Factor Graphs Models for Explainable and Causal Machine Learning using Infer.NET and Python

Celso Axelrud

12/13/20

Introduction

The Machine Learning Frameworks are showing different paths in software development due to the application requirements.
They are:

- Non-explainable, non-causal, high fitting models using Deep Learning Neural-Networks.
- Explainable, causal models using Probabilistic Learning (PPL: Probabilistic Programing Language, Bayesian Networks, Probabilistic Graphical Models).

Probabilistic Programming allows the development of models that include parameters' uncertainty and, consequently, predictions' uncertainty.

Causal model requires much more than Probabilistic Learning.
But it is straightforward to represent Causal models by Direct Acyclic Graphs (DAGs) and First-Principles models using Probabilistic Learning technologies.

For example, *Tensorflow* (Google) is an automatic differentiation, linear algebra, hardware accelerated (GPU) language. *Tensorflow* has support for Deep Learning using *Keras* and for Probabilistic Learning using *Tensorflow Probability* and *NumPyro* (*JAX*) (both are PPL).

Another example is *Torch* (Facebook). It is a language equivalent to *Tensorflow*. *Torch* has support for Deep Learning and for Probabilistic Learning using *Pyro* (PPL from Uber).

PPL, in combination with accelerated hardware, are revolutionary because allows the practical use of Bayesian Statistics, using numeric methods like Markov Chain Monte Carlo (MCMC) and Variational Inference (VI).

But PPL is not a Graphical Models (Graphs or Networks) and there are important functionalities that are missing (by design).
They are:

- Automatic inference in any direction in the network for the variables without evidence.
- Straightforward On-line Learning (Bayesian) procedure.

These functionalities are present in the Bayesian Networks.

But Bayesian Networks have limitations like:

- Linear models.
- Only Categorical and Gaussian distribution.
- Difficult to create models with large number (thousands) of variables.

An interesting Probabilistic Graphical Modeling technology is Factor Graph
(https://en.wikipedia.org/wiki/Factor_graph).

Factor Graph extends the Bayesian Network with:

- Nonlinear models.
- Other distributions as Gaussian, Gamma, Bernoulli, Beta, Dirichlet, Discrete/Categorical/Multimodal, Poisson, Binomial and Wishart.
- Online Learning of continuous variable.
- Models with large number of variable (thousands) using variable arrays.
- Models with logic.
- Hierarchical Models (share knowledge between similar populations).

There are few Factor Graph software available.
The most important is Infer.Net (Microsoft). It is open-source and free for commercial use.
Infer.NET is used internally at Microsoft as the machine learning engine in some of their products such as Office, Azure, and Xbox (**TrueSkill** 2 – a system that matches players in online video games).

In 2019, I wrote a document demonstrating the use of Infer.Net using F# (Microsoft).

In this document, I want to present the same demonstration using Python.
The use of Infer.Net in Python unlock many possibilities due to the popularity of Python.
It includes online applications connected to IIOT system (as ptc/Thingworx
https://www.ptc.com/en/technologies/iiot).

Embedding Infer.Net in Python is possible using Pythonnet (http://pythonnet.github.io/).
It is the same technology already used to embed *Bayes Server* in Python (https://www.bayesserver.com/
).

Example – Cycling Time

This document is inspired on *"Infer.NET 101 A sample-based introduction to the basics of Microsoft Infer.NET programming"* (https://dotnet.github.io/infer/InferNet101.pdf).

This example is based on the following scenario:

- You bicycle to work every day.
- An individual cyclist's travel time varies randomly from day to day, so its value is uncertain.
- The travel time's uncertainty is represented by a probability distribution that defines the average travel time and how much it varies.
- The application will learn this distribution from several observed travel times and use that knowledge to make predictions about future travel times.

This simple example demonstrates the main functionalities of Infer.Net in Python including feature retraction and online training.

```
#CyclingTime.py

# Refs:
#https://github.com/Tobimaru/InferNet-pythonnet/blob/master/InferDotNet_simple_examples.ipynb
# Comments:
# use help() to inspect .NET objects
# DLLs:
# MicrosoftML.Probabilistic (netstandar2.0)
# Microsoft.ML.Probabilistic.Compiler.dll (netstandar2.0)
# System.CodeDom.dll (net461)
# MathNet.Numerics

import os
os.chdir(r"C:\Users\inter\OneDrive\_myWork\Research2020\Infernet_2020")

import sys
sys.path.append(r'C:\Users\inter\OneDrive\_myWork\Research2020\Infernet_2020')

import clr

clr.AddReference("Microsoft.ML.Probabilistic")
clr.AddReference("Microsoft.ML.Probabilistic.Compiler")

clr.AddReference("MathNet.Numerics")

import System
from System import Array, Double, Type , Converter
import Microsoft.ML.Probabilistic
import Microsoft.ML.Probabilistic.Models
import Microsoft.ML.Probabilistic.Distributions
import Microsoft.ML.Probabilistic.Math
import Microsoft.ML.Probabilistic.Algorithms

from Microsoft.ML.Probabilistic.Models import Variable, VariableArray, Range, InferenceEngine
from Microsoft.ML.Probabilistic.Algorithms import VariationalMessagePassing
from Microsoft.ML.Probabilistic.Distributions import Gaussian, Gamma
from Microsoft.ML.Probabilistic.Distributions import Distribution, VectorGaussian, Gaussian, G
amma, Bernoulli, Discrete
from Microsoft.ML.Probabilistic.Math import Rand, Vector, PositiveDefiniteMatrix

from utils import plot_graph

import MathNet.Numerics
import math
import matplotlib.pyplot as plt

# Example CyclingTime - 1
#Refs:
#    Infer_Cycling_0_2.fsx
#    C:\Users\inter\OneDrive\Projects(Comp)\Dev_2018\Infer.NET_2018\infer-
master\test\TestFSharp
#    https://github.com/dotnet/infer
#    https://github.com/dotnet/infer/tree/master/test/TestFSharp
#    https://dotnet.github.io/infer/InferNet101.pdf

#[1] The model
```

```
averageTime =  Variable.GaussianFromMeanAndPrecision(15.0, 0.01).Named("AverageTime") #precisi
on:reciprocal of the variance (1/sigma^2)
trafficNoise = Variable.GammaFromShapeAndScale(2.0, 0.5).Named("trafficNoise")
travelTimeMonday = Variable.GaussianFromMeanAndPrecision(averageTime, trafficNoise).Named("Tra
velTimeMon")
travelTimeTuesday = Variable.GaussianFromMeanAndPrecision(averageTime, trafficNoise).Named("Tr
avelTimeTue")
travelTimeWednesday = Variable.GaussianFromMeanAndPrecision(averageTime, trafficNoise).Named("
TravelTimeWed")

#[2] Train the model
travelTimeMonday.ObservedValue = 13.0
travelTimeTuesday.ObservedValue = 17.0
travelTimeWednesday.ObservedValue = 16.0

engine = InferenceEngine()
engine.ShowFactorGraph=False

averageTimePosterior = engine.Infer[Gaussian](averageTime)
trafficNoisePosterior = engine.Infer[Gamma](trafficNoise)

print("averageTimePosterior (mean,variance): {}".format(averageTimePosterior))
#averageTimePosterior (mean,variance): Gaussian(15.33, 1.32)

print("trafficNoisePosterior: {}".format(trafficNoisePosterior))
#trafficNoisePosterior: Gamma(2.242, 0.2445)[mean=0.5482]

#dist2 = MathNet.Numerics.Distributions.Gamma(trafficNoisePosterior.Shape,trafficNoisePosterio
r.Rate)
#samples2 = [dist2.Sample() for x in range(10000)]

#plt.hist(samples2, bins='auto')
#plt.title("Histogram with 'auto' bins")
#plt.show()

# [3] Add a prediction variable and retrain the model
tomorrowsTime = Variable.GaussianFromMeanAndPrecision(averageTime,trafficNoise).Named("Tomorro
wsTime")
#engine.ShowFactorGraph=False
tomorrowsTimeDist = engine.Infer[Gaussian](tomorrowsTime)
tomorrowsMean = tomorrowsTimeDist.GetMean()
tomorrowsStdDev = math.sqrt(tomorrowsTimeDist.GetVariance())
print("Tomorrows predicted time: {} plus or minus {}".format(tomorrowsMean,tomorrowsStdDev))
#Tomorrows predicted time: 15.326309046104376 plus or minus 2.147767008944041

# You can also ask other questions of the model
#threshold = Variable[float].Constant(18.0) .Named("threshold")

threshold = Variable.New[Double]().Named("threshold")
threshold.ObservedValue = 18.0

probTripTakesLessThan18Minutes = engine.Infer[Bernoulli](tomorrowsTime.op_GreaterThan(threshol
d,tomorrowsTime)).GetProbTrue()
print("Probability that the trip takes less than 18 min: {}".format(probTripTakesLessThan18Min
utes))
#Probability that the trip takes less than 18 min: 0.8934102503238182

# INPUT RETRACTION
#----------------
#Add observed value for tomorrowsTime (CA)
tomorrowsTime.ObservedValue = tomorrowsMean
```

```
# Clear TimeWednesday
travelTimeWednesday.ClearObservedValue()
travelTimeWednesdayDist = engine.Infer[Gaussian](travelTimeWednesday)
print("travelTimeWednesdayDist: {}".format(travelTimeWednesdayDist))
#travelTimeWednesdayDist: Gaussian(15.11, 4.301)

#---------------------------------------------------------------------------
# Infer.NET: script for CyclingTime 2
# List, Dictionary

#Priors
averageTimePrior =  Variable.GaussianFromMeanAndPrecision(15.0, 0.01).Named("AverageTimePrior"
)
trafficNoisePrior = Variable.GammaFromShapeAndScale(2.0, 0.5).Named("trafficNoisePrior")
# CreateModel-----
trainingData_Length1=10
numTrips1 = Range(trainingData_Length1).Named('numTrips1')
travelTimes1 = Variable.Array[Double](numTrips1).Named('travelTimes1')
travelTimes1.set_Item(numTrips1, Variable.GaussianFromMeanAndPrecision(averageTimePrior, traff
icNoisePrior).ForEach(numTrips1))
trainingData1 = [ 13.0, 17.0, 16.0, 12.0, 13.0, 12.0, 14.0, 18.0, 16.0, 16.0 ]
travelTimes1.set_ObservedValue(Array[Double](trainingData1))
# Infer-----
#engine = InferenceEngine()
#engine.ShowFactorGraph=False

averageTimePost_Dist_1 = engine.Infer[Gaussian](averageTimePrior)
trafficNoisePost_Dist_1 = engine.Infer[Gamma](trafficNoisePrior)

print("averageTimePost_Dist_1 (mean,variance): {}".format(averageTimePost_Dist_1))
#averageTimePost_Dist_1 (mean,variance): Gaussian(14.7, 0.4457)

print("trafficNoisePosterior1 (mean,variance): {}".format(trafficNoisePost_Dist_1))
#trafficNoisePost_Dist_1 (mean,variance): Gamma(5.33, 0.05402)[mean=0.2879]

Cyclist_L1=[trainingData_Length1,trainingData1,travelTimes1,averageTimePrior,trafficNoisePrior
,averageTimePost_Dist_1,trafficNoisePost_Dist_1]

print("averageTimePost_Dist_1 (mean,variance): {}".format(Cyclist_L1[5]))
#averageTimePost_Dist_1 (mean,variance): Gaussian(14.7, 0.4457)

Cyclist_D1 = {
  "trainingData_Length":trainingData_Length1,
  "trainingData": trainingData1,
  "travelTimes": travelTimes1,
  "averageTimePrior": averageTimePrior,
  "trafficNoisePrior":trafficNoisePrior,
  "averageTimePost_Dist":averageTimePost_Dist_1,
  "trafficNoisePost_Dist":trafficNoisePost_Dist_1
}
print("averageTimePost_Dist_1 (mean,variance): {}".format(Cyclist_D1["averageTimePost_Dist"]))
#averageTimePost_Dist_1 (mean,variance): Gaussian(14.7, 0.4457)

tomorrowsTimePrior = Variable.GaussianFromMeanAndPrecision(averageTimePrior,trafficNoisePrior)
.Named("TomorrowsTimePrior")
tomorrowsTimePost_Dist_1 = engine.Infer[Gaussian](tomorrowsTimePrior)
tomorrowsMean1 = tomorrowsTimePost_Dist_1.GetMean()
tomorrowsStdDev1 = math.sqrt(tomorrowsTimePost_Dist_1.GetVariance())
print("Tomorrows predicted time 1: {} plus or minus {}".format(tomorrowsMean1,tomorrowsStdDev1
))
#Tomorrows predicted time 1: 14.701115540046697 plus or minus 2.1727781502564896
```

```
Cyclist_D1["tomorrowsTimePosterior"]=tomorrowsTimePost_Dist_1

#-------------
# NEW TRAINING
# Re-train based on a new data (ignore old data)

trainingData1_2 = [17.0, 19.0, 18.0, 21.0, 19.0,17.0, 19.0, 18.0, 21.0, 18.0,17.0, 19.0, 18.0,
 21.0, 23.0]
trainingData_Length1_2=len(trainingData1_2)
numTrips1_2 = Range(trainingData_Length1_2).Named('numTrips1_2')
travelTimes1_2 = Variable.Array[Double](numTrips1_2).Named('travelTimes1_2')

travelTimes1_2.set_Item(numTrips1_2, Variable.GaussianFromMeanAndPrecision(averageTimePrior,tr
afficNoisePrior).ForEach(numTrips1_2))
travelTimes1_2.set_ObservedValue(Array[Double](trainingData1_2))

averageTimePost_Dist_1_2 = engine.Infer[Gaussian](averageTimePrior)
trafficNoisePost_Dist_1_2 = engine.Infer[Gamma](trafficNoisePrior)

#---------
# BAYESIAN TRAINING (ONLINE LEARNING)
# Re-train based on a new data (consider old data)

m1=averageTimePost_Dist_1.GetMean()
v1=averageTimePost_Dist_1.GetVariance()
averageTimePrior_2 =  Variable.GaussianFromMeanAndVariance(m1, v1).Named("AverageTimePrior_2")


s1=trafficNoisePost_Dist_1.Shape
r1=trafficNoisePost_Dist_1.Rate
trafficNoisePrior_2 = Variable.GammaFromShapeAndRate(s1,r1).Named("trafficNoisePrior_2")

trainingData1_2 = trainingData1_2 = [17.0, 19.0, 18.0, 21.0, 19.0,17.0, 19.0, 18.0, 21.0, 18.0
,17.0, 19.0, 18.0, 21.0, 23.0]
trainingData_Length1_2=len(trainingData1_2)
numTrips1_2 = Range(trainingData_Length1_2).Named('numTrips1_2')
travelTimes1_2a = Variable.Array[Double](numTrips1_2).Named('travelTimes1_2') # Check: re-
use the same variable
travelTimes1_2a.set_Item(numTrips1_2, Variable.GaussianFromMeanAndPrecision(averageTimePrior_2
,trafficNoisePrior_2).ForEach(numTrips1_2))
travelTimes1_2a.set_ObservedValue(Array[Double](trainingData1_2))

averageTimePost_Dist_1_2a = engine.Infer[Gaussian](averageTimePrior_2)
trafficNoisePost_Dist_1_2a = engine.Infer[Gamma](trafficNoisePrior_2)

print("averageTimePost_Dist_1 (mean,variance): {}".format(averageTimePost_Dist_1))
#averageTimePost_Dist_1 (mean,variance): Gaussian(14.7, 0.4457)
print("averageTimePost_Dist_1_2 (mean,variance): {}".format(averageTimePost_Dist_1_2))
#averageTimePost_Dist_1_2 (mean,variance): Gaussian(17.27, 0.3118)
print("averageTimePost_Dist_1_2a (mean,variance): {}".format(averageTimePost_Dist_1_2a))
#averageTimePost_Dist_1_2a (mean,variance): Gaussian(17.07, 0.2119)

print("trafficNoisePost_Dist_1 (mean,variance): {}".format(trafficNoisePost_Dist_1))
#trafficNoisePost_Dist_1 (mean,variance): Gamma(5.33, 0.05402)[mean=0.2879]
print("trafficNoisePost_Dist_1_2 (mean,variance): {}".format(trafficNoisePost_Dist_1_2))
#trafficNoisePost_Dist_1_2 (mean,variance): Gamma(13.22, 0.01057)[mean=0.1397]
print("trafficNoisePost_Dist_1_2a (mean,variance): {}".format(trafficNoisePost_Dist_1_2a))
#trafficNoisePost_Dist_1_2a (mean,variance): Gamma(12.06, 0.01524)[mean=0.1839]
```