Artificial Intelligence Techniques - Lecture Notes

Notes to aid, not to replace. I'm simply a human, thus the risk you embrace.

Zohar Cochavi

Introduction

Not much needs to be said regarding the introduction of probabilistic machine learning. However, one rule deserves special attention and a quick reminder is therefore in place.

Bayes' Rule

In short, Bayes' rule allows us to 'update' our current belief, i.e. the probability of a state, S given some possible actions, $a_0, \dots, a_i$, $P(S|a_0, \dots, a_i)$, using observations. That is, by 'flipping' the dependence, actions given the next state instead of the next state given actions, we can relate actions and states.

Let me clarify by first providing Bayes' rule:

$$P(S|a) = \frac{P(a|S)P(S)}{P(a)} \tag{1}$$

Instead of using a for action, we can also use o for observation. When considering a classification example, this makes more sense. In that case, we would like to 'learn' how certain attributes relate to classes (the states in the previous example). This is done by making 'observations' in the form of features and, since we already have the label, this results in *an observation given some label*. Using Bayes' theorem (see eq. 1), we can thus determine the *label given some features*.

The example of actions and states applies when considering an **agent** acting in an unfamiliar environment. Here, we consider $P(S)$ our **prior** belief, or the belief before (prior to) the observation, and 'update' the belief by considering our belief given some observations, $P(S|o)$.

This implies somehow that $P(o|S)$ is easier to determine than its inverse. [...]

Bayesian Networks and Inference

Core to the idea of an intelligent agent is that the agent somehow manages to solve a task for which the solution is not explicitly embedded into the agent. If we told the agent explicitly what to do, it would simply be a front for some deterministic algorithm. The lack of determinism therefore implies the need for the agent to handle uncertainty. Therefore, we need to be able to formally represent uncertainty.

More formally, an issue arises when an agent attempts to determine whether it will be able to complete a task. It might guarantee a certain outcome, *given* that a whole host of events do, or do not, take place. This just kicks the can down the road, as we then have to guarantee whether the dependencies for the result are also satisfied. This leads to something called the **qualification problem**.

Combining some fundamental intuition about probability and the issue of the qualification problem, we can look at events as a chain of probabilistic dependencies. One way of structuring such dependencies is in something called a **Bayesian Network**.

Representing Knowledge in an Uncertain Domain

A Bayesian Network is a data structure that encapsulates the full joint probability distribution in a concise manner. Meaning it is nothing more than a representation of a (possibly very complex) function. Let's take the example of a patient having a cavity in their tooth, $P(Cavity)$ which might induce a toothache and can be observed with a 'catch' (see fig.).

```
graph TD;
    Cavity-->Toothache;
    Cavity-->Catch;
    Weather;
```

There, we see how one random variable might influence two other random variables. Furthermore, there is no need for all the nodes to be connected, a forest is a valid instance of a Bayesian network (the weather is independent of the cavity).

There is another assumption hidden in this graph, namely that $Toothache$ and $Catch$ are **conditionally independent** given $Cavity$. Conditional independence is formalized by the following relation.

$$P(Catch \wedge Toothache|Cavity) = P(Catch|Cavity) \wedge P(Toothache|Cavity)$$

A formal specification of a Bayesian Network (BN) can be characterized as follows:

1. Each node corresponds to a random variable, which may be discrete or continuous.

2. A set of directed links or arrows connects pairs of nodes. If there is an arrow from node X to node Y, X is said to be a parent of Y. The graph has no directed cycles (and hence is a directed acyclic graph, or DAG).
3. Each node X_i has a conditional probability distribution $P(X_i|Parents(X_i))$ that quantifies the effect of the parents on the node.

A more involved example would be the following, where an *Alarm* might be triggered by *Burglary* or an *Earthquake* which then proceeds to call John, *JohnCalls*, or Mary, $MaryCalls$.

```
graph TD;
    Burglary ---> Alarm;
    Earthquake ---> Alarm;
    Alarm ---> JohnCalls;
    Alarm ---> MaryCalls;
```

The **conditional probability table** (CPT) of *Alarm* can be found in tbl. 1. Since we are dealing with binary conditions (either A happens or it doesn't), $P(\not{A})$ is implicit. In other cases, the sum of the probabilities in one row should equal 1 to be considered valid probabilities.

Table 1: Conditional probability table (CPT) of the *Alarm* event. Notice how dependence increases the size of the table according to 2^n, where n is the number of dependent variables. This is because we are dealing with Boolean conditions.

B	E	P(A)
t	t	.95
t	f	.94
f	t	.29
f	f	.001

The Semantics of Bayesian Networks

As mentioned before, a BN is a representation of a joint distribution function. Another way to interpret it, however, is to view it as an encoding of a collection of conditional independence statements. Instead of focusing on how one is dependent on another, we think about which events are *not* dependent on another.

A BN can be described and used according to the following relation, which stems from the **chain rule**.

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i))$$

That is, the probability of a certain set of outcomes for the events described by the BN (i.e. the values of it's nodes) can be calculated by taking the product

of the probabilities of all these events given their respective parent(s). The conditional probability used in the product series can then be described by the aforementioned CPT.

This means that, even though the BN is technically equivalent to the joint probability distribution, assuming the network is sparse (i.e. not all nodes are connected to all other nodes) it is more efficient in terms of computation. Consider the following: if a joint probability distribution of Boolean random variables scales according to 2^n where n is the number of random variables, a BN scales according to $n2^k$ where k is the average number of parents.

Formally, we can say that a node is conditionally independent of its **non-descendents** given its parents. Furthermore, we can say that a node is conditionally independent of the whole network given its parents, children, and children's parents, also called its **Markov Blanket**. The more connected a BN is, the bigger the average Markov Blanket of its nodes.

Instead of considering all combinations of events, we assume that some events do not influence one another. At the core of the BN lies this assumption, and one should therefore consider the balance between accuracy and computational complexity.

Efficient Representation of Conditional Distributions

Assumptions can be made about the interaction of parents and how they relate to their children. We can assume logical or min/max statements, and from there construct the complete CPT. An example would be a $Fever$ which could be caused by the Flu, a $Cold$, or $Malaria$. We then assume that these relate to each other in something called a **fuzzy OR** scenario, i.e. the relation tends to that of a logical OR. This means that we would only need, for example, the probability of someone to get a $Fever$ from the $Cold$ and the probability to get a $Fever$ from the Flu to determine what the combined probability (see eq. 2)

$$P(Fever|Cold, Flu, \neg Malaria) = \\ (1 - P(\neg Fever|Cold, \neg Flu, \neg Malaria) \cdot P(\neg Fever|\neg Cold, Flu, \neg Malaria)) \tag{2}$$

Instead of using a CPT, we can also employ 'normal' probability density functions (PDFs) to describe the probabilistic characteristics of a random variable. While we could use discretization to work with continious random variables, this often comes at a loss of accuracy and potentially unwieldy CPTs. There are some considerations to make with regard to **hybrid networks** where we find both discrete and continuous random variables. Most notably, when a continuous parent has a discrete child, we have to create some threshold function. The most interesting here is the so-called **logistic function** which is often used in neural networks for the same purpose.

⋈ *Exact Inference*

Now my friends, the time has come, to collect info and probably infer some. We know how BNs work and how to construct them. Now, we would like to use our model to infer information given some state, i.e. perform a query.

In this section, we will only discuss *exact* inference. First, we discuss the 'easy' way, and then apply some optimization on top of that in the form of memoization.

Inference by Enumeration

Let's take the BN presented in fig. , and use the following query: $P(Burglary \mid JohnCalls = true, MaryCalls = true)$. Inference by enumeration simply takes all the possible contexts in which John might call, and where Mary might call and considers where this would have been caused by a burglary.

We can start solving our query using the following relation,

$$P(X \mid e) = \alpha P(X, e) = \alpha \sum_{y} P(X, e, y)$$

where $\alpha = \frac{1}{e}$.

Applying our query to the equation then gives,

$$P(B \mid j, m) = \alpha P(B, j, m) = \alpha \sum_{e} \sum_{a} P(B, j, m, e, a)$$

For the sake of simplicity, we only consider $B = true$, which can be rewritten to the following when taking our BN as the joint function (i.e. substitute $P(B, j, m, e, a$ with the characteristic of the BN).

$$P(b \mid j, m) = \alpha \sum_{e} \sum_{a} P(b)P(e)P(a \mid b, e)P(j \mid a)P(m \mid a)$$

Which can then be simplified as follows,

$$P(b \mid j, m) = \alpha P(b) \sum_{e} P(e) \sum_{a} P(a \mid b, e)P(j \mid a)P(m \mid a) \tag{3}$$

and then be solved by looking up the respective values in the CPTs.

Variable Elimination

One can imagine the previous algorithm to be suboptimal. Considering the time complexity, we come to the conclusion that it runs in $\mathcal{O}(2^n)$. Realizing your algorithm runs in exponential time is never a nice experience.. But, it's a great improvement considering calculating the straight joint probability runs in $\mathcal{O}(n2^n)$.

One way in which we can improve the performance of the model, is by reusing past calculations (memoization, if you know, you know). This dynamic programming technique is called **variable elimination**. The idea is to solve the equation from the bottom up (that is, when viewed as a syntax tree, we start at the leaves and move up) so that we can re-use calculations we've made before.

[...]

Probability Theory Refresh

Maintaining a Belief over Time

Introduction

- Changing the simple **sensor model** to a **transition model** by incorporating time steps, but no ability to which next state is the most likely (why? most probable is not most likely?).
- Different Types of models:
  - Hidden Markov Models
  - Kalman Filters
  - Dynamic Bayesian Networks
- Incorporating time can be important to, for example, take into account the rate of change (first derivative) when making decisions.

Transition and Sensor Models

- **Markov assumption**: *"The current state depends on only a finite fixed number of previous states."* Any system or process that satisfies this assumption is called a **Markov Process** or **Markov Chain**.
- The number of dependent prior states, n, is integrated into the so-called order, a so-called n**th-order Markdown Process**.
- Formally, a first-order Markov process is denoted as:

$$P(X_t|X_{0:t-1}) = P(X_t|X_{t-1})$$

- Important is that the underlying laws of the system do not change, therefore, the distributions are stationary over t.
- Another important assumption is that of the **sensor Markov assumption** (see eq. 4). That is, the observable variable is only dependent on the hidden variable in the current time-step. Also called the observation model.

$$P(E_t|X_{0:t}, E_{0:t-1}) = P(E_t|X_t) \tag{4}$$

- The process has to initialize with a prior probability at $t = 0$, $P(X_0)$.
- Thus, we can denote the complete joint probability distribution for a **first-order Markov Process** (see eq. 5).

$$P(X_{0:t}, E_{1:t}) = P(X_0) \prod_{i=1}^{t} P(X_i|X_{i-1})P(E_i|X_i) \tag{5}$$

- The accuracy of such a model depends on how 'reasonable' the Markov assumption is, and how closely the chain of causality resembles the real world (the probability that someone brings an umbrella might also be dependent on the amount of sun, instead of only whether it's raining.).
- There are two ways to improve the accuracy of a model (which are reformulations of one-another):

  1. Increase the order of the Markov process model.
  2. Increase the set of state variables.

Inference in Temporal Models

- Various methods for inference exist which can complement each other to improve accuracy of the model besides just answering queries.

  1. **Filtering**: Informs our agent about the distribution of the current hidden state based on the observations made until now, $P(X_t|e_{1:t})$.
  2. **Prediction**: Determine the distribution of the next hidden state based on the observations made until now, $P(X_{t+1}|e_{1:t})$.
  3. **Smoothing**: Determine the distribution of a past hidden state, $0 \geq kt$, based on the observations made until now, $P(X_k|e_{1:t})$.
  4. **Most likely explanation**: Determine the most likely sequence of events based on the observations made until now, $P(x_{1:t}|e_{1:t})$.

- Another technique called **Learning**, is based on **smoothing** combined with the **EM** algorithm.

Filtering And Prediction

- The process of filtering (see eq. 6) depends on prediction, (at least, based on the formulations mentioned above). (α is always a normalization constant).

$$P(X_{t+1}|e_{1:t+1}) = \alpha P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}|x_t)P(x_t|e_{1:t}) \tag{6}$$

- They are two steps that are necessary for one another. Therefore, while they are separate queries, they could be considered two steps in the same process. To get $P(X_{t+1}|e_{1:t+1})$,

1. Predict:
$$P(X_{t+1}|e_{1:t}) = \sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1})$$
2. Filter:
$$P(X_{t+1}|e_{1:t+1}) = \alpha P(e_{t+1}|X_{t+1})P(X_{t+1}|e_{1:t})$$

- The initialization predict step could be considered as prediction with empty evidence (i.e. without evidence), e_0. In which case it reduces to the prior, $P(x_0|e_0) = P(x_0)$.

$$\begin{aligned} P(X_1|e_0) &= \sum_{x_0} P(X_1|x_0)P(x_0|e_0) \\ &= \sum_{x_0} P(X_1|x_0)P(x_0) \end{aligned}$$

Smoothing

- With smoothing, we can essentially improve the model to take into account new observations, and can be described by eq. 7.

$$\begin{aligned} P(X_k|e_{1:t}) &= \alpha P(X_k|e_{1:k})P(e_{k+1:t}|X_k) \\ &= \alpha f_{1:k} \times b_{k+1:t} \end{aligned} \tag{7}$$

- The factors f and b, implying forward and backward respectively, can then be described by eq. 6 and eq. 8 respectively.

$$\begin{aligned} P(e_{k+1:t}|X_k) &= \sum_{x_{k+1}} P(e_{k+1}|x_{k+1})P(e_{k+2:t}|x_{k+1})P(x_{k_1}|X_k) \\ &\implies b_{k+1:t} = Backward(b_{k+2:t}, e_{k+1}) \end{aligned} \tag{8}$$

- While smoothing would take $O(t)$ for a single time-step, and thus $O(t^2)$ for all timesteps, by saving the intermediate results, we can smooth the whole sequence in $O(t)$ by applying the **forward-backward algorithm**.

Most-likely Sequence

- Since the most-likely sequence is formed by the joint probability of all the hidden states, we cannot just apply smoothing and calculate the most likely hidden state based on that states prior.

$$\begin{aligned} &\max_{x_1 \dots x_t} P(x_1, \dots, x_t, X_{t+1}|e_{1:t+1}) \\ &= \alpha P(e_{t+1}|X_{t+1}) \max_{x_t} \left(P(X_{t+1}|x_t) \max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, x_t|e_{1:t}) \right) \end{aligned} \tag{9}$$

- The algorithm to compute the most likely sequence is also called the **Viterbi algorithm** (see eq. 9).

Dynamic Bayesian Networks

- Dynamic Bayesian Networks (DBNs) are a generalization of the HMMs. Every HMM is a single-variable DBN; every discrete DBN is an HMM. **A DBN can have multiple 'sensors', while a HMM can have only one** (that is one observable).
- Exact inference in DBNs becomes intractable as their size grows. Thus, we use approximate inference. Specifically, **particle filtering**:

  First, a population of N initial-state samples is created by sampling from the prior distribution $P(X_0)$. Then, the update cycle is repeated for each time step:

  1. Each sample is propagated forward by sampling the next state value x_{t+1} given the current value x_t for the sample, based on the transition model $P(X_{t+1}|x_t)$.
  2. Each sample is weighted by the likelihood it assigns to the new evidence, $P(e_{t+1}|x_{t+1})$
  3. The population is resampled to generate a new population of N sample. Each new sample is selected from the current population; the probability that a particular sample is selected is proportional to its weight. The new samples are unweighted.

Learning

Why Learning

1. Some model might not be available. In this case, it can be built up over time by some other algorithm. An example would be to train a deep learning model; there is no *specific* version of the model available to the programmers, but we can adapt a general model to our specific purposes.
2. …
3. Not understanding the problem.

What is learning

While there are many approaches to learning, some of which might be more useful/correct in certain contexts than others. Here, we focus on **learning as induction**. How to use induction to learn is then dependent on the technique that is used to implement some agent.

Thus, while were are not explicitly building a model, we provide information to some statistical model to then make decisions. It is therefore important to decided what kind of information we should provide the model for it to make the right decision. Not only that, we also have to quantify what is the right decision.

We can split the space of what is the 'right' decision into two philosophies, the **Idealistic** and **Pragmatic** approach. The idealistic attempts to stay true to the way the world works, making observations and then using statistical models to

learn. The pragmatic approach, however, simply considers "whatever improves the performance".

While the idealistic perspective would have great accuracy, the tractability, however, is at odds. As the world is incredibly complex, the search space for hypothesis (often denoted as H) is also *huge*. Therefore, assumptions often need to be made, which brings most, if not any, model somewhere in between the two extremes.

Quantification of Objectives and Reward Hacking

- AI can be viewed as a model that is optimized according to a function that reflects 'intelligent behavior. (Backpropagation, Reinforcement Learning, Genetic Algorithms, etc.)

Fitness and Quantification

- This 'fitness' has to be adjusted based on the problem at hand. For example,
  - Sum of squared errors for regression.
  - *Cross-entropy* for classification (i.e. difference between two probability distributions).
- Because we want to compute how our 'current' model compares to previous ones, we want the function to be quantifiable. That is, quantifiable in both input and output. (not quite the right use of quantifiable)
- Quantifying is hard because you embed assumptions about the problem you are trying to solve in the data you supply to the algorithm, and *the algorithm is only as good as the data you supply it with.*

Reward Hacking

- **Reward Hacking**, or the **Inner Alignment** problem:

  The objective function admits some clever "easy" solution that formally maximizes it but perverts the spirit of the designer's intent (i.e. the objective function can be "gamed")
- **Goodhart's law** puts this behavior into perspective and reminds us that this is similar to human behavior:

  Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes.

  When a measure becomes a target, is ceases to be a good measure.
- Several reasons could exist for an agent not displaying the expected/desired behavior that are associated with reward hacking:
  1. The objective function is wrong.

2. The objective function is not properly evaluated (implementation or other practical issue besides the mathematical basis).
3. The objective is 'correct', but the agent could not learn the 'correct behavior'.

- All these concepts might be responsible for agents acting out. It's important to consider that *someone* has to then take responsibility for these problems.

Meaningful Human Control over AI

Humans not computers and their algorithms should ultimately remain in control of, and thus be *morally responsible* for relevant decisions.

- One approach to ensure humans can still bear responsibility to these systems, is to consider the **tracking** and **tracing** conditions:

  Tracking: The human-AI system (consisting of human agents, AI agents, infrastructures, etc.) should be able to track the relevant moral reasons of the relevant human agent(s) and be responsive to these reasons.

  Tracing: The system's behavior should be traceable to at least one human in the system design history or use context who can appreciate the capabilities of the system and their own role as morally responsible.

- Tracking and tracing can be further quantified into the following properties:

  1. The system has an explicit moral operational design domain (moral-ODD) and adheres to its boundaries.
  2. Humans and AI agents have appropriate and mutually compatible representations of each other and the task context.
  3. The agents' responsibility should be commensurate with their ability and authority to control the system.