머신러닝 프로그램 테스팅하기

서울대학교 ROPAS

김지훈

Makefile 사용법

make -j8

make check

make install

- 컴파일 할때는 -jN옵션을 꼭 쓰세요! (N=CPU코어수)
- 컴파일 후에는 테스트를 해야 합니다.
- 머신러닝 라이브러리는 어떻게 테스트 하나요?

잠깐. 왜 정적분석은 안하고..?

머신러닝은:

- C 같은 언어를 많이 사용(OpenCL, ISPC, …)
- 분산 컴퓨팅(MPI, Hadoop, …)
- 근사 계산 : 수학적으로 검증하기 어려움

정적분석을 안한다기 보다는 추후에..?

테스트의 중요성

- 테스트 = 보이는 부분
  - 보는 만큼 압니다.
- 구현 목표 != 스펙
  - 사용자의 스팩 = 개발자의 구현 목표
  - 개발자의 스팩 = 유닛 테스트

테스트의 어려움

- 머신러닝 프로그램: 미지의 답을 구하기 위한 계산
- 답을 모르는데 테스트를 할 수 있나요?
  - 네. 잘 돌아가는 코드인지 보면 됩니다(?)
  - 사실 과학계산 분야에서 늘상 있는 문제

답이 없는 프로그램 테스트하기

- 같은 프로그램을 독립된 그룹들이 각자 구현 후 비교
- 쉬운 경우에 대하여 답을 제대로 구하나 확인
- 주먹구구 동적분석 : 우리의 친구 printf
- 답이 가져야 할 성질을 알아낸 후, 실제로 그런지 확인
  - Metamorphic testing

목표

- 머신러닝, 확률 추론을 위한 metamorphic testing
- 검증이 어려운 부분을 랜덤 테스팅으로 대체
- 가능한 경우에 대해서는 필요충분조건 테스팅 하기
  (metamorphic testing = 필요조건 테스팅)
- 재사용이 높은 테스팅

예제

- 머신러닝에서 자주 쓰이는 연산 유형:
  - 샘플링(예제1)
  - 벡터&행렬 연산(예제2)
  - 데이터 입출력(예제3)

예제1. 샘플링

```
//W = matrix
//x, b = vector
//p = P(x) ~ exp(x * W * x + b * x)
auto p = GaussianPDF{W, b};

    //Sample x.
    auto x = p();
```

- C++11
- N차원 정규분포 샘플링

정규분포 샘플링 테스팅

```
//Rewrite P(x) to P(y):
//P(x) ~ exp(x * W * x + b * x)
//     ~ exp(0.5* y * I * y)
//where W = R^T * D * R
//and y_i = (R_i * x - mu_i) / sig_i

//for each i:
y[i] = (R[i]*x - mu[i]) / sigma[i];
assert( y ~ Gauss{0,1});

//for each i,j:
auto theta ~ Uniform{0, 2*pi}();
z = sin(theta)*y[i] + cos(theta)*y[j];
assert( z ~ Gauss{0,1});
```

예제 2. 벡터 내적

```
export uniform float V1dotV2(
    const uniform float v1[], const uniform float v2[],
    uniform int count)
{
    float sum = 0.0;
    foreach (i = 0 ... count) {
        sum += v1[i]*v2[i];
    }
    return reduce_add(sum);
}
```

- ISPC : SIMD 명령어 활용에 특화된 언어.
- C언어 함수API를 생성:

```
extern float V1dotV2(const float *v1, const float *v2, int count);
```

- uniform? reduce_add?

벡터 내적 테스팅

```
auto a1 = randomVector(n);
auto a2 = randomVector(n);
auto b  = randomVector(n);
//Linear Property
auto a = a1 + a2;
assert(dot(a, b) ~ dot(a1, b)+dot(a2, b));

//Shuffling invariance
const auto s = Shuffle{ran};
assert(dot(a, b) ~ dot(s(a), s(b)));

//Inner product of unit vectors
assert(dot(e1, e1) ~ 1.0);
```

예제3 : 원소 합 구하기

```
val file = spark.textFile(“hdfs://citadel/adun/data.txt”)
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

- Spark MLlib: Scala언어 기반 머신러닝 라이브러리
- 분산 입출력/계산
- flatMap? reduceByKey?

원소 합 테스팅

```
auto a = randomList();
//Shuffling invariance
assert(sum(Shuffle{ran}(a)) ~ sum(a));

//Linear property
auto b = a;
b.randomElm() += delta;
assert(sum(a) + delta ~ sum(b));

//Normalization
a.setAllElement(1);
assert(sum(a) ~ len(a));
```

장점

- 버그 발생시 원인 파악에 큰 도움 : 생산성 향상
- 테스트 재사용성 : 내 요구사항(=테스트)을
  어느 라이브러리로도 시험 가능.
- 코드나 의존하는 라이브러리가 업데이트 되도 OK!
- 기본적인 명제를 테스트로 확인하고나면, 복잡한 명제도 간단한 명제들 기반으로 테스트 없이 증명 가능
- 테스트 중복제거, 테스트 완료 표시 : 타입 시스템 활용

정리

- 머신러닝 프로그램이 올바른지 확인하는 방법 연구중
- 다른 과학계산 분야에도 적용 가능
- 테스트를 합시다!
  - 언제까지 하나요? 필요충분조건 찾을때까지!
  - 찾은줄은 어떻게 아나요? 그것이 문제입니다.