test_size=0.2,
    random_state=42,
    stratify=df_balanced["label_main"]
)

print("Jumlah data train:", len(df_train_main))
print("Jumlah data test :", len(df_test_main))

trainer_main = train_model(
    df_train_main,
    df_test_main,
    label_col="label_main",
    num_labels=len(label_encoder_main.classes_),
    id2label=id2label_main,
    label2id=label2id_main,
    model_save_dir="./results_main"
)
```

```
Jumlah data train: 1632
Jumlah data test : 409
```

1632 Training Data

489 Testing Data

Train-Test 2nd Stage

```
df_train_multi, df_test_multi = train_test_split(
    df_multiclass_balanced,
    test_size=0.2,
    random_state=42,
    stratify=df_multiclass_balanced["label_22"]
)

print("Jumlah data train 2nd stage:", len(df_train_multi))
print("Jumlah data test 2nd stage:", len(df_test_multi))

trainer_multiclass = train_model(
    df_train_multi,
    df_test_multi,
    label_col="label_22",
    num_labels=len(label_encoder_22.classes_),
    id2label=id2label_22,
    label2id=label2id_22,
    model_save_dir="./results_multiclass"
)
```

```
Jumlah data train 2nd stage: 9416
Jumlah data test 2nd stage: 2354
```

9416 Training Data

2354 Testing Data

Pertarungan Paradigma: KLASIK vs. MODERN

Fine-Tuned Two-Stage

IndoBERT

Let's Get Deeper!

Pertarungan Paradigma: KLASIK vs. MODERN

Fungsi `train_model(`

Membuat sebuah alur kerja (pipeline) yang terstandarisasi dan dapat digunakan kembali (reusable) untuk melatih kedua model klasifikasi secara konsisten

Bagian 1: Input & Persiapan Data

Parameter Fungsi

```
def train_model(
    df_train,
    df_test,
    label_col,
    num_labels,
    id2label,
    label2id,
    model_save_dir
):
```

Fungsi ini dirancang untuk menjadi fleksibel dengan menerima beberapa parameter kunci yang memungkinkan kita menggunakannya untuk kedua tahap training.

- **df_train, df_test**: Data untuk training dan testing.
- **label_col**: Nama kolom yang berisi label target (label_main atau label_22).
- **num_labels, id2label, label2id**: Informasi spesifik tentang label untuk setiap model.
- **model_save_dir**: Lokasi untuk menyimpan model yang telah dilatih.

Tahap Persiapan & Pembersihan Data

1. Menyamakan nama kolom label menjadi **"label"** untuk konsistensi.
2. Menghapus baris data yang tidak memiliki komentar atau label **(dropna)**.
3. Memastikan tipe data kolom label adalah integer **(astype(int))**.
4. Mengubah Pandas DataFrame menjadi format Dataset **Hugging Face** yang dioptimalkan untuk training.

```
# Rename label column
df_train = df_train.rename(columns={label_col: "label"})
df_test = df_test.rename(columns={label_col: "label"})

# Bersihkan data
df_train = df_train.dropna(subset=["Comment", "label"]).reset_index(drop=True)
df_test = df_test.dropna(subset=["Comment", "label"]).reset_index(drop=True)
df_train["label"] = df_train["label"].astype(int)
df_test["label"] = df_test["label"].astype(int)

# Dataset HuggingFace
train_dataset = Dataset.from_pandas(df_train[["Comment", "label"]], preserve_index=False)
test_dataset = Dataset.from_pandas(df_test[["Comment", "label"]], preserve_index=False)
```

Kalkulasi Bobot Kelas

```
# Class weights
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(df_train['label']),
    y=df_train['label']
)
class_weights = torch.tensor(class_weights, dtype=torch.float)
```

Fungsi **compute_class_weight** digunakan dari Scikit-learn dengan argumen **class_weight='balanced'** untuk menghitung bobot ini secara otomatis.

Pertarungan Paradigma: KLASIK vs. MODERN

Fungsi `train_model(`

Bagian 2: Konfigurasi Model & Training

Membuat sebuah alur kerja (pipeline) yang terstandarisasi dan dapat digunakan kembali (reusable) untuk melatih kedua model klasifikasi secara konsisten

Inisialisasi Tokenizer & Model

```python
def tokenize_function(example):
    return tokenizer(example["Comment"], truncation=True)

tokenized_train = train_dataset.map(tokenize_function, batched=True)
tokenized_test = test_dataset.map(tokenize_function, batched=True)

model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=num_labels,
    id2label=id2label,
    label2id=label2id
)
```

- **Tokenizer**: Memuat **AutoTokenizer** dari **indobenchmark/indobert-base-p1**. Ini berfungsi sebagai "penerjemah" yang mengubah teks komentar menjadi token numerik yang dimengerti model.
- **Model**: Memuat **AutoModelForSequenceClassification**. Ini adalah arsitektur IndoBERT dengan tambahan "kepala" klasifikasi di atasnya. Kepala ini disesuaikan secara dinamis **(num_labels=num_labels)** untuk setiap tugas **(3 kelas atau 22 kelas)**.

Mengatur Training Arguments

```python
# TrainingArguments
training_args = TrainingArguments(
    output_dir=model_save_dir,
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=20,
    weight_decay=0.01,
    load_best_model_at_end=True,
    metric_for_best_model="eval_loss",
    greater_is_better=False,
    save_total_limit=1,
    logging_dir=f"{model_save_dir}/logs",
    logging_strategy="epoch",
    report_to="none"
)
```

Hyperparameter Kunci:

- **learning_rate**=2e-5: Seberapa besar langkah yang diambil model saat belajar. Nilai kecil ini umum untuk fine-tuning.
- **per_device_train_batch_size**=16: Berapa banyak sampel yang diproses bersamaan dalam satu waktu.
- **num_train_epochs**=20: Berapa kali model akan melihat keseluruhan data training.
- **weight_decay**=0.01: Teknik regularisasi untuk mencegah overfitting.

Strategi Terbaik:

- **load_best_model_at_end**=True: Memastikan bahwa model yang disimpan adalah versi terbaik berdasarkan kinerja pada data evaluasi, bukan hanya model dari epoch terakhir.

Pertarungan Paradigma: KLASIK vs. MODERN

Fungsi `train_model(`

Bagian 3: Eksekusi & Optimisasi

Membuat sebuah alur kerja (pipeline) yang terstandarisasi dan dapat digunakan kembali (reusable) untuk melatih kedua model klasifikasi secara konsisten

Accuracy: Persentase prediksi yang benar.
Precision: Dari yang diprediksi kategori X, berapa yang benar.
Recall: Dari yang seharusnya kategori X, berapa yang terdeteksi.
F1-Score: Rata-rata harmonik Precision & Recall, cocok untuk data tidak seimbang.

Inisialisasi Tokenizer & Model

```python
# Metrics
def compute_metrics(p):
    preds = torch.argmax(torch.tensor(p.predictions), axis=1)
    labels = torch.tensor(p.label_ids)
    return {
        "accuracy": accuracy_score(labels, preds),
        "precision": precision_score(labels, preds, average="macro", zero_division=0),
        "recall": recall_score(labels, preds, average="macro", zero_division=0),
        "f1": f1_score(labels, preds, average="macro")
    }
```

Custom Loss dengan WeightedTrainer

```python
# WeightedTrainer
class WeightedTrainer(Trainer):
    def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
        labels = inputs.get("labels")
        outputs = model(**inputs)
        logits = outputs.get("logits")
        device = next(model.parameters()).device
        loss_fct = nn.CrossEntropyLoss(weight=class_weights.to(device))
        loss = loss_fct(logits, labels)
        return (loss, outputs) if return_outputs else loss
```

Di dalam **compute_loss** yang baru, secara eksplisit digunakan **nn.CrossEntropyLoss** dan memasukkan **class_weights** yang telah kami hitung. Ini memastikan bahwa bobot tersebut benar-benar digunakan saat menghitung kesalahan model di setiap langkah training.

- **Data Collator**: **DataCollatorWithPadding** digunakan untuk secara cerdas menambahkan padding pada kalimat-kalimat dalam satu batch agar panjangnya sama. Ini lebih efisien daripada membuat semua kalimat di dataset memiliki panjang yang sama.
- **Early Stopping**: Callback ini memantau **eval_loss**. Jika loss pada data evaluasi tidak turun selama 2 epoch berturut-turut, training akan dihentikan secara otomatis untuk menghemat waktu dan mencegah overfitting.
- **Eksekusi**: Semua komponen (**model, args, dataset, metrics, callbacks**) digabungkan dalam **WeightedTrainer**, dan **trainer.train()** dipanggil untuk memulai proses.

Eksekusi Akhir Model

```python
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
early_stopping = EarlyStoppingCallback(early_stopping_patience=2)

trainer = WeightedTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    callbacks=[early_stopping]
)

trainer.train()
```

Pertarungan Paradigma: KLASIK *vs.* MODERN

ZERO-SHOT

Gemini API

Let's Get Deeper!

Pertarungan Paradigma: KLASIK vs. MODERN

Alur Kerja & Prompt Engineering

Perbedaan dengan **IndoBERT**

ZERO-SHOT Gemini API

Zero-SHOT Learning
model gemini-1.5-pro

Alur Kerja Pipeline:

Batch Processing

Untuk efisiensi dan menghindari limitasi **API**, data komentar diproses dalam batch (kelompok) berisi **10** komentar.

Koneksi Data

Skrip terhubung ke **Google Sheets** menggunakan gspread untuk mengambil data komentar secara real-time.

Prompt Engineering

Sebuah prompt dinamis dibuat untuk setiap batch, berisi instruksi, daftar kategori, dan data komentar yang akan diklasifikasikan.

Eksekusi & Parsing

Prompt dikirim ke **Gemini API**. Jawabannya kemudian dibersihkan (parsing) dan diubah menjadi format data yang terstruktur.

```
    prompt = f"""
Saya memiliki daftar komentar dari orang tua murid. Tugas Anda adalah mengklasifikasikan **setiap komentar** ke dalam satu atau lebih dari kategori berikut:

{kategori_str}

**Jika komentar kosong, tidak diisi, hanya simbol (-), atau serupa**, maka hasilkan: **[]** (list kosong).

Format keluaran HARUS Python list of lists, TANPA penjelasan tambahan:

[
  ["Feedback Positive"],
  [],
  ["Crowded Waiting Room"],
  ...
]

Jumlah komentar: {len(batch_df)}. Pastikan jumlah list juga {len(batch_df)}.

Komentar:
{chr(10).join(komentar_lines)}
""".strip()
```

Pertarungan Paradigma: KLASIK vs. MODERN

Technical & Robust Execution

ZERO-SHOT learning with GEMINI API

Komunikasi API & Rate Limiting

menambahkan time.sleep(5) setelah setiap permintaan ke Gemini API untuk memastikan kami tidak melebihi batas tersebut.

```
# ✅ Kirim ke Gemini dan parsing
try:
    response = model.generate_content(prompt)
    time.sleep(5)
```

Parsing & Validasi Jawaban

- **Pembersihan (Regex):** Menggunakan regular expression **(re.search)** untuk menemukan dan mengekstrak hanya bagian list of lists dari keseluruhan teks jawaban Gemini.
- **Konversi Aman (AST):** Menggunakan **ast.literal_eval()** untuk mengubah string list of lists menjadi objek Python asli. Ini jauh lebih aman daripada fungsi **eval()** biasa.

```
def bersihkan_jawaban(jawaban_mentah):
    cocok = re.search(r'\[\s*\[.*?\]\s*\]', jawaban_mentah, re.DOTALL)
    return cocok.group(0) if cocok else None
```

```
kategori_batch = ast.literal_eval(jawaban_bersih)
```

Pertarungan Paradigma: KLASIK vs. MODERN

Technical & Robust Execution

ZERO-SHOT learning with GEMINI API

Error Handling

Jika terjadi error, skrip akan mencetak pesan kesalahan dan secara otomatis memberikan label Uncategorized pada komentar yang gagal diproses, lalu melanjutkan ke batch berikutnya. Ini memastikan ketahanan (resilience) pipeline.

```
try:
    response = model.generate_content(prompt)
    time.sleep(5)
    jawaban = response.text.strip()
    jawaban_bersih = bersihkan_jawaban(jawaban)
    if not jawaban_bersih:
        raise ValueError("❌ Tidak ditemukan list valid di hasil Gemini.")
    kategori_batch = ast.literal_eval(jawaban_bersih)
except Exception as e:
    print(f"❌ Gagal parsing hasil Gemini untuk batch {batch_start}-{batch_start + batch_size}: {e}")
    kategori_batch = [[] if flag else ["Uncategorized"] for flag in komentar_kosong_flags]
```

Strukturisasi Data & Multi-Label

Desain prompt dan skrip ini secara inheren mendukung output multi-label, di mana satu komentar bisa masuk ke dalam beberapa kategori. Skrip akan membuat baris duplikat untuk setiap label yang diberikan.

```
    for kategori in kategori_list_hasil:
        if kategori not in kategori_list:
            kategori = "Uncategorized"
        row_copy = row.copy()
        row_copy['Category'] = kategori
        rows_expanded.append(row_copy)
```

Pertarungan Paradigma: KLASIK vs. MODERN

HOW'S THE RESULT?

ZERO-SHOT

Gemini API

Fine-Tuned Two-Stage

IndoBERT

Mana yang lebih baik?

Pertarungan Paradigma: KLASIK vs. MODERN

RESULT?

Mana yang Terbaik?

Fine-Tuned Two-Stage
IndoBERT

1st Stage Evaluation

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.496200	0.165351	0.948655	0.955982	0.939582	0.947191
2	0.077600	0.086929	0.970660	0.969136	0.972278	0.970684
3	0.027200	0.136613	0.968215	0.968871	0.967368	0.968110
4	0.004000	0.123365	0.975550	0.978632	0.972246	0.975282

Model ini mampu belajar dengan sangat cepat dan efektif, mencapai puncak **F1-Score 97.1%** dan **akurasi 97.1%**.

Metrik **Validation Loss** terendah dicapai pada **epoch ke-2**, menunjukkan bahwa model ini dapat membedakan kategori umum **(Positive, Suggestion, Uncategorized)** dengan sangat baik tanpa memerlukan training yang panjang.

97%
1st Stage

2nd Stage
98%

2nd Stage Evaluation

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	1.011200	0.251599	0.946049	0.947425	0.946049	0.945011
2	0.136300	0.120960	0.967290	0.967314	0.967290	0.966477
3	0.049500	0.110970	0.974087	0.974277	0.974087	0.973681
4	0.031500	0.109086	0.975786	0.975996	0.975786	0.975423
5	0.021400	0.098586	0.980459	0.980469	0.980459	0.980369
6	0.019100	0.096638	0.978335	0.978190	0.978335	0.978097
7	0.016700	0.102826	0.977910	0.978042	0.977910	0.977704
8	0.014300	0.104385	0.976211	0.976133	0.976211	0.976032

Pada tugas yang jauh lebih kompleks untuk mengklasifikasikan **22 topik spesifik**, model ini juga menunjukkan kinerja yang luar biasa dengan **F1-Score puncak ~98%** dan **akurasi ~98%**.

Kinerja terbaik dicapai antara **epoch ke-5 dan ke-6**, menunjukkan bahwa model spesialis ini berhasil mempelajari nuansa antar kategori topik dengan sangat detail.

Pertarungan Paradigma: KLASIK vs. MODERN

RESULT ?

Mana yang terbaik

ZERO-SHOT Gemini API

Hasil Kinerja Gemini API (Zero-Shot)

Untuk menguji kemampuan dari **Gemini**, dilakukan evaluasi pada **200 sampel** komentar baru yang belum pernah dilihat oleh model manapun.

Pendekatan ini murni **Zero-Shot**, artinya **tidak dilakukan fine-tuning** sama sekali, hanya menggunakan prompt yang telah dirancang sebelumnya. Hasil prediksi dari **Gemini API** kemudian dibandingkan langsung dengan kategori asli yang telah dilabeli secara manual.

Hasil Evaluasi

200 Komentar Baru

- **196** Komentar Benar
- **4** Komentar Salah

98% accuracy

Karena akurasi kedua metode sama sama sangat baik, mari kita lihat kekurangan dan kelebihannya

Next PAGE ⇒

Pertarungan Paradigma: KLASIK vs. MODERN

Fine-Tuned Two-Stage IndoBERT

Kelebihan Two-Stage IndoBERT

- **Kontrol Penuh:** Anda memiliki kendali penuh atas model, data, dan infrastruktur yang digunakan.
- **Kinerja Teroptimasi:** Dapat mencapai akurasi yang sangat tinggi karena dilatih secara spesifik pada data Anda.
- **Biaya Jangka Panjang:** Biaya per prediksi bisa menjadi sangat rendah pada volume penggunaan yang tinggi dan stabil.

Kekurangan Two-Stage IndoBERT

- **Waktu Pengembangan Lama:** Membutuhkan waktu signifikan untuk persiapan data, training, dan fine-tuning.
- **Kompleksitas & Perawatan:** Memerlukan perawatan berkelanjutan, infrastruktur, dan potensi training ulang jika ada perubahan data.
- **Biaya Infrastruktur:** Ada biaya tetap untuk server atau cloud GPU yang harus dibayar, terlepas dari jumlah penggunaan.

ZERO-SHOT Gemini API

Kelebihan Zero-Shot Gemini API

- **Kecepatan Implementasi:** Dapat diimplementasikan dalam hitungan jam atau hari, bukan minggu atau bulan.
- **Fleksibilitas Tinggi:** Sangat mudah untuk mengubah atau menambah kategori baru hanya dengan memodifikasi prompt.
- **Tidak Butuh Data Training:** Efektif bahkan saat Anda memiliki sedikit atau tanpa data berlabel sama sekali.

Kekurangan Zero-Shot Gemini API

- **Biaya Pay-Per-Use:** Biaya akan meningkat secara linear dengan jumlah permintaan, bisa menjadi sangat mahal pada volume tinggi.
- **Ketergantungan & Limitasi:** Anda bergantung pada layanan pihak ketiga, termasuk adanya batasan permintaan (rate limit) dan perubahan kebijakan.
- **Kontrol Terbatas:** Anda tidak memiliki kendali atas arsitektur atau versi model yang digunakan.