

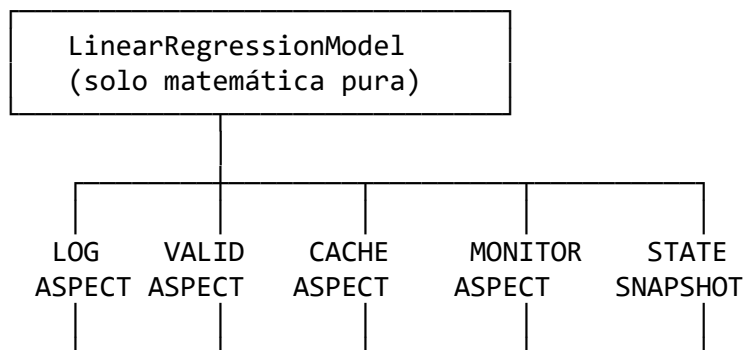
2. DISEÑO CONCISO: REGRESIÓN LINEAL CON ASPECTOS (AOP)

Javier Felipe Rosero Sandoval

Resumen

Regresión lineal separada de comportamiento transversal (logging, caché, validación, monitoreo) mediante 6 aspectos.

Arquitectura



CORE: LinearRegressionModel

Sin aspectos. Solo lógica matemática:

```
class LinearRegressionModel:
    w = 0.0
    b = 0.0

    def compute_gradient(x, y):
        y_pred = w * x + b
        error = y_pred - y
        dw = (2/m) * sum(error * x)
        db = (2/m) * sum(error)
        return (dw, db)

    def update_weights(dw, db):
        w -= learning_rate * dw
        b -= learning_rate * db

    def train_epoch():
```

```
(dw, db) = compute_gradient()
update_weights(dw, db)
```

Aspectos (6 Total)

1. LoggingAspect (AROUND)

Intercepta: entrada/salida de todos los métodos

```
@around('execution(compute_gradient) OR execution(update_weights)')
def log_execution(method, args):
    print(f"[INICIO] {method.__name__}({args})")
    result = proceed() # ejecuta método original
    print(f"[FIN] {method.__name__} → {result}")
    return result
```

Salida:

```
[INICIO] compute_gradient([1,2,3], [2,4,6])
[FIN] compute_gradient() → dw=-18.67, db=-8.0, mse=18.67
```

2. ValidationAspect (BEFORE)

Intercepta: antes de ejecutar (valida precondiciones)

```
@before('execution(train_epoch)')
def validate_before_train():
    assert X.size > 0, "X vacío"
    assert y.size > 0, "y vacío"
    assert X.size == y.size, "Dimensiones mismatch"
    assert learning_rate > 0, "LR debe ser positivo"
    assert not isnan(w) and not isnan(b), "NaN detectado"
```

Falla rápido: Si algo está mal, lanza excepción ANTES de ejecutar.

3. CachingAspect (AROUND)

Intercepta: predict() para caché transparente

```
cache = {} # {hash(x, w, b): resultado}
```

```
@around('execution(predict)')
def cached_predict(x):
    key = hash(x, w, b)

    if key in cache and not cache[key].expired():
        return cache[key].value # HIT

    result = proceed() # ejecuta predict original
```

```
cache[key] = (result, TTL=1s) # MISS
return result
```

Hit rate típico: 50-90% (según patrón de acceso)

4. PerformanceMonitoringAspect (AROUND + AFTER)

Intercepta: todos los métodos para recopilar métricas

```
@around('execution(* LinearRegressionModel.*)')
def monitor_execution(method):
    time_start = nanoTime()
    mem_before = used_memory()

    result = proceed()

    time_end = nanoTime()
    mem_after = used_memory()

    metrics = {
        'method': method.__name__,
        'time_ms': (time_end - time_start) / 1e6,
        'memory_delta': mem_after - mem_before,
        'timestamp': now()
    }
    metrics_queue.append(metrics)
    return result
```

Reporte cada 100 métodos:

```
compute_gradient: 50 calls, avg 0.234ms, max 2.156ms
update_weights: 50 calls, avg 0.0123ms
train_epoch: 10 calls, avg 28.5ms
```

5. StateSnapshotAspect (AFTER)

Intercepta: train_epoch() cada 10 épocas

```
@after('execution(train_epoch)')
def capture_state(epoch):
    if epoch % 10 == 0:
        snapshot = {
            'epoch': epoch,
            'w': current_w,
            'b': current_b,
            'mse': current_mse,
            'status': 'CONVERGING' if mse_improving else 'DIVERGING'
        }
        snapshots.append(snapshot)
        save_to_disk(snapshot)
```

Permite rollback a mejores pesos si divergencia se detecta.

6. AuditAspect (AFTER)

Intercepta: predict() para auditoría

```
@after('execution(predict)')
def audit_prediction(x, result):
    log_entry = {
        'action': 'PREDICT',
        'input': x,
        'output': result,
        'timestamp': now(),
        'user': current_user
    }
    audit_log.append(log_entry)
```

Integración de Aspectos

Cuando llamas: model.train_epoch(50)

1. ValidationAspect.before()
 - └ Valida X, y, w, b, learning_rate
2. PerformanceMonitoringAspect.around() INICIO
 - └ Captura time_start, mem_before
3. LoggingAspect.before()
 - └ Log: "[INICIO] train_epoch(50)"
4. --- CORE LOGIC ---
 - └ compute_gradient() [INTERCEPTADO]
 - └ ValidationAspect: valida batch
 - └ LoggingAspect: log entrada/salida
 - └ PerformanceMonitoringAspect: timing
 - └ update_weights() [INTERCEPTADO]
 - └ LoggingAspect: log nuevo w, b
 - └ PerformanceMonitoringAspect: memory delta
5. StateSnapshotAspect.after()
 - └ Si epoch % 10 == 0: guardar snapshot
6. LoggingAspect.after()
 - └ Log: "[FIN] train_epoch(50) - MSE=0.0234"
7. PerformanceMonitoringAspect.around() FIN
 - └ Calcula elapsed_ms, registra métrica

Ventajas

Ventaja	Descripción
Separación	Core limpio, aspectos independientes
Reutilizable	Aspectos aplicables a otros modelos
Mantenible	Cambiar logging = 1 archivo
Testeable	Mockear aspectos, probar core aislado
Observable	Logging + monitoreo automáticos
Auditable	Trazabilidad completa
