Lab 2: Multi-label Learning and Classification

INF581: Advanced Topics in Artificial Intelligence

In this lab we deal with multi-label classification, both with probabilistic methods and neural-network architectures. We will use the multi-label Music-Emotions dataset where attributes describing a piece of music are associated to a subset of six emotions: {`amazed-surprised`, `happy-pleased`, `relaxing-clam`, `quiet-still`, `sad-lonely`, and `angry-aggressive`}. The data is available in the file `music.csv`, and is already loaded and prepared for you in the code made available.

Note: The lab contains two tasks, only the Task 2 is graded directly (the first task will still be useful for upcoming TDs). **Submission Instructions:** See `README.md`

Task 1: Deep Multi-label Learning

This task uses `PyTorch`. If not familiar with `PyTorch`, start by going through the official `PyTorch` tutorial `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`, which contains 4 notebooks, the last one trains a CNN to classify images on CIFAR10. In the task below, the network trained is less complex but contains a skip layer.

The task is to implement the network shown in Figure 1 (based on [1]), and test it on the Music-Emotions dataset. Obviously, additional architecture could be added for more advanced problems [3].

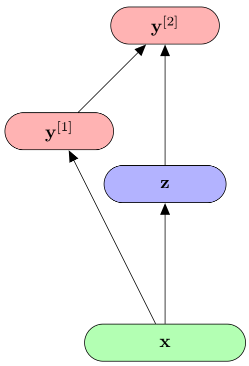


Figure 1: A neural network, predicting $\boldsymbol{y} = [y_1, y_2, y_3, y_4, y_5, y_6]$ from instance $\boldsymbol{x}$ ($\boldsymbol{z}$ is a hidden layer). Note that, e.g., $\boldsymbol{y}^{[1]} = [y_1, y_2, y_3]$ and $\boldsymbol{y}^{[2]} = [y_4, y_5, y_6]$; and we predict $y_j \in [0, 1]$ for each j.

In the Jupyter notebook `NN_Notebook.ipynb`, complete the following tasks at the respective places indicated by `TODO` in the provided code, in adherence to the documentation:

1. Design the network by modifying the class `multilabel_classifier`
2. Assign to `my_loss` the following loss function,

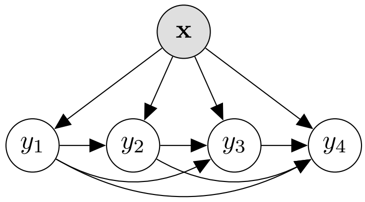
$$\ell(\boldsymbol{y}^{(i)}, \boldsymbol{\sigma}^{(i)}) = -\frac{1}{m} \sum_{j=1}^{m} \{y_j \log \sigma_j + (1 - y_j) \log(1 - \sigma_j)\}$$

   where $\sigma_j \approx P(Y_j = 1|\boldsymbol{x})$, and $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_m]$, input instance $\boldsymbol{x}$. **Hints:** 1) Use `PyTorch` losses as appropriate; 2) Be careful: if you define a non-linearity in `forward`, you don't need it in `my_loss`.

3. After training the network, compute the Hamming and 0/1 loss on the test set

Task 2: Inference as a Search

In this task we consider a *probabilistic classifier chain* model: essentially a Bayesian network, exemplified as follows (for a problem with $m = 4$ labels),



where conditional probabilities are modeled by some probabilistic binary *base classifier*, hence the j-th base classifier providing

$$\hat{y}_j = h_j(\boldsymbol{x}) := \underset{y_j \in \{0,1\}}{\arg\max}\, P(y_j|\boldsymbol{x}, \hat{y}_1, \ldots, \hat{y}_{j-1}) \tag{1}$$

for a test instance $\boldsymbol{x}$ for each label $j = 1, \ldots, m$; and thus the 'chain' consists of *base classifiers* $h_1, \ldots, h_m$. The goal is to be able to provide predictions $\hat{\boldsymbol{y}}$ and associated confidence/uncertainty

$$P(\hat{\boldsymbol{y}}|\boldsymbol{x}) = \prod_{j=1}^{m} P(\hat{y}_j|\boldsymbol{x}, \hat{y}_1, \ldots, \hat{y}_{j-1})$$

Since exact inference is usually intractable, we are going to **implement epsilon ϵ-approximate search** for inference. A survey of inference methodologies for probabilistic classifier chains, is provided in [2] that includes examples/figures that may be useful (also included in the lecture slides). See also Figure 2.

There is code provided (under `__name__ == "__main__"`) which loads the Music-Emotions dataset, instantiates and trains a classifier chain (i.e., fitting the base classifiers; which is logistic regression in this case), evaluates the model, and visualizes inference for one of the test instances (note: you need the `graphviz` library for the visualisation). However, the current implementation is *greedy inference* only; you should complete the code such that it performs ϵ-approximate inference.

Re-implement the function `epsilon_approximate_tree_inference` in `lab2.py` as *ϵ-approximate search.* You can use any modules from the Python 3 standard library `https://docs.python.org/3/library/` (this is optional; not required for a working solution). Your code must be documented/commented sufficiently to be understood easily.

Hints: Notice the similarity to TD 1, which also involved a tree search for inference. Note that, according to the grading scheme you can get partial marks for *any* working inference scheme that improves over greedy inference (80%, if it is competitive with ϵ-approximate without being much more computationally complex); if you take this option *do not* rename the function; simply ignore the `epsilon` parameter.

Bonus Task: Create a new `.py` file with a class `Adios` ('Architecture Deep in the Output Space'), and integrate/modify your work from Task 1 such that the model can be instantiated (e.g., `adios = Adios(H)` where `H` the number of hidden units), trained (e.g., `adios.fit(X,Y)`), and used for predictions (e.g., `Ypred = adios.predict(X,Y)`) as a Scikit-Learn classifier. Submit it to `https://nuage.lix.polytechnique.fr/index.php/s/P2oY88jMSDSpjcG` (don't forget to include a Python dictionary with `info['Email'] = your.email@polytechnique.edu`).

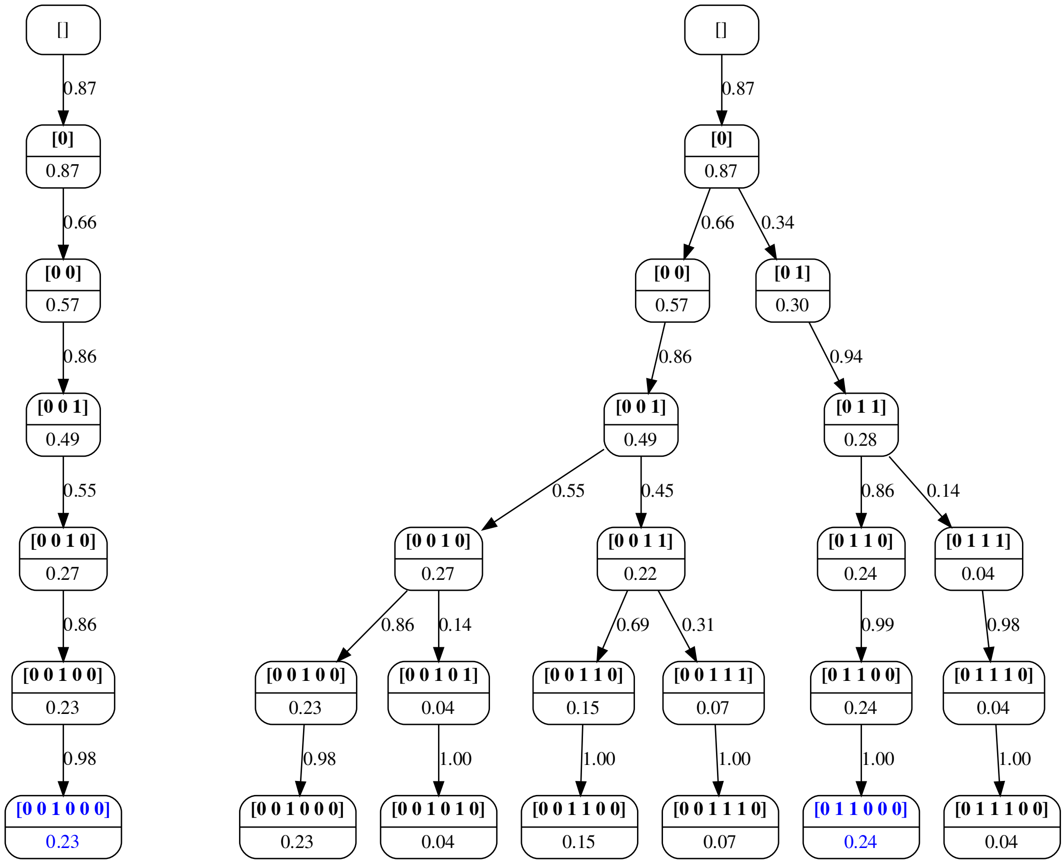


Figure 2: The part of the probability tree explored by *greedy search* (left) and *Monte Carlo search* (right) for a given instance $\boldsymbol{x}$; of the Music-Emotions data. The value of each path, $P(y_1, \ldots, y_j|\boldsymbol{x})$, is shown in each node; and the value of each edge, $P(y_j|\boldsymbol{x}, y_1, \ldots, y_{j-1})$ is shown on the edge label. Unexplored paths are not shown. Recall: You *do not have to implement* Monte Carlo sampling; it is shown here only as a demonstration.

References

[1] Moustapha Cisse, Maruan Al-Shedivat, and Samy Bengio. Adios: Architectures deep in output space. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 2770–2779, New York, New York, USA, 20–22 Jun 2016. PMLR.

[2] Deiner Mena, Elena Montañés, José Ramón Quevedo, and Juan José Coz. An overview of inference methods in probabilistic classifier chains for multilabel classification. *Wiley Int. Rev. Data Min. and Knowl. Disc.*, 6(6):215–230, November 2016. https://digibuo.uniovi.es/dspace/bitstream/handle/10651/39325/surveypcc-gracc.pdf.

[3] Willem Waegeman and Dimitrios Iliadis. Multi-target prediction with deep neural networks: A hands-on tutorial. In *ECML/PKDD 2022 Tutorials*, 2022. https://kermit.ugent.be/multi-target-prediction/.