Données publiques

- Procédure d'entrainement de l'algorithme MAML sur données publiques

Définition des modelés de base

```
# ==========================
# 2️⃣ Modèle de base
# ==========================
def create_mlp(output_dim=None):
    if output_dim is None:
        output_dim = input_dim
    return models.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(64, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(output_dim, activation='linear')
    ])
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-1 : entrainement de l'algorithme MAML sur données publiques*

- On entraine l'algorithme MAML sur données publiques

```python
def sample_task(X, k_support=10, k_query=20):
    idx = np.random.permutation(len(X))
    return X[idx[:k_support]], X[idx[k_support:k_support+k_query]]

def maml_train(X_train, meta_epochs=5, inner_steps=3, k_support=10, k_query=20, meta_batch_size=5):
    meta_model = create_mlp()
    meta_optimizer = optimizers.Adam(1e-3)
    for epoch in range(meta_epochs):
        meta_grads = [np.zeros_like(var.numpy()) for var in meta_model.trainable_variables]
        for _ in range(meta_batch_size):
            X_supp, X_q = sample_task(X_train, k_support, k_query)
            X_supp_tf, X_q_tf = tf.convert_to_tensor(X_supp, dtype=tf.float32), tf.convert_to_tensor(X_q, dtype=tf.float32)

            task_model = create_mlp()
            task_model.set_weights(meta_model.get_weights())
            inner_opt = optimizers.SGD(0.01)

            for _ in range(inner_steps):
                with tf.GradientTape() as tape:
                    loss = tf.reduce_mean(tf.square(task_model(X_supp_tf) - X_supp_tf))
                grads = tape.gradient(loss, task_model.trainable_variables)
                inner_opt.apply_gradients(zip(grads, task_model.trainable_variables))

            with tf.GradientTape() as tape:
                query_loss = tf.reduce_mean(tf.square(task_model(X_q_tf) - X_q_tf))
            grads_query = tape.gradient(query_loss, task_model.trainable_variables)
            meta_grads = [mg + gq for mg, gq in zip(meta_grads, grads_query)]

        meta_optimizer.apply_gradients(zip(meta_grads, meta_model.trainable_variables))
        print(f"[MAML Epoch {epoch+1}] Query Loss = {query_loss.numpy():.4f}")
    return meta_model
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-2 : entrainement de l'algorithme MAML sur données publiques*

Dans cette partie on fait l'évaluation et l'exécution du modèle

```python
# ==========================
# 4️⃣ Évaluation
# ==========================
def eval_model(model, X_train, X_test, y_test):
    mse_test = np.mean(np.square(X_test - model.predict(X_test)), axis=1)
    threshold = np.percentile(np.mean(np.square(X_train - model.predict(X_train)), axis=1), 95)
    y_pred = (mse_test > threshold).astype(int)
    return f1_score(y_test, y_pred, average='macro'), recall_score(y_test, y_pred, average='macro')

# ==========================
# 5️⃣ Exécution
# ==========================
print("=== Entraînement MAML ===")
maml_model = maml_train(X_train)
print("=== Évaluation finale MAML ===")
f1, rec = eval_model(maml_model, X_train, X_test, y_test)
print(f"MAML -> F1: {f1:.4f}, Recall: {rec:.4f}")
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-3 : entrainement de l'algorithme MAML sur données publiques*

- Voici le résultat

```
=== Résultats Méta-Learning sur données p
MAML       -> F1: 0.9742, Recall: 0.9742
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-4 : résultat de l'algorithme MAML sur données publiques*

On répète la même procédure sur les données privées :

Données privées

```python
# ---------------------------
# Autoencoder
def build_autoencoder(input_dim, encoding_dim):
    inputs = layers.Input(shape=(input_dim,))
    e = layers.Dense(encoding_dim*2, activation='relu')(inputs)
    e = layers.Dense(encoding_dim, activation='relu')(e)
    bottleneck = layers.Dense(max(1,encoding_dim//2), activation='relu')(e)
    d = layers.Dense(encoding_dim, activation='relu')(bottleneck)
    d = layers.Dense(encoding_dim*2, activation='relu')(d)
    outputs = layers.Dense(input_dim, activation='linear')(d)
    return models.Model(inputs, outputs)
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-5 : entrainement de l'algorithme MAML sur données privées*

```python
# MAML class
class MAML:
    def __init__(self, input_dim, encoding_dim, lr=1e-3):
        self.model = build_autoencoder(input_dim, encoding_dim)
        self.opt = tf.keras.optimizers.Adam(lr)

    def fit(self, X_train, support, epochs=5):
        X_train_tf = tf.convert_to_tensor(X_train, dtype=tf.float32)
        support_tf = tf.convert_to_tensor(support, dtype=tf.float32)
        for epoch in range(epochs):
            # inner loop
            with tf.GradientTape() as tape:
                recon_s = self.model(support_tf, training=True)
                loss_s = tf.reduce_mean(tf.square(recon_s - support_tf))
            grads = tape.gradient(loss_s, self.model.trainable_variables)
            self.opt.apply_gradients(zip(grads, self.model.trainable_variables))

            recon_train = self.model(X_train_tf, training=False)
            loss_train = tf.reduce_mean(tf.square(recon_train - X_train_tf))
            print(f"[MAML] Epoch {epoch+1}/{epochs} - Train MSE: {loss_train.numpy():.6f}")

    def predict(self, X):
        return self.model.predict(X)
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-6 : entrainement de l'algorithme MAML sur données privées*

```
# --------------------------
# Support set
support_idx = np.random.choice(X_test_with_anom.shape[0], size=20, replace=False)
X_support = X_test_with_anom[support_idx]

# --------------------------
# Entraînement & évaluation
maml = MAML(input_dim, encoding_dim)
maml.fit(X_train, support=X_support, epochs=8)

recon_maml = maml.predict(X_test_with_anom)
mse_test = np.mean(np.square(X_test_with_anom - recon_maml), axis=1)
threshold = np.percentile(np.mean(np.square(X_train - maml.predict(X_train)), axis=1), 95)
y_pred = (mse_test > threshold).astype(int)

print("=== Résultats MAML ===")
print(f"F1: {f1_score(y_test, y_pred):.4f}, Recall: {recall_score(y_test, y_pred):.4f}")
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-7 : entrainement de l'algorithme MAML sur données privées*

```
{'MAML': (0.9357798165137614, 1.0)
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-8 : résultat de l'algorithme MAML sur données privées*