# Week 6: Decision Trees and Variational Autoencoders

http://mlvu.github.io

March 15, 2021

## 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 instances:

| Heavy | Smelly | Big | Growling | Bites |
|---|---|---|---|---|
| No | No | Yes | No | No |
| No | No | No | No | No |
| Yes | Yes | Yes | Yes | No |
| Yes | No | Yes | Yes | Yes |
| No | Yes | No | No | Yes |
| No | No | No | Yes | Yes |
| No | No | Yes | Yes | Yes |
| Yes | Yes | Yes | No | Yes |

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

We will model the target value as a random variable $B$ with values Yes and No. In decision trees, we estimate the probabiliies of these from the data by relative frequency. This gives us

$$p(B = \text{Yes}) = 3/8 \quad \text{and} \quad p(B = \text{No}) = 5/8$$

The entropy of a random variable is defined as $H(X) = -\sum_x p(x) \log_2 p(x)$, where the sum iterates over the possible values of the random variable. Here, that gives us $H(B) = -p(\text{Yes}) \log_2 p(\text{Yes}) - p(\text{No}) \log_2 p(\text{No})$.

Filling in our estimates, we get

$$\mathrm{H(B)} = -5/8 \log_2 5/8 - 3/8 \log_2 3/8 \approx 0.9544 \ .$$

Tip: You can punch this into a calculator, but you can also rewrite the expression a little first:

$$\begin{aligned}\mathrm{H(B)} &= -5/8 \log_2 5/8 - 3/8 \log_2 3/8 \approx 0.9544 \\ &= -5/8(\log 5 - \log 8) - 3/8(\log 3 - \log 8) \\ &= -\frac{1}{8}(5(\log 5 - 3) + 3(\log 3 - 3))\end{aligned}$$

This has several benefits. First, you get nice round numbers for any numerator or denominator that is a power of 2, so you may end up with a more accurate answer. More importantly, this allows you to double-check your answers. It's easy to make mistakes in the computation of the entropy, so it pays to compute it in different ways, to guard against mistakes.

Another way to double-check your answer is to test it against your intuitive understanding of the wntropy. Remember, the entropy is the expected codelength under p, if we use the code that corresponds to p. First of all that means the answer can't be negative. It also means that if you have two outcomes, you can always assign them the codewords 0 and 1, and get an expected codelength of 1, whatever the probabilities. This means that the optimal codelength, in this case must be between 0 and 1. Since there is a little imbalance in the classes, but not much, we expect the entropy to be a little below 1 (the maximum entropy, which we get for a 50/50 probability).

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

The information gain is the average drop in entropy we get by splitting on a feature. We're considering the four features Heavy, Smelly, Big and Growling as the first split in the tree. Since the entropy before the split is the same for all four, we can ignore that and only look at which value gives us the lowest average entropy post-split.

We start by tallying up the class frequencies after the split

| | value/class | Yes | No |
|---|---|---|---|
| Heavy: | Yes | 2 | 1 |
| | No | 3 | 2 |

| | value/class | Yes | No |
|---|---|---|---|
| Smelly: | Yes | 2 | 1 |
| | No | 3 | 2 |

| | value/class | Yes | No |
|---|---|---|---|
| Big: | Yes | 3 | 2 |
| | No | 2 | 1 |

| | value/class | Yes | No |
|---|---|---|---|
| Growling: | Yes | 3 | 1 |
| | No | 2 | 2 |

Tip: It's easy to make a mistake in these tallies. You can double-check your work by computing the marginal sums of the tables: the vertical margin should correspond to the totals per value (which isn't the same for all features) and the horizontal margin should correspond to the the call totals. If you start by filling in the margins (which are usually easier to tally up), then you only need to count one value/class combination, and you can work out the rest from the margins.

ID3 chooses the split that gives us the maximal **information gain**. We can look up the formula and plug in the probability estimates from the data, but it's a good idea to think about the process a little before we start calculating. This will guard against mistakes, and it will save us a lot of computation.

Here is the formula for the information gain:

$$I(F) = H(S) - \sum_{v \in F} \frac{|S_v|}{|S|} H(S_v) \, .$$

Where $F$ is the feature we are splitting on $S$ is the set if instances before the split, $v$ iterates over the values of $F$, with subsets $S_v$ representing those instances of $S$ that have value $v$ for feature $F$, and $H(S)$ is the entropy of the class distribution among the instances in the set $S$.

Since we are computing the first split in the tree $H(S)$ is the entropy of the class distribution over the whole data. We computed this in the last question, but we actually don't need it. It's the same for all features, so it doesn't affect which feature minimizes the information gain: we can compute only the second term.

We can see that the tallies for Heavy and Smelly are the same, so we'll only need to compute one of them. With a little more thinking, we can see that that the information gain for Big muast *also* be the same. First, not that the entropy of a 3/2 split is the same as that of a 2/3 split: it's just reversing the two terms in the formula. Second, Heavy has 3/8 Yes and 5/8 and No, and Big simply reverses these proportions, but with the same entropies. Thus, for the first three features, the information gain is the same.

Here, it's difficult to say whether we expect Growling to do better or worse than the other three. The split after Growling=No is uniform, so that's the highest in all 8 possible splits, but the split Growling=Yes is the most uneven in all possible splits. Here, we have to calculate to be sure, but

in the questions in the practice exam, you can usually tell immediately from the tallies which split gives you the best gain.

Filling in the formula for Big:

$$\begin{aligned}
\mathrm{I}(\text{Big}) &= \mathrm{H}(\mathrm{S}) - \frac{5}{8}\mathrm{H}(\mathrm{S}_{\text{Yes}}) - \frac{3}{8}\mathrm{H}(\mathrm{S}_{\text{No}}) \\
&= \mathrm{H}(\mathrm{S}) - \frac{5}{8}\left(-\frac{2}{5}\log\frac{2}{5} - \frac{3}{5}\log\frac{3}{5}\right) - \frac{3}{8}\left(-\frac{1}{3}\log\frac{1}{3} - \frac{2}{3}\log\frac{2}{3}\right) \\
&= \mathrm{H}(\mathrm{S}) - 0.9512\ .
\end{aligned}$$

Be careful to keep track of the minus in the entropy. It's common mistake to forget about it. Note that the logarithms of values between 0 and 1 produce negative values.

And for Growling:

$$\begin{aligned}
\mathrm{I}(\text{Growling}) &= \mathrm{H}(\mathrm{S}) - \frac{4}{8}\mathrm{H}(\mathrm{S}_{\text{Yes}}) - \frac{4}{8}\mathrm{H}(\mathrm{S}_{\text{No}}) \\
&= \mathrm{H}(\mathrm{S}) - \frac{1}{2}\left(-\frac{3}{4}\log\frac{3}{4} + \frac{1}{4}\log\frac{1}{4}\right) - \frac{1}{2}\left(-\frac{2}{4}\log\frac{2}{4} - \frac{2}{4}\log\frac{2}{4}\right) \\
&= \mathrm{H}(\mathrm{S}) - \frac{1}{2}\left(-\frac{3}{4}\log\frac{3}{4} - \frac{1}{4}\log\frac{1}{4}\right) - \frac{1}{2}\cdot 1 \\
&= \mathrm{H}(\mathrm{S}) - 0.9056\ .
\end{aligned}$$

There's not much in it, but the information gain for Growling is slightly higher, which is what we've looking for, so we pick Growling as the first split.

**question 3:** What is the information gain of the attribute you chose in the previous question? Using our answer to the first question to fill in $\mathrm{H}(\mathrm{S})$, we get $0.9544 - 0.9056 = 0.0487$ as an approximate answer.

**question 4:** Draw the full decision tree that would be learned for this data using ID3 without pruning. The split on Growling partitions the data into the following subsets.

For Growling=Yes:

| Heavy | Smelly | Big | Growling | Bites |
|---|---|---|---|---|
| Yes | Yes | Yes | Yes | No |
| Yes | No | Yes | Yes | Yes |
| No | No | No | Yes | Yes |
| No | No | Yes | Yes | Yes |

And for Growling=No:

| Heavy | Smelly | Big | Growling | Bites |
|---|---|---|---|---|
| No | No | Yes | No | No |
| No | No | No | No | No |
| No | Yes | No | No | Yes |
| Yes | Yes | Yes | No | Yes |

At this point, we can take a shortcut. This will help to verify our calculated answer, or to gues the answer quickly if we're running out of time. In both tables, the values of Smelly correspond exactly to one of the classes. After splitting on Smelly, in all cases the resulting subset of the data contains only one class. This means that the distribution on the classes is 0/1 and the entropy is 0. This isn't true for any of the other features, so **Smelly must be the best feature for both branches**.

To double-check this intuition, we'll do the full calculation anyway. For each branch in the tree so far, we need to decide the next split separately. We'll start with Growling=Yes. The features left are Heavy, Smelly and Big. These are the tallies:

| Heavy: | Yes | No |
|---|---|---|
| Yes | 1 | 1 |
| No | 2 | 0 |

| Smelly: | Yes | No |
|---|---|---|
| Yes | 0 | 1 |
| No | 3 | 0 |

| Big: | Yes | No |
|---|---|---|
| Yes | 2 | 1 |
| No | 1 | 0 |

As before, they all have the same entropy $H(S_{\text{Growling=Yes}})$ on the incoming instances, so we won't compute that. Here are the information gains. Note that if the class distribution is 0/1 the entropy is 0 and if it's uniform, the entropy is 1.

$$\begin{aligned}
I(\text{Heavy}) &= H(S) - \left(\frac{2}{4}1 + \frac{2}{4}0\right) \\
&= H(S) - \frac{1}{2} \\
I(\text{Smelly}) &= H(S) - \left(\frac{1}{4}0 + \frac{3}{4}\right) \\
&= H(S) \\
I(\text{Big}) &= H(S) - \left(\frac{3}{4}\left(-\frac{1}{3}\log\frac{1}{3} - \frac{2}{3}\log\frac{2}{3}\right) + \frac{1}{4}0\right) \\
&= H(S) - 0.6887
\end{aligned}$$

Our guess is validated. Smelly has the highest information gain on this branch.

Next up, the branch with Growling=No. Here are the tallies:

| | | Yes | No |
|---|---|---|---|
| Heavy: | Yes | 1 | 0 |
| | No | 1 | 2 |

| | | Yes | No |
|---|---|---|---|
| Smelly: | Yes | 2 | 0 |
| | No | 0 | 2 |

| | | Yes | No |
|---|---|---|---|
| Big: | Yes | 1 | 1 |
| | No | 1 | 1 |

And here are the information gains.

$$\begin{aligned}
\mathrm{I}(\text{Heavy}) &= \mathrm{H}(S) - \left(\frac{1}{4}0 - \frac{3}{4}\left(-\frac{1}{3}\log\frac{1}{3} - \frac{2}{3}\log\frac{2}{3}\right)\right) \\
&= \mathrm{H}(S) - 0.6887 \\
\mathrm{I}(\text{Smelly}) &= \mathrm{H}(S) - \frac{1}{4}0 - \frac{3}{4}0 \\
&= \mathrm{H}(S) \\
\mathrm{I}(\text{Big}) &= \mathrm{H}(S) - \frac{2}{4}1 - \frac{2}{4}1 \\
&= \mathrm{H}(S) - 1
\end{aligned}$$

Here too, we see that Smelly has the highest information gain.

This gives us a tree with four leaf nodes. In each of the four resulting subsets of the data, all instances have the same class, so four all leaf nodes, we have reached a stop condition and we can stop expanding. We label the leaf nodes with the majority class in the subset:

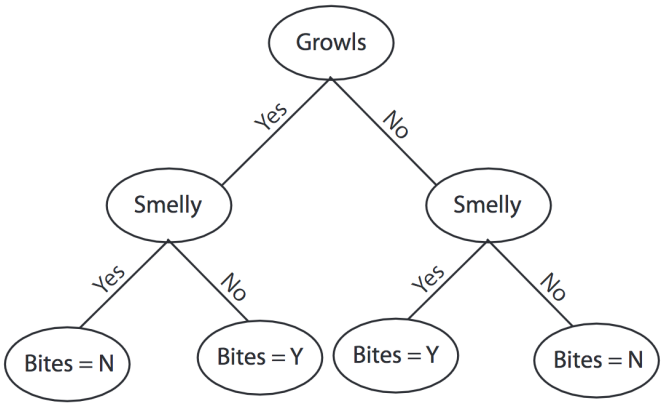


Figure 1: Decision tree

**question 5:** Suppose three new dogs appear in your round as listed in the table below. Classify them using the 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 could: 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.

Another way to think of this is that a smaller tree is likely to have a large subset of nodes at its leaves, because it can only split the data so many times. All theese nodes need to be assigned the same label, so if they don't all have the same label in the data, we can't overfit on these nodes.

**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 $\theta$ (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

$$
\begin{aligned}
q(z \mid x) &\text{ any distribution on } z \\
KL(q, p) &\text{ the Kullback-Leibler divergence}[1] \\
&\quad \text{between } q(z \mid x) \text{ and } p(z \mid x, \theta) \\
L(q, \theta) &= \mathbb{E}_q \ln \frac{p(x, z \mid \theta)}{q(z \mid x)}
\end{aligned}
$$

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 $z$, while $x$ has some definite value. Note that $\mathbb{E}f(z) + \mathbb{E}g(z) = \mathbb{E}[f(z) + g(z)]$.

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

If you're struggling with this derivation, you can always memorize it, in case it appears on the exam, but if you want to understand it it helps to remember the following properties of expectations, logarithms and probabilities. To see where they come from, review the first homework exercise.

[1] The KL divergence can be defined using $\log_2$ or using $\ln$. The first leads to a value in *bits*, the second to a value in the slightly more abstract unit of *nats*. That is, the information distance between the two distributions is the same, but expressed in different units. In the context of the VAE, the natural logarithm $\ln$ is preferred, mainly because it cancels out neatly against the exponent in the pdf of the normal distribution.

$$p(a, b) = p(a \mid b)\, p(b)$$
$$\log(f(x)g(x)) = \log f(x) + \log g(x)$$
$$\log \frac{f(x)}{g(x)} = \log f(x) - \log g(x)$$
$$\mathbb{E}\left[f(x) + g(x)\right] = \mathbb{E}f(x) + \mathbb{E}g(x)$$
$$\mathbb{E}c = c \text{ if } c \text{ is a constant with respect to the distribution of } \mathbb{E}$$

In EM, we search by alternately (1) optimizing $L(q \mid \theta)$ with respect to $\theta$ and (2) 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 $p(x \mid z, \theta)$ $p_w(x \mid z)$, 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.[2]

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).

[2] We've turned $\theta$ 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.

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

$$\ln p_{\boldsymbol{w}}(\boldsymbol{x}) \geqslant L(\boldsymbol{v}, \boldsymbol{w})$$

for *any* $q_{\boldsymbol{v}}$ we choose. This is why L is called the variational *lower bound*.[3] If we choose our parameters $\boldsymbol{w}, \boldsymbol{v}$ to maximize L, we are also, indirectly, maximizing $\ln p_{\boldsymbol{w}}(\boldsymbol{x})$.[4]

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

$$L(\boldsymbol{v}, \boldsymbol{w}) = -KL(q_{\boldsymbol{v}}(\boldsymbol{z} \mid \boldsymbol{x}), p_{\boldsymbol{v}}(\boldsymbol{z})) + \mathbb{E}_{q_{\boldsymbol{v}}} \ln p_{\boldsymbol{w}}(\boldsymbol{x} \mid \boldsymbol{z})$$

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

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

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 $q_{\boldsymbol{v}}(\boldsymbol{z} \mid \boldsymbol{x})$ and use $\ln p_{\boldsymbol{w}}(\boldsymbol{x} \mid \boldsymbol{z})$ 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 $q_{\boldsymbol{v}}(\boldsymbol{z} \mid \boldsymbol{x})$ 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 $\boldsymbol{v}$, so ultimately we will take the derivative over those parameters, as we are used to.

[4] How close the lower bound L comes to the true value $p_{\boldsymbol{w}}(\boldsymbol{x})$ depends on how well our encoder network $q_{\boldsymbol{v}}$ approximates the true conditional distribution on $\boldsymbol{z}$: $p_{\boldsymbol{w}}(\boldsymbol{z} \mid \boldsymbol{x})$. I.e. how small the KL term in the original decomposition is.