PGM_PyLib: A Toolkit for Probabilistic Graphical Models in Python

Jonathan Serrano-Pérez JS.PEREZ@INAOEP.MX

L. Enrique Sucar ESUCAR@INAOEP.MX

Instituto Nacional de Astrofísica, Óptica y Electrónica, Puebla, México

Abstract

PGM_PyLib is a toolkit that contains a wide range of Probabilistic Graphical Models algorithms implemented in Python, and serves as a companion of the book Probabilistic Graphical Models: Principles and Applications. Currently, the algorithms implemented include: Bayesian classifiers, hidden Markov models, Markov random fields, and Bayesian networks; as well as some general functions. The toolkit is open source, can be downloaded from:
https://github.com/jona2510/PGM_PyLib .

Keywords: PGM library; Python; Bayesian classifiers, HMMs, MRFs, BNs.

1. Introduction

Probabilistic Graphical Models (PGMs) include several computational techniques based on a graphical representation of independence relations, such as Bayesian classifiers, hidden Markov models, Markov networks, Bayesian networks, influence diagrams, etc. PGMs have a wide range of applications (Sucar, 2015), such as medical diagnosis and decision making; mobile robot localization, navigation and planning; modeling the evolution of viruses; student modeling, among many others.

Although there are many implementations of PGMs, most of them are focused on particular types of models, such as: Bayesian classifiers – *weka*[1], *meka*[2], *scikit-learn*[3]; hidden Markov models – *UMDHMM*[4], *MAMOT*[5]; or Bayesian networks – *Hugin*[6], *Elvira*[7], *OpenMarkov*[8].

The objective of developing this new toolkit is twofold. Firstly, it is to gather different types of PGMs algorithms in one place and in the same language. Secondly, the toolkit can be used as complement of the book *Probabilistic Graphical Models: Principles and Applications* (Sucar, 2015), where the theory and algorithms can be found. **PGM_PyLib** is implemented in Python due to the great popularity of this language for researchers in machine learning and artificial intelligence. Currently, the toolkit contains algorithms for Bayesian classifiers, hidden Markov models, Markov random fields, and Bayesian networks; and in the near future it will also include dynamic Bayesian networks, influence diagrams and Markov decision processes.

The algorithms included, so far, in the toolkit are briefly described in the next sections: Bayesian classifiers 2, hidden Markov models 3, Markov random fields 4, Bayesian networks 5 and others 6. Furthermore, the manual (Serrano-Pérez and Sucar, 2020) together with the toolkit can found

1. https://www.cs.waikato.ac.nz/ml/weka/
2. http://waikato.github.io/meka/
3. https://scikit-learn.org/stable/
4. http://www.kanungo.com/software/software.html
5. https://bcf.isb-sib.ch/mamot/
6. https://www.hugin.com/
7. http://leo.ugr.es/elvira/
8. http://www.openmarkov.org

at: https://github.com/jona2510/PGM_PyLib, where there are examples of how to use each algorithm.

2. Bayesian Classifiers

Classification consists on assigning classes or labels to objects (Sucar, 2015). However, there are different variants of classification, that is, an instance could be associated to a single class, or it could be associated to multiple classes. Bayesian classifiers are grouped in 3 sections.

2.1 Multiclass Classification

In multiclass classification an instance is associated to only one class. The classifiers that includes the toolkit are the following:

- *Naive Bayes Classifies (NBC)*: it is based in the assumption that all the attributes are independent given the class variable. NBC can only handle nominal attributes.
- *Gaussian Naive Bayes Classifier (GNBC)*: it follows the same idea than NBC, but GNBC is able to handle numeric attributes.
- *BANs*: TAN and BAN incorporate dependencies between attributes, represented as a tree (TAN) or a Directed Acyclic Graph (BAN). Our implementation is more related with BANs, because it can handle any structure.
- *Semi-Naive Bayes Classifier (SNBC)*: it eliminates or joins attributes which are not independent given the class, in order to improve the performance of the classifier.

2.1.1 EXAMPLE OF A BAN CLASSIFIER IN PYTHON

In Listing 1 an example of how to use the BAN classifier is presented. First of all, it is required to import the package which contains the classifier (line 2). In this case we are considering a problem with 3 classes and 5 attributes. Once the data is ready, the next step is to instantiate the classifier (line 11) with its parameters, in this case, the structure is generated automatically, the smooth used for the estimations of probabilities is 0.1 and the prior probabilities will be used in the prediction phase. Then, the classifier is trained with training data (line 12). Once the classifier is trained, it is used to predict the class of new instances (line 13). Finally, an evaluation measure such as *exact-match/accuracy* can be used to evaluated the performance of the classifier (line 14).

2.2 Multidimensional classification

In Multidimensional classification (MDC) an instance is associated to D classes. A particular case of MDC is multilabel classification where all classes are binary. Currently, the toolkit includes *Bayesian Chain Classifiers (BCCs)*. BCCs take advantage of the relations between classes represented as a Directed Acyclic Graph (DAG), so the predictions of a class are influenced by the predictions of its *neighbours*. Three variants of neighbours are implemented: **Parents**: which corresponds to the original version of BCC. **Ancestors**: which influences the predictions of the class with its ancestors. **Children**: which influences the predictions of the class with its children.

```
1  import numpy as np
2  import PGM_PyLib.augmented as abc
3  # 5 attributes, 3 classes
4  # simulation of 100 instances for training
5  data_train = np.random.randint(0,5,size=(100,5))
6  cl_train = np.random.randint(0,3,size=100)
7  # simulation of 50 instances for testing
8  data_test = np.random.randint(0,5,size=(50,5))
9  cl_test = np.random.randint(0,3,size=50)
10 # create the classifiers
11 c = abc.augmentedBC(algStructure="auto", smooth=0.1, usePrior=True)
12 c.fit(data_train, cl_train)     # train the classifier
13 p = c.predict(data_test)        # predict
14 print(c.exactMatch(cl_test, p)) # evaluation
```

Listing 1: Example of BAN

2.3 Hierarchical classification

Hierarchical classification (HC) can be seen as a special case of multilabel classification, where the labels are arranged in a predefined structure (DAG), and the predictions have to comply with the hierarchical constraint. The toolkit includes *HC with Bayesian Networks and Chained Classifiers (BNCC)*: it combines two strategies in order to predict the labels to which an instance is associated while it complies the hierarchical constraint. Four variants are available, which differ by the type of chained classifier: **HCP** - parents, **HCA** - ancestors, **HCC** - children, and **HBA** - independent.

3. Hidden Markov Models

Hidden Markov Models (HMMs) can be seen as a double stochastic process, that is, a hidden stochastic process that we can not directly observe, and a second stochastic process that produces the sequence of observations given the first process. The toolkit includes algorithms to solve the following problems: evaluation - *forward*, state estimation - *Viterbi*, and learning - *Baum-Welch*.

3.1 Example of HMM

An example of how to use HMMs is shown in listing 2. First, the package that contains the HMM is imported (line 1). Then, the model is instantiated with the required parameters (line 8). The observation sequence is shown in line 9, and it is evaluated with the forward algorithm (line 12). Finally, the most probable sequence of states is obtained with the Viterbi algorithm (line 14).

4. Markov Random Fields

Markov random fields (MRF) are undirected graphical models where each variable can take different values and is influenced probabilistically by the values of its neighbors. When the variables are arranged as a regular grid, it is known as regular MRF (RMRF). For inference in a RMRF, a general stochastic search procedure for finding the configuration of minimum energy was implemented. It supports three variants: *Iterative Conditional Modes, Metropolis* and *Simulated Annealing*. Furthermore, two alternatives with respect to the *optimal* configuration are available: *Maximum A posteriori Probability (MAP)* and *Maximum Posterior Marginals (MPM)*.

```python
import PGM_PyLib.HMM as hmm, numpy as np
states = ["M1", "M2"]
obs = ["H", "T"]
PI = np.array( [0.5, 0.5] ) #prior probabilities
A = np.array( [[0.5, 0.5], [0.5, 0.5]] ) #transition probabilities
B = np.array( [[0.8, 0.2], [0.2, 0.8]] ) #observation probabilities
# Inializating the model with all its parameters
h = hmm.HMM(states=states,observations=obs,prior=PI, transition=A,observation=B)
O = ["H","H","T","T"] # observation sequence
# evaluating an observation sequence
print("Score of: H,H,T,T")
print(h.forward(O))
# obtaining the most probable sequence of states
mpss,score = h.viterbi(O)
print("Most probable sequence of states: "+ str(mpss))
```

Listing 2: Example of HMM

5. Bayesian Networks

Currently the toolkit includes algorithms for learning Bayesian networks, later we will incorporate inference algorithms. A particular case for learning Bayesian networks is learning trees, that is, each variable has only one parent, except the root that does not have parents. The algorithms included in the toolkit are the following:

- *Chow-Liu Procedure (CLP)*: it estimates the Mutual Information (MI) between each pair of variables, and uses the pairs of variables with the highest MI for building the *skeleton* of a tree. Directions of the arcs are given by selecting one variable as the root of the tree and assigning directions to the arcs starting from the root.
- *CLP with Conditional Mutual Information (CLP-CMI)*: it follows the same principles than CLP, however, CMI is estimated for each pair of variables given a third, this last is an additional variable that is used for all the CMI estimations.

6. Others

Modules that are useful for previous implementations are also available: mutual information $MI(X;Y)$, conditional mutual information $CMI(X;Y|Z)$, and estimation of probabilities $P(A|C_1, C_2, ..., C_n)$.

Acknowledgments

This work was partially supported by CONACYT under project No. CB2017-2018 43346.

References

J. Serrano-Pérez and L. E. Sucar. *PGM PyLib: A Python Library for Inference and Learning of Probabilistic Graphical Models*, 2020. URL https://github.com/jona2510/PGM_PyLib.

L. E. Sucar. *Probabilistic Graphical Models Principles and Applications*. Springer, London, 2015.