Topik : 4.4. Finalisasi Prototipe

Objective : Siapkan pipeline end-to-end: dataset → FL → DP → hasil

Task : Uji ulang semua modul, pastikan reproducible

1. Dataset (Load Data)

```python
# =========================================================
# 0) File input
# =========================================================
DINSOS_CSV   = "dinsos.csv"
DUKCAPIL_CSV = "dukcapil.csv"
KEMENKES_CSV = "kemenkes.csv"

# =========================================================
# 1) Load CSV
# =========================================================
def load_or_fail(path):
    p = Path(path)
    if not p.exists():
        raise FileNotFoundError(f"File tidak ditemukan: {path}")
    return pd.read_csv(p)

dinsos   = load_or_fail(DINSOS_CSV)
dukcapil = load_or_fail(DUKCAPIL_CSV)
kemenkes = load_or_fail(KEMENKES_CSV)
```

2. Labeling

```python
# =========================================================
# 2) Labeling rules
# =========================================================
def label_dinsos(row):
    jt = row.get("jumlah_tanggungan", np.nan)
    ph = row.get("penghasilan", np.nan)
    kr = row.get("kondisi_rumah", "")
    if pd.isna(jt): jt = 0
    if pd.isna(ph): ph = 0
    if ph < 2_000_000: return 1
    if (2_000_000 <= ph < 3_500_000) and jt >= 4: return 1
    if (kr in {"tidak layak","semi permanen","sangat sederhana"}) and (ph < 5_000_000): return 1
    if (jt >= 5) and (ph < 6_000_000): return 1
    if ph >= 8_000_000: return 0
    return 0

def label_dukcapil(row):
    u  = row.get("umur", np.nan)
    sp = row.get("status_pekerjaan", "")
    st = row.get("status_pernikahan", "")
    if pd.isna(u): u = 0
    if u > 65: return 1
    if sp in {"pengangguran","buruh","pekerja informal"}: return 1
    if (sp=="wirausaha") and (u>55): return 1
    if (st in {"cerai","janda","duda"}) and (sp!="pegawai tetap"): return 1
    if (sp=="pegawai tetap") and (25<=u<=55) and (st=="menikah"): return 0
    if (u<25) and (sp=="wirausaha"): return 0
    return 0

def label_kemenkes(row):
    rp = row.get("riwayat_penyakit", "")
    sg = row.get("status_gizi", "")
    t  = row.get("tinggi_cm", np.nan)
    b  = row.get("berat_kg", np.nan)
    bmi = None
    if pd.notna(t) and pd.notna(b) and t > 0:
        bmi = b / ((t/100.0) ** 2)
    if rp in {"kronis","jantung","asma","diabetes","disabilitas"}: return 1
    if sg in {"kurang","stunting","gizi buruk"}: return 1
    if (bmi is not None) and (bmi < 18.5): return 1
    if (bmi is not None) and ((bmi < 17) or (bmi > 35)): return 1
    return 0

# Apply label
dinsos["layak_subsidi"]   = dinsos.apply(label_dinsos, axis=1)
dukcapil["layak_subsidi"] = dukcapil.apply(label_dukcapil, axis=1)
kemenkes["layak_subsidi"] = kemenkes.apply(label_kemenkes, axis=1)
```

3. Preprocessing

```python
# ==========================================================
# 3) Preprocess (one-hot + min-max scaling global)
# ==========================================================
cat_cols_all = ["kondisi_rumah","status_pekerjaan","status_pernikahan","riwayat_penyakit","status_gizi"]
num_cols_all = ["jumlah_tanggungan","penghasilan","umur","tinggi_cm","berat_kg"]

for df in (dinsos, dukcapil, kemenkes):
    if {"tinggi_cm","berat_kg"}.issubset(df.columns):
        df["BMI"] = df["berat_kg"] / ((df["tinggi_cm"]/100.0) ** 2)
    else:
        df["BMI"] = np.nan
if "BMI" not in num_cols_all:
    num_cols_all.append("BMI")

def union_categories(series_list):
    cats = set()
    for s in series_list:
        if s is not None:
            for v in s.dropna().unique().tolist():
                if isinstance(v, str): cats.add(v)
    return sorted(list(cats))

global_vocabs = {
    "kondisi_rumah":     union_categories([dinsos.get("kondisi_rumah")]),
    "status_pekerjaan":  union_categories([dukcapil.get("status_pekerjaan")]),
    "status_pernikahan": union_categories([dukcapil.get("status_pernikahan")]),
    "riwayat_penyakit":  union_categories([kemenkes.get("riwayat_penyakit")]),
    "status_gizi":       union_categories([kemenkes.get("status_gizi")]),
}

def encode_and_scale(df):
    for c in num_cols_all:
        if c not in df: df[c] = np.nan
    for c in cat_cols_all:
        if c not in df: df[c] = np.nan
    oh_parts = []
    for col, vocab in global_vocabs.items():
        for v in vocab:
            oh_parts.append((f"{col}__{v}", (df[col] == v).astype(float)))
    oh_df = pd.DataFrame({n:a.values for n,a in oh_parts}, index=df.index) if oh_parts else pd.DataFrame(index=df.index)
    num_df = df[num_cols_all].copy()
    for c in num_cols_all:
        if not np.isfinite(num_df[c]).any(): num_df[c] = 0.0
        else: num_df[c] = num_df[c].fillna(num_df[c].median())
    X_raw = pd.concat([num_df, oh_df], axis=1)
    y = df["layak_subsidi"].astype(int).values
    return X_raw, y

X_dinsos_raw,   y_dinsos   = encode_and_scale(dinsos)
X_dukcapil_raw, y_dukcapil = encode_and_scale(dukcapil)
X_kemenkes_raw, y_kemenkes = encode_and_scale(kemenkes)

all_X = pd.concat([X_dinsos_raw, X_dukcapil_raw, X_kemenkes_raw], axis=0)
mins, maxs = all_X.min(axis=0), all_X.max(axis=0)
rng = (maxs - mins).replace(0, 1.0)
def scale_like_global(X): return (X - mins) / rng

X_dinsos   = scale_like_global(X_dinsos_raw).fillna(0.0)
X_dukcapil = scale_like_global(X_dukcapil_raw).fillna(0.0)
X_kemenkes = scale_like_global(X_kemenkes_raw).fillna(0.0)

FEATURE_COLS = list(X_dinsos.columns)
```

4. Dataset per klien

```python
# ==========================================================
# 4) Dataset per klien
# ==========================================================
LOCAL_EPOCHS = 1
BATCH_SIZE   = 64
DP_NUM_MICROBATCHES = BATCH_SIZE

def to_ds(X, y, local_epochs, shuffle=True):
    X = X.values.astype("float32")
    y = y.astype("float32").reshape(-1,1)
    ds = tf.data.Dataset.from_tensor_slices((X,y))
    if shuffle:
        ds = ds.shuffle(buffer_size=min(len(y), 2048), reshuffle_each_iteration=True)
    ds = ds.batch(BATCH_SIZE, drop_remainder=True).repeat(local_epochs).prefetch(tf.data.AUTOTUNE)
    return ds

client_data = [(X_dinsos, y_dinsos), (X_dukcapil, y_dukcapil), (X_kemenkes, y_kemenkes)]
client_sizes = [len(y_dinsos), len(y_dukcapil), len(y_kemenkes)]

def make_client_datasets(local_epochs, shuffle=True):
    return [to_ds(X, y, local_epochs, shuffle) for (X, y) in client_data]
```

5. Model

```python
# ==========================================================
# 5) Model
# ==========================================================
def build_model(input_dim):
    i = tf.keras.Input(shape=(input_dim,))
    x = tf.keras.layers.Dense(64, activation="relu")(i)
    x = tf.keras.layers.Dense(32, activation="relu")(x)
    o = tf.keras.layers.Dense(1, activation="sigmoid")(x)
    return tf.keras.Model(i, o)

def get_weights(model): return [w.copy() for w in model.get_weights()]
def set_weights(model, weights): model.set_weights([w.copy() for w in weights])

def evaluate_global(model):
    X_all = np.vstack([X_dinsos.values, X_dukcapil.values, X_kemenkes.values]).astype("float32")
    y_all = np.concatenate([y_dinsos, y_dukcapil, y_kemenkes]).astype("float32").reshape(-1,1)
    loss = tf.keras.losses.BinaryCrossentropy(from_logits=False)
    y_pred = model.predict(X_all, batch_size=256, verbose=0)
    l = float(loss(y_all, y_pred).numpy())
    acc = float(np.mean((y_pred >= 0.5) == y_all))
    return l, acc
```

6. Federated learning

```python
# ----------------------------------------------------------
# 6) Federated Training Loop (fungsi)
# ----------------------------------------------------------
def clip_by_l2_norm(tensors, clip):
    s = 0.0
    for t in tensors: s += np.sum(np.square(t))
    norm = float(np.sqrt(s)) + 1e-12
    if norm <= clip: return tensors, 1.0
    factor = clip / norm
    return [t * factor for t in tensors], factor

def federated_train(noise_multiplier, rounds=5):
    DP_CLIENT_LR = 0.05
    DP_L2_NORM_CLIP_CLIENT = 1.5
    SERVER_CLIP = 5.0
    SERVER_LR = 1.0

    tf.keras.utils.set_random_seed(42)
    np.random.seed(42)

    global_model = build_model(len(FEATURE_COLS))
    global_model.compile(optimizer="sgd", loss="binary_crossentropy", metrics=["accuracy"])
    global_weights = get_weights(global_model)

    acc_log, loss_log = [], []

    for round_idx in range(1, rounds+1):
        client_ds = make_client_datasets(local_epochs=LOCAL_EPOCHS, shuffle=True)
        clipped_weighted_sums = None
        total_weight = 0.0

        for k, ds in enumerate(client_ds):
            local_model = build_model(len(FEATURE_COLS))
            set_weights(local_model, global_weights)
            opt = DPKerasSGDOptimizer(
                l2_norm_clip=DP_L2_NORM_CLIP_CLIENT,
                noise_multiplier=noise_multiplier,
                num_microbatches=BATCH_SIZE,
                learning_rate=DP_CLIENT_LR,
                momentum=0.9
            )
            loss = tf.keras.losses.BinaryCrossentropy(from_logits=False, reduction=tf.losses.Reduction.NONE)
            local_model.compile(optimizer=opt, loss=loss, metrics=["accuracy"])
            local_model.fit(ds, epochs=1, verbose=0)

            w_local = local_model.get_weights()
            delta = [wl - wg for wl, wg in zip(w_local, global_weights)]
            delta_clipped, _ = clip_by_l2_norm(delta, SERVER_CLIP)

            weight_k = float(client_sizes[k])
            if clipped_weighted_sums is None:
                clipped_weighted_sums = [d * weight_k for d in delta_clipped]
            else:
                for i in range(len(delta_clipped)):
                    clipped_weighted_sums[i] += delta_clipped[i] * weight_k
            total_weight += weight_k

        avg_delta = [cw / (total_weight+1e-12) for cw in clipped_weighted_sums]
        avg_delta = [SERVER_LR * d for d in avg_delta]
        global_weights = [wg + d for wg, d in zip(global_weights, avg_delta)]
        set_weights(global_model, global_weights)

        gl_loss, gl_acc = evaluate_global(global_model)
        acc_log.append(gl_acc)
        loss_log.append(gl_loss)
        print(f"[nm={noise_multiplier:.3f}] Round {round_idx} | acc={gl_acc:.4f} | loss={gl_loss:.4f}")

    try:
        n = max(client_sizes)
        eps, _ = compute_dp_sgd_privacy_lib.compute_dp_sgd_privacy(
            n=n,
            batch_size=BATCH_SIZE,
            noise_multiplier=noise_multiplier,
            epochs=LOCAL_EPOCHS,
            delta=1e-5
        )
    except Exception:
        eps = np.nan

    return np.mean(acc_log), np.mean(loss_log), eps, acc_log, loss_log
```

7. Differential Privacy

```python
local_model = build_model(len(FEATURE_COLS))
set_weights(local_model, global_weights)
opt = DPKerasSGDOptimizer(
    l2_norm_clip=DP_L2_NORM_CLIP_CLIENT,
    noise_multiplier=noise_multiplier,
    num_microbatches=BATCH_SIZE,
    learning_rate=DP_CLIENT_LR,
    momentum=0.9
)
```

```python
ry:
    n = max(client_sizes)
    eps, _ = compute_dp_sgd_privacy_lib.compute_dp_sgd_privacy(
        n=n,
        batch_size=BATCH_SIZE,
        noise_multiplier=noise_multiplier,
        epochs=LOCAL_EPOCHS,
        delta=1e-5
    )
```

8. Evaluasi

```python
# ---------------------------------------------------------
# 7) Evaluasi Trade-off
# ---------------------------------------------------------
noise_list = [0.01, 0.05, 0.1, 0.5, 1.0]
results = []
rounds = 5

for nm in noise_list:
    acc, loss, eps, acc_log, loss_log = federated_train(noise_multiplier=nm, rounds=rounds)
    results.append({"NoiseMultiplier": nm, "Accuracy": acc, "Loss": loss, "Epsilon": eps})

df_results = pd.DataFrame(results)
print("\nHasil Evaluasi Trade-off:")
print(df_results)
```

9. Plot Trade-off

```python

# ---------------------------------------------------------
# 8) Plot Trade-off (Accuracy vs Epsilon)
# ---------------------------------------------------------
plt.figure(figsize=(7,5))
sns.lineplot(data=df_results, x="Epsilon", y="Accuracy", marker="o")
for i, row in df_results.iterrows():
    plt.text(row["Epsilon"], row["Accuracy"], f"nm={row['NoiseMultiplier']}", fontsize=9)

plt.xlabel("Epsilon (ε) → Privasi ↓")
plt.ylabel("Accuracy → Utility ↑")
plt.title("Trade-off Federated Learning (DP-SGD)")
plt.grid(True)
plt.tight_layout()
plt.savefig("tradeoff_fl_accuracy_epsilon.png", dpi=150)
print("Grafik trade-off disimpan sebagai tradeoff_fl_accuracy_epsilon.png")

```

Pipeline end-to-end :

1. Dataset (load data)

   Tahap pertama Adalah mengambil data mentah dari tiga client :

   - Dinsos → berisi informasi jumlah_tanggungan, penghasilan, kondisi rumah
   - Dukcapil → berisi informasi umur, status_pekerjaan, status_pernikahan
   - Kemenkes → berisi informasi Riwayat_penyakit, status_gizi, tinggi, berat

   Data ini di-*load* dari file CSV menggunakan pandas.read_csv(). Jika file tidak ada, program berhenti agar reproducibility terjaga.

2. Labeling

   Data mentah tidak memiliki target langsung. Maka dibuat aturan untuk menentukan label layak_subsidi (1 = layak, 0 = tidak layak).

   - Output tahap ini : kolom baru layak_subsidi pada tiap dataset

3. Pre-processing

   Agar model dapat dilatih :

   - Numerik : nilai yang hilang diisi mendian, lalu di-*scale* pakai min-max normalisasi global
   - Kategori : diubah menjadi one-hot encoding berdasarkan vocab gabungan (supaya konsisten antar klien)
   - Fitur Tambahan : BMI dihitung dari tinggi & berat badan

   Output : matriks fitur numerik dan vector label yang sudah bersih

4. Dataset per Klien

   Setiap sumber data dianggap sebagai client dalam Federated Learning.

   - Data tiap klien dikonversi ke tf.data.Dataset kemudian dilakukan batching, shuffling, repeat sesuai dengan local epochs
   - Tiga dataset klien siap dipakai dalam loop training

   Hal ini mensimulasikan kondisi nyata Dimana data tidak dikumpullkan disatu tempat, melainkan dilatih secara terdistribusi

5. Model

   Menggunakan arsitektur :

   - Input layer sesuai jumlah fitur
   - Hidden layer : 64 neuron (ReLU) → 32 neuron (ReLU)
   - Output layer : 1 neuron sigmoid (binary classification)

   Penerapan model yang cukup ringan sehingga cocok untuk melakukan federated learning.

6. Federated Learning

   Implementasi manual Federated Averaging (FedAvg) :

   - Tiap klien dimulai dari bobot global
   - Klien melatih model local pada datasetnya dengan optimizer DP-SGD
   - Delta bobot hasil training di-clip dengan L2 norm agar tidak terlalu besar

- Server melakukan aggregasi delta bobot dari semua klien (weighted average berdasarkan ukuran dataset
- Update bobot global → looping untuk beberapa round

7. Differential privacy

   Untuk menjamin privasi data tiap klien, maka digunakan :

- DP-SGD Optimizer → menambahkan *noise Gaussian* ke gradien, plus clipping gradien
- Menggunakan parameter penting
  - L2_norm_clip → batas clipping gradien
  - Noise_multiplier → besar noise yang ditambahkan
- Estimasi privasi dihitung dengan menggunakan compute_dp_sgd_privacy → menghasilkan nilai ε (epsilon), semakin kecil artinya privasi lebih kuat

8. Evaluasi

   Setelah setiap federated round :

- Model global dievaluasi pada gabungan semua data klien
- Metrics : accuracy dan loss
- Disimpan log per round agar bisa dianalisis konsistensinya
- Dicatat pula nilai rata – rata akurasi/loss untuk tiap noise multiplier

9. Plot Trade-off

   Hasil evaluasi divisualisasikan :

- Grafik Accuracy vs Epsilon (ε) → menunjukkan trade-off antara utility(akurasi) dan privacy **(ε)**.
- Tiap titik diberi label nm = noise_multiplier
- Grafik disimpan dalam file PNG agar reproducible