Hot Topics in Machine Learning (HWS17)
Assignment 3: Graphical Models

November 3, 2017

The archive provided to you contains this assignment description, datasets, as well as Python code fragments for you to complete. Comments and documentation in the code provide further information.

It suffices to fill out the "holes" that are marked in the code fragments provided to you, but feel free to modify the code to your liking. You need to stick with Python though.

Please adhere to the following guidelines in all the assignments. If you do not follow those guidelines, we may grade your solution as a FAIL.

Provide a single ZIP archive with name `html17-<assignment number>-<your ILIAS login>.zip`. The archive needs to contain:

- A single PDF report that contains answers to the tasks specified in the assignment, including helpful figures and a high-level description of your approach. Do not simply convert your Jupyter notebook to a PDF! Write a separate document, stay focused and brief. As a guideline, try to stay below 10 pages.
- All the code that you created and used in its original format.
- A PDF document that renders your Jupyter notebook with all figures. (If you don't use Jupyter, then you obviously do not need to provide this.)

Make sure that your report is self-explanatory and follows standard scientific practice. Use the tasks numbers of the assignments as your section and subsection numbers. Label all figures (and refer to figure labels in your write-up). Include references if you used additional sources or material.

Hand-in your solution via ILIAS until the date specified there. This is a hard deadline.

Provided Material

The code provided to you contains a barebones API and implementation for working with factor graphs on categorical variables. It provides classes for categorical variables (`Var`), for categorical distributions (`Dist`, normalized or unnormalized), for factors (`Factor`), and for factor graphs (`FactorGraph`). The implementations are not necessarily efficient, but they should be relatively straightforward to understand.

Before getting started, familiarize yourself with the factor graph API. The provided source code contains a number of examples for the misconception factor graph that we discussed in the lecture. You may also run `help(<class>)` to browse the documentation of class `<class>` (e.g., try `help(Var)`).

For the CRF task, we use the *Reuters-128 dataset*; see the task description for details.

1 Factor Graphs

The goal of this assignment is to implement and experiment with various inference methods for factor graphs. We use the misconception example throughout; refer to the lecture slides for details.

A skeleton for each of the function that you are requested to implement is provided to you. A description of the parameters and expected result can be found in the Python file. Alternatively, execute `help(f)` to browse the documentation of some function f.

a) Implement a function `naive` that takes a factor graph and outputs the full joint distribution of its variables using naive inference.

b) Implement a function `eliminate` that takes a factor graph and uses variable elimination to eliminate a specified set of variables.

   Provide in your solution the factor graphs in which (i) $\{A\}$, (ii) $\{A, B\}$, and (iii) $\{A, B, C\}$ have been eliminated. If you have trouble implementing function `eliminate`, proceed manually and describe each step. Otherwise, it suffices to specify the factors.

c) Implement a function `gibbs` that takes a factor graph and uses Gibbs sampling to resample the values of a specified set of variables. In particular, to resample variable X, compute $P(X \mid \text{values of all other variables})$ and sample from this conditional distribution.

   Suppose $A = 1$, $B = 0$, $C = 1$, $D = 0$. Provide in your solution the conditional distributions for (i) A, (ii) B, (iii) C, and (iv) D conditioned on the just specified values of the other variables. If you have trouble implementing function `gibbs`, proceed manually and describe each step. Otherwise, it sufficies to specify the resulting distributions.

   Optional. The `gibbs` function takes an argument `log` to perform sampling in log space. You can ignore this argument. Optionally, implement `gibbs` such that when `log` is `True`, the computation of $P(X \mid \text{values of all other variables})$ is performed in log space.

d) The code provided to you contains a function `run_gibbs` to run Gibbs sampling and summarize the result. The function makes use of your implementation of `gibbs`. In this task, always run the function as follows

```
estimated_g = run_gibbs(Gm, Gm.vars, n, warmup=w, skip=s,
              marginals=True, normalize=True).
```

Here n, w, and s are non-negative integers; their meaning is described in the documentation of `run_gibbs` (try `help(run_gibbs)`). The function estimates the marginal distribution $P(X)$ of each variable in factor graph `Gm` based on n resamplings of each variable.

Investigate the quality of the estimated marginal distributions for various choices of n (keep $s = w = 0$ for now). To do so, compare the estimates with the exact result (code provided) as well as with the estimates provided from n independent samples (code provided). Discuss your findings.

Now fix $n = 1000$ and investigate the impact of values of s and w. Discuss your findings.

2 Factor Graphs and Naive Bayes

Consider the task of classifying handwritten digits from Assignment 2. As described in the lecture, we can convert a Naive Bayes model to a factor graph. The goal of this task is to discuss and optionally experiment with the resulting factor graph. The code provided to you contains a fit of a Naive Bayes model, code to convert it to a factor graph, and a function to plot the image corresponding to the current values of all variables. We use variable Y for the class label and variables X_i for each pixel. Let $\mathcal{X}$ (in code: `Xs`) be set of all pixel variables.

Answer each of the following questions briefly (1–3 sentences each). Your answer should relate to the above "Naive Bayes factor graph", not to factor graphs in general.

a) Suppose we use Gibbs sampling to sample $P(Y \mid \mathcal{X})$ once. Describe informally the result.

b) Suppose we compute n Gibbs samples $y_1, \ldots, y_n$ by sampling $P(Y \mid \mathcal{X})$ n times using Gibbs sampling. Are these n samples dependent or independent (conditioned on $\mathcal{X}$)? Discuss briefly.

c) Suppose we compute a Gibbs sample of $P(\mathcal{X} \mid Y)$. Describe informally the result.

d) Suppose we compute n Gibbs samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ by sampling $P(\mathcal{X} \mid Y)$ n times using Gibbs sampling. Are these n samples dependent or independent (conditioned on Y)? Discuss briefly.

e) Suppose we compute n Gibbs samples $(y_1, \boldsymbol{x}_1), \ldots, (y_n, \boldsymbol{x}_n)$ by sampling $P(\mathcal{X}, Y)$ n times using Gibbs sampling. These n samples will be dependent. Do you expect to see each class in roughly $n/10$ samples? Why or why not?

f) Can we run (in feasible time) naive inference on this factor graph? Can we eliminate all variables in $\mathcal{X}$? Or just variable Y? In each case, why or why not?

g) **Optional.** Use `run_gibbs` to experiment with Gibbs sampling on the digits factor graph. (Doing so will help you to answer the questions above, of course.)

3 Conditional Random Fields

The goal of this assignment is to apply linear-chain conditional random fields (CRF) to a simplified variant of the named entity recognition task. We use the *Reuters-128* dataset for this task. The dataset contains multiple documents, in which each word is annotated with a label indicating whether the word is part of a named entity (label=1) or not (label=0). The dataset has been (naively) preprocessed to be more easily accessible in Python. In more detail, we provide to you (1) a list of sentences, each consisting of a list of tokens, (2) the part-of-speech tag of each token, as well as (3) the label of each token. See the provided code and examples for more information. Our goal is to build a classifier that determines whether or not each token of a new sentence is part of a named entity.

You do not need to implement CRFs yourselves; we will make use the *pycrfsuite* library. The pycrfsuite library represents each CRF as a (conditional) log-linear model with indicator features.

Familiarize yourself with the code examples we provided to you. The examples read the dataset, split it into train and test, extract a number of features from each example, train a basic CRF, and evaluate the result (using a MAP estimate). Run all examples and investigate the obtained prediction performance (no need to write about this in your hand-in).

a) Provide additional features to the CRF that help to improve prediction performance. To do so, implement the function `get_features_improved` using function `get_features` as a starting point. Some suggestions for potentially useful additional features are given in the Python file. Which prediction performance can you achieve?

b) Investigate the importance of each feature for prediction by inspecting its weight (code provided). Which features are important? Is it intuitive to you why these features are important? Which of your newly added features from a) helped most?

c) **Optional.** The code provided to you contains a function to convert a fitted CRF to a factor graph (using our API from the previous tasks). Study the code that performs this conversion. Why do the resulting CRFs only have Y-variables, but no X-variables?

d) **Optional.** Perform Gibbs sampling on the factor graph obtained in task c) (code provided). Compare the Gibbs samples to (1) the true labels and (2) the MAP estimate of the *pycrfsuite* library (code provided). Discuss.