Generative Learning Algorithms

So far, we've mainly been talking about learning algorithms that model $p(y \mid x; \theta)$, the conditional distribution of y given x. For instance, logistic regression modeled $p(y \mid x; \theta)$ as $h_\theta(x) = g(\theta^T x)$ where g is the sigmoid function. In these notes, we'll talk about a different type of learning algorithm.

Consider a classification problem in which we want to learn to distinguish between elephants $(y = 1)$ and dogs $(y = 0)$, based on some features of an animal. Given a training set, an algorithm like logistic regression or the perceptron algorithm (basically) tries to find a straight line——that is, a decision boundary——that separates the elephants and dogs. Then, to classify a new animal as either an elephant or a dog, it checks on which side of the decision boundary it falls, and makes its prediction accordingly.

Here's a different approach. First, looking at elephants, we can build a model of what elephants look like. Then, looking at dogs, we can build a separate model of what dogs look like. Finally, to classify a new animal, we can match the new animal against the elephant model, and match it against the dog model, to see whether the new animal looks more like the elephants or more like the dogs we had seen in the training set.

Algorithm that try to learn $p(y \mid x; \theta)$ directly (such as logistic regression), or algorithms that try to learn mappings directly from the space of inputs $\mathcal{X}$ to the labels $\{0, 1\}$, (such as the perceptron algorithm) are called **discriminative learning algorithms**.

Here, we'll talk about algorithms that instead try to model $p(x \mid y)$ (and $p(y)$). These algorithms are called **generative learning algorithms**. For instance, if y indicates whether a example is a dog (0) or an elephant (1), then $p(x \mid y = 0)$ models the distribution of dog's features, and $p(x \mid y = 1)$ models the distribution of elephants' fearures.

After modeling $p(y)$ (called the **class priors**) and $p(x \mid y)$, our algorithm can then use Bayes rule to derive the posterior distribution on y given x:

$$p(y \mid x) = \frac{p(x \mid y)p(y)}{p(x)}$$

Here, the denominator is given by $p(x) = p(x \mid y = 1)p(y = 1) + p(x \mid y = 0)p(y = 0)$ and thus can also be expressed in terms of the quantities $p(x \mid y)$ and $p(y)$ that we've learned. Actually, if we're calculating $p(y \mid x)$ in order to make a prediction, then we don't actually need to calculate the denominator, since

$$
\begin{aligned}
arg\max_y p(y|x) &= arg\max_y \frac{p(x|y)p(y)}{p(x)} \\
&= arg\max_y p(x|y)\,p(y)
\end{aligned}
$$

```
In [ ]:
```