Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Uncertain<T>: Abstraction Technique for Programming with Uncertain Data

Team:
Hammam Abdelwahab
Krishnateja Nallanukala
Manoj Kumar Murugan
Vishnu Vardhan Dadi

Customer & Supervisor:
Deebul Nair

Introduction:

Applications that measure and use the complexity of the world make use of estimates. For instance, mobile phones use information from GPS sensors to *estimate* the position. Machine learning models estimate hidden parameters form data. Uncertainty is defined as the difference between the true value and the estimate. Every estimate contains uncertainty caused due to systematic and random errors. For instance, random variables estimate uncertainty with probability distributions, which gives a probability to each possible value. When a biased coin is flipped there is a 90% chance for getting heads and 10% chance of getting tails. In this case, considering the outcome from one flip does not provide the estimate of a true value.

When uncertainty is not taken into account it creates three types of uncertainty bugs which are required to be avoided. Bug 1: using estimates as facts, when random noise in data is ignored it will introduce error, Bug 2: Computation compounds errors, computations performed on uncertain data reduce the accuracy significantly, Bug 3 : “conditionals asks boolean questions of probabilistic data”, which results in false negatives and false positives. *Need for Uncertainty:* Existing probabilistic programing languages require expertise beyond the requirement of clients application. Further, these existing languages do not take into account the estimates of the data, this leads to problems unaided.

For arbitrary probability distributions uncertainty is a programming language abstraction. During the runtime uncertainty creates a Bayesian network that describes computations using various distributions. Further, during the runtime, the hypothesis tests take into account only as many samples as required, for evaluating the particular conditional. These hypothesis tests guarantee bounds of accuracy and also result in high performance.
This uncertainty library is expected to address the following:

1. Characterization of the uncertainty bugs.
2. It provides semantics and abstraction, for uncertain data.
3. These implementations make the semantics more practical.
4. Uncertainty improves, correctness and expressiveness.

Motivation:

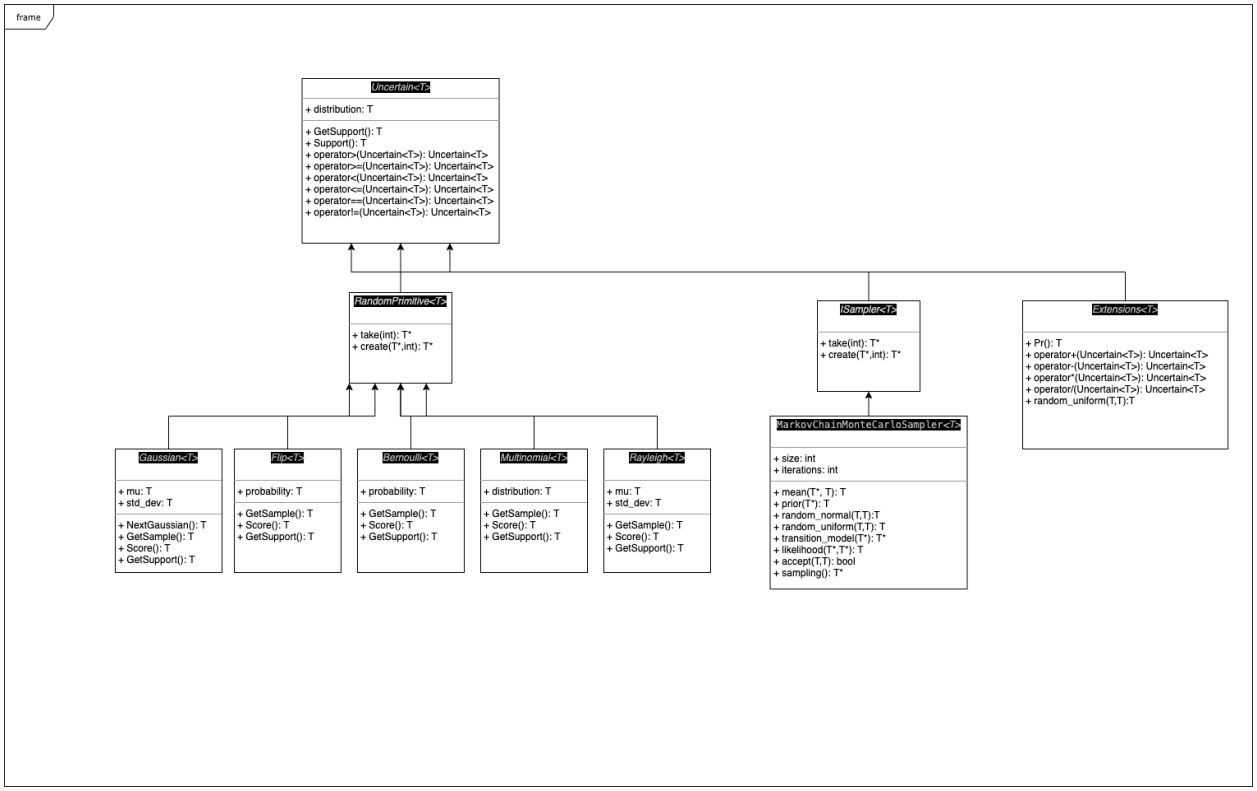
- Current day applications make computations using uncertain data from various experiments that generate such types of data such as vision, search, mobile sensors, chemical simulations, medical trials, benchmarking, and human surveys.
- In order to characterize uncertainty in the above mentioned sources of data, domain expertise is required. However most of the developers who use these results are non-experts.
- These non-expert developers treat these uncertain data as ground truth, and this is where the problem occurs.
- Uncertain library helps these non expert developers to handle this uncertain data better.

Problem statement :

We need to develop a library that enables non-experts in statistics to easily work with distributions. The implementation on this library is based on the code developed by [1] in C#. Our contribution is to enable Python and C++ developers to use this library easily.

Uncertain<T> structure :

The figure below demonstrates the UML of the developed Uncertain<T> datatype.



We developed several distributions such as Gaussian, Bernoulli, Rayleigh and more. Each of which can be used separately to generate random samples using RandomPrimitive class. As mentioned before, the Uncertain<T> data type enables the implementation of different arithmetic and logical operations. To do this, the Monte Carlo sampling class is used to estimate the parameters of the recently calculated distributions. The sampler class then takes samples from the new distribution generated by the Monte Carlo sampling class. For a clearer view, check the diagram in the link here.

Design decisions:

- Software development approach: Agile
- Programming paradigm : Object Oriented Programming
- Python coding convention : PEP 8
- C++ coding convention : General Best Practices

System setup:

- Ubuntu 14.0.4 and above
- Windows 7 and above
- Processor intel i3 and above (recommended)
- Python 2.7 and above
- C++ 11 and above

Dependencies:

The tools and libraries used while porting the Uncertain <T> library to python were:

- Python version: .2.7,3.x
- numpy version: 1.18.x
- scipy version: 1.4.x
- pytest: 6.2.2

The tools and libraries used while porting the Uncertain <T> library to C++ were:

- Catch2 for testing.

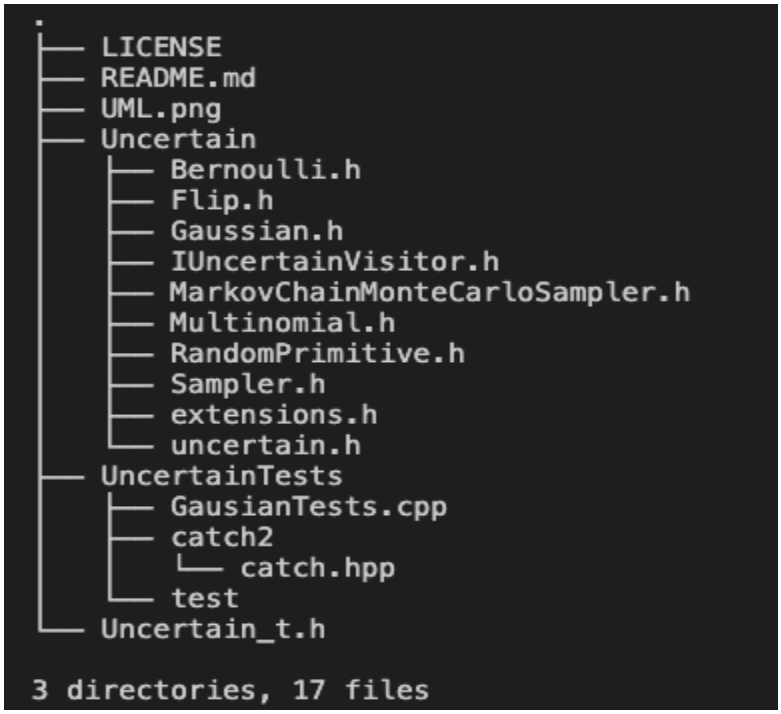
Package folder structure:

The Uncertain python package folders and files structure is as follows:

```
├── __init__.py
├── LICENSE
├── Readme.md
├── UML_diagram.png
├── Uncertain
│   ├── Extensions.py
│   ├── Flip.py
│   ├── Gaussian.py
│   ├── __init__.py
│   ├── MarkovChainMonteCarloSampler.py
│   ├── Sampler.py
│   └── Uncertain.py
└── UncertainTests
    ├── __init__.py
    ├── Readme.md
    └── test_GaussainTests.py

2 directories, 14 files
```

The Uncertain package folders and files structure for C++ is as follows:



Class and function definitions:

Filename	**MarkovChainMonteCarloSampler.py** , MarkovChainMonteCarloSampler.h			
Class name	**MarkovChainMonteCarloSampler**			
Function Name	**Signature**	**Description**	**Parameters**	**Result**
prior	prior(x)	Check log does not return negative values	x->list	boolean
transition_model	transition_mode(x)	Defines how the current parameter value moves to new parameter value.	x-> list	new parameter->list
likelihood	likelihood(x, data)	Computes the likelihood of the data given the parameters	x-> list	likelihood_value-> float
			data->list	
accept	accept(oldTrace, trace)	Accepts/rejects the sample	oldTrace->list	boolean
			trace->list	
mcmc_sampler	mcmc_sampler()	Implements Metropolis Hastings algorithm	-	New_distribution ->list

Filename	Extensions.py, extensions.h			
Class name	Extensions			
Function Name	Signature	Description	Parameters	Result
pr	pr(source)	Implements Wald'd likelihood ratio test	source->list	boolean

Filename	Flip.py, Flip.h			
Class name	Flip			
Function Name	Signature	Description	Parameters	Result
get_samples	get_samples()	Function to check the probability of flip	-	samples->list
get_support	get_support()	Returns flip samples	-	samples->list

Filename	Gaussian.py, Gaussian.h			
Class name	Gaussian			
Function Name	Signature	Description	Parameters	Result
get_samples	get_samples()	Creates random Gaussian samples with the mean and standard deviation.	-	samples->list
get_support	get_support()	Returns Gaussian samples	-	samples->list

Filename	Sampler.py, Sampler.h			
Class name	Gaussian			
Function Name	Sampler	Description	Parameters	Result
create	create(source)	Creates samples using MCMC sampler	source->object	-
take	take(samples)	Randomly picks n samples from a distribution	samples->int	samples->list

Installation of the python library:

To install the package in python run:

```
$ pip install UncertainPythonSDP==1.1.3
```

Installation of the C++ library:

Simply download the Uncertain_t.h file and the Uncertain folder and place them locally on your project.

Usage of the Python library:

The below code snippet shows the usage of functions in the library.

```
import UncertainPythonSDP
from UncertainPythonSDP.Uncertain.Gaussian import Gaussian
from UncertainPythonSDP.Uncertain.Uncertaint import Operator

distribution_1 = Gaussian(1.0,2.0)
distribution_2 = Gaussian(2.0,4.0)

sum_distribution = Operator(distribution_1)+Operator(distribution_2)
```

Usage of the C++ library:

```
#include <iostream>
#include "Uncertain_t.h"



int main(){

   Uncertain<double>* A = new Gaussian(5,0);
   Uncertain<double>* B = new Gaussian(3,0);

   Uncertain<double>* C = *A + *B;

   std::cout << (*C >= 8.0) << endl;

   return 0;
}
```

Testing and continuous integration:

- pytest - for passing test cases.
- Travis -ci - using continuous integration to check build status on github.

```
(base) dadi_vardhan@ubuntu20:~/Downloads/SDP/GUI Repo/SDP_Assignments/Uncertain_T/Uncer
tain_python$ pytest -v -W ignore
================================= test session starts ==================================
platform linux -- Python 3.8.5, pytest-6.2.2, py-1.10.0, pluggy-0.13.1 -- /home/dadi_va
rdhan/anaconda3/bin/python
cachedir: .pytest_cache
rootdir: /home/dadi_vardhan/Downloads/SDP/GUI Repo/SDP_Assignments/Uncertain_T/Uncertai
n_python
collected 6 items

UncertainTests/test_GaussainTests.py::test_gaussian_sample PASSED                [ 16%]
UncertainTests/test_GaussainTests.py::test_gaussian_mean PASSED                  [ 33%]
UncertainTests/test_GaussainTests.py::test_gaussian_bnn_sample PASSED            [ 50%]
UncertainTests/test_GaussainTests.py::test_gaussian_bnn_mean PASSED              [ 66%]
UncertainTests/test_GaussainTests.py::test_gaussian_bernoulli_mean PASSED        [ 83%]
UncertainTests/test_GaussainTests.py::test_gaussian_bernoulli_conditional PASSED [100%]

================================== 6 passed in 12.36s ==================================
```

For C++, Catch2 is used for unit tests as shown below.

```
TERMINAL  PROBLEMS 7  OUTPUT  DEBUG CONSOLE                    2: zsh

Hammam UncertainTests % g++ -std=c++11 -o test GausianTests.cpp
Hammam UncertainTests % ./test
X > Y evaluates true, incorrectly
Y > X evaluates true, incorrectly
Y > X evaluates false, incorrectly
X > Y evaluates false, incorrectly
===============================================================================
All tests passed (6 assertions in 1 test case)

Hammam UncertainTests %
```

Future Work:

- Implementation of other distributions. [Python/C++]
- Proving the remaining case studies.

References:

[1]J. Bornholt, T. Mytkowicz, and K. S. McKinley, “Uncertain : A First-Order Type for Uncertain Data,” in Proceedings of the 19thinternational conference on Architectural support for programming languages and operating systems, New York, NY, USA, Feb. 2014,pp. 51–66, doi: 10.1145/2541940.2541958.