Week 6: Decision Trees and Variational Autoencoders

http://mlvu.github.io

February 29, 2020

1 Decision trees

Imagine you do a newspaper round to help you get through these lean times. On your round, you encounter a number of dogs that either bark or (try to) bite. The dogs are described by the following binary features: *Heavy*, *Smelly*, *Big* and *Growling*. Consider the following set of examples:

Heavy	Smelly	Big	Growling	Bites
No	No	No	No	No
No	No	Yes	No	No
Yes	Yes	No	Yes	No
Yes	No	No	Yes	Yes
No	Yes	Yes	No	Yes
No	No	Yes	Yes	Yes
No	No	No	Yes	Yes
Yes	Yes	No	No	Yes

question 1: What is the entropy of the target value *Bites* in the data?

$$\mathrm{H}(\text{Bites}) = -\tfrac{5}{8}\log_2 \tfrac{5}{8} - \tfrac{3}{8}\log_2 \tfrac{3}{8} \approx 0.9544$$

question 2: Which attribute would the ID3 algorithm choose to use for the root of the tree (without pruning)? Growling

question 3: What is the information gain of the attribute you chose in the previous question? approximately 0.0487

question 4: Draw the full decision tree that would be learned for this data using ID3 without pruning.

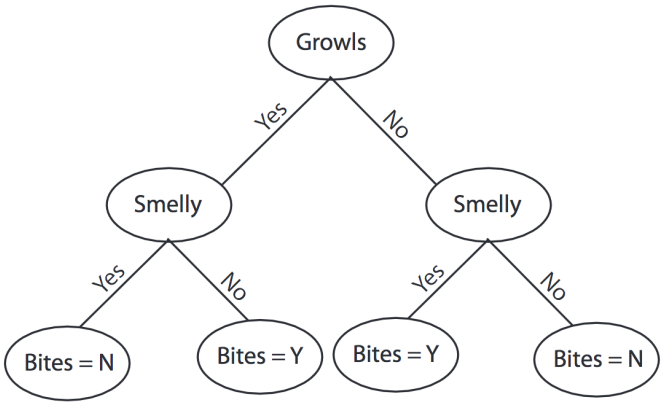

Figure 1: Decision tree

question 5: Suppose three new dogs appear in your round as listed in the table below. Classify them using the decision tree from the previous question.

Dog	Heavy	Smelly	Big	Growling	Bites
Buster	Yes	Yes	Yes	Yes	No
Pluto	No	Yes	No	Yes	No
Zeus	Yes	Yes	No	No	Yes

question 6: Someone proposes a new scheme to prevent overfitting: she suggests to set a pre-defined maximum depth for the decision trees. When the standard algorithm reaches this depth, it terminates. Could this help to prevent overfitting? Why (not)? It ensures smaller, simpler trees (a smaller model space) and therefore can reduce the risk of overfitting. Overfitting is essentially memorizing too much of your data. Smaller trees can memorize less.

question 7: In the maximum depth scheme introduced above, how would you determine a good value for the maximum depth for a given data set?

The maximum depth is a *hyperparameter*. We can choose good values for our hyperparameters by splitting our training data into a validation set and trying different values (either by cross validation or just single runs).

question 8: Why can't we apply L1-regularization to this decision tree learning problem?

L1 regularization assumes that your model is described by a real valued vector of parameters (i.e. your model space is continuous.) Decision trees have a *discrete* model space.

2 Variational autoencoders

The maximum likelihood principle tells us to optimize the quantity $p(x \mid \theta)$ as a function of θ (the model parameters).

For complex models, this does not usually lead to a closed form solution.

Instead, we will rewrite the maximum likelihood objective using the following decomposition.

$$\ln p(\mathbf{x} \mid \theta) = L(q, \theta) + KL(q, p)$$

with

$$\begin{aligned} q(\mathbf{z} \mid \mathbf{x}) &\text{ any distribution on } \mathbf{z} \\ KL(q, p) &\text{ the Kullback-Leibler divergence} \\ &\quad \text{between } q(\mathbf{z} \mid \mathbf{x}) \text{ and } p(\mathbf{z} \mid \mathbf{x}, \theta) \\ L(q, \theta) &= \mathbb{E}_q \ln \frac{p(\mathbf{x}, \mathbf{z} \mid \theta)}{q(\mathbf{z} \mid \mathbf{x})} \end{aligned}$$

We will first prove that this equality holds. We start with the right hand side, fill in the components, and derive the left-hand side.

question 9: Fill in the blanks. We have written everything in terms of *expectations* $\mathbb{E}$ to simplify the notation. The expectation is over the random variable $\mathbf{z}$, while $\mathbf{x}$ has some definite value. Note that $\mathbb{E}f(\mathbf{z}) + \mathbb{E}g(\mathbf{z}) = \mathbb{E}[f(\mathbf{z}) + g(\mathbf{z})]$.

$$\begin{aligned} L(q, \theta) + KL(q, p) &= \mathbb{E}_q \ln \frac{p(\mathbf{x}, \mathbf{z} \mid \theta)}{q(\mathbf{z} \mid \mathbf{x})} - \mathbb{E}_q \ln \frac{p(\mathbf{z} \mid \mathbf{x}, \theta)}{q(\mathbf{z} \mid \mathbf{x})} \\ &= \mathbb{E}_q \ln p(\mathbf{x}, \mathbf{z} \mid \theta) - \mathbb{E}_q \ln q(\mathbf{z} \mid \mathbf{x}) - \mathbb{E}_q \ln p(\mathbf{z} \mid \mathbf{x}, \theta) + \mathbb{E}_q \ln q(\mathbf{z} \mid \mathbf{x}) \\ &= \mathbb{E}_q \ln p(\mathbf{x}, \mathbf{z} \mid \theta) - \mathbb{E}_q \ln p(\mathbf{z} \mid \mathbf{x}, \theta) \\ &= \mathbb{E}_q \ln \frac{p(\mathbf{x}, \mathbf{z} \mid \theta)}{p(\mathbf{z} \mid \mathbf{x}, \theta)} = \mathbb{E}_q \ln \frac{p(\mathbf{z} \mid \mathbf{x}, \theta) p(\mathbf{x} \mid \theta)}{p(\mathbf{z} \mid \mathbf{x}, \theta)} \\ &= \mathbb{E}_q \ln p(\mathbf{x} \mid \theta) = \ln p(\mathbf{x} \mid \theta) \end{aligned}$$

In the EM algorithm, we search by alternately optimizing $L(q \mid \theta)$ with respect to θ and setting q equal to p (so that the KL term becomes zero).

question 10: For the variational autoencoder, we cannot (easily) perform this last step. Why not? In the variational autoencoder, our model is a neural network that transforms z into a distribution on x. To set the KL divergence term equal to zero, we would have to compute $p(z \mid x, \theta)$: i.e. a probability distribution on z that indicates for which z our observed x is most likely.

While sampling techniques exist to approximate this kind of distribution, they are costly and can be very inaccurate.

Instead, we *approximate* $p(z \mid x, \theta)$ with a neural network $q_v(z \mid x)$ that produces a distribution on z given some x. We call the neural network computing $p(x \mid x, \theta)$ $p_w(x \mid z)$, to make the notation a little more friendly. Here, w, stands for all parameters of the p network, and v stands for all parameters of the q network.[1]

This gives us an auto-encoder-like structure. An input is mapped to a distribution $q_v(z \mid x)$ by the encoder. We sample a single z from this distribution and pass it through the decoder $p_w(x \mid z)$ to produce a distribution on x (see the slides for diagrams).

To find a way to train such an architecture, we turn again to our decomposition of the likelihood. In our new notation:

$$\ln p_w(x) = L(v, w) + KL(q, p) \; .$$

The KL divergence term is difficult to compute: it's an expectation, and it contains the function $p_w(z \mid x)$ which requires us to invert the decoder neural network (that is, to reason about the inputs given the outputs).

However, because the KL divergence is always positive, we know that

$$\ln p_w(x) \geqslant L(v, w)$$

for *any* q_v we choose. This is why L is called the variational *lower bound*.[2] If we choose our parameters w, v to maximize L, we are also, indirectly, max-

[1] We've turned θ into w and added parameters v for our approximation q on the conditional distribution on z. We've also taken the parameters out of the conditional, because we will always talk about the function "given the parameter"; we will never talk about the probability on the parameters themselves.

imizing $\ln p_w(x)$.[3]

To do so, we rewrite $L(v, w)$ into two separate terms: a KL divergence and an expectation:

$$L(v, w) = -KL(q_v(z \mid x), p_v(z)) + \mathbb{E}_{q_v} \ln p_w(x \mid z)$$

question 11: Show that this equation holds. That is, rewrite the left part into the right. We will assume that all expectations are over q.

$$\begin{aligned} L(v, w) &= \mathbb{E}_q \ln \frac{p_w(x, z)}{q_v(z \mid x)} \\ &= \mathbb{E} \ln p_w(x, z) - \mathbb{E} q_v(z \mid x) \\ &= \mathbb{E} \ln [p_w(x \mid z) p_w(z)] - \mathbb{E} \ln q_v(z \mid x) \\ &= \mathbb{E} \ln p_w(x \mid z) + \mathbb{E} \ln p_w(z) - \mathbb{E} \ln q_v(z \mid x) \\ &= \mathbb{E} \ln p_w(x \mid z) - [\mathbb{E} \ln q_v(z \mid x) - \mathbb{E} \ln p_w(z)] \\ &= \mathbb{E} \ln p_w(x \mid z) - KL(q_v(z \mid x), p_w(z)) \end{aligned}$$

Thus, to optimize our variational autoencoder, we should maximize L. In other words, $-L$ is our loss function. The only problem left to solve is that the second term is an expectation (which we cannot compute explicitly).

question 12: How is this solved in practice?

We take a single sample from $q_v(z \mid x)$ and use $\ln p_w(x \mid z)$ as a (very crude) estimate of the expectation term.

To let the gradient propagate through the sampling, we add a sample from the standard MVN to the input and transform it to a sample from $q_v(z \mid x)$ by multiplying by a matrix A (with $\Sigma = AA^T$) and adding the mean.

[3] The word *variational* comes from the fact that one of its arguments, q, is a function (the calculus of functions is called *variational* calculus). For our purposes, this distinction doesn't matter much, since the function q is defined by a set of parameters v, so ultimately we will take the derivative over those parameters, as we are used to.

[3] How close the lower bound L comes to the true value $p_w(x)$ depends on how well our encoder network q_v approximates the true conditional distribution on z: $p_w(z \mid x)$. I.e. how small the KL term in the original decomposition is.