Contents

1 Learning

1.1 The machine analogy

We assume a machine which has a configuration specified by a set of parameters Θ that produces some output Data D. We understand how the machine works so the probability of some D given θ is known. But since we observe the data we want to know the probability of some configuration θ given the data D. The machine here in practice is often a probability distribution.

1.2 Frequentist learning

Definition 1.1 (Frequentist learning)**.** In frequentist learning we are given some data and our job is to guess the true model (out of a set of models) that generated some data. In other words, we want to pick the right θ so that the probability distribution fits the data best.

Definition 1.2 (Maximum likelihood)**.** The maximum likelihood estimate/principle is the model criteria for which we guess the model for which the probability of seeing the data that we saw is the highest. Formally, this is given by

$$\hat{\theta} = \arg\max_{\theta} P(D|\theta)$$

this just expresses that our configuration $\hat{\theta}$ is the one that maximizes the probability of seeing the data D given θ.

Coinflip example - model selection problem

There is a straight and a bent coin. Without telling us a friend picks up one of the coins and flips it 12 times. This yields the following probabilities.

$$p(\text{Heads} \mid \text{Straight}) = 1/2 \qquad p(\text{Heads} \mid \text{Bent}) = 4/5$$

$$p(\text{Tails} \mid \text{Straight}) = 1/2 \qquad p(\text{Tails} \mid \text{Bent}) = 1/5$$

HTH HHT HHT HTH

Some notes regarding this example:

- the model class conists of the two coins: straight and bent
- the data consits of 12 instances: the 12 coin flips
- the coins are a Bernoulli distribution with the parameters $\frac{1}{2}$ and $\frac{4}{5}$

The maximum liklihood estimation here can be expressed as:

$$\arg\max_{\text{Coin} \in \{\text{Bent,Straight}\}} p(\text{HTHHHTHHTHTH} \mid \text{Coin})$$

$$\arg\max_{\text{Model} \in \text{Model space}} p(D \mid \text{Model})$$

Here we just compute $p(D \mid \text{Bent})$ and $p(D \mid \text{Straight})$ and pick the one that is higher. We have that

$$p(D \mid \text{Bent}) \approx 0.000286$$
$$p(D \mid \text{Straight}) \approx 0.000244$$

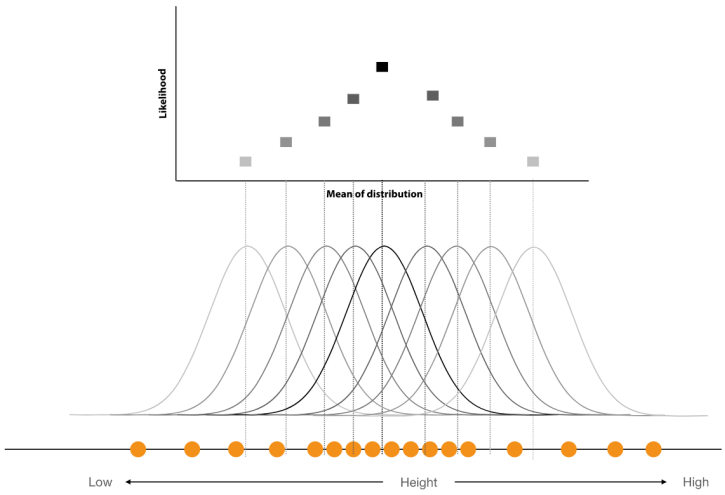
From which we can conclude that we would prefer the bent coin model for the observed data.

Estimating mean example

Another example would be wanting to find the maximum likelihood estimate for the mean of a normal distribution given some data.

1.3 Log-likelihood (loss)

Definition 1.3 (log-likelihood)**.** The log-likelihood is the logarithm of the maximum likelihood function. It shares the maxima and is easier to symbolically maniuplate. It also provides a smoother loss surface for methods like gradient descent. Usually we take the *negative* log-likelihood, so that we can preform gradient descent to find the optimum.

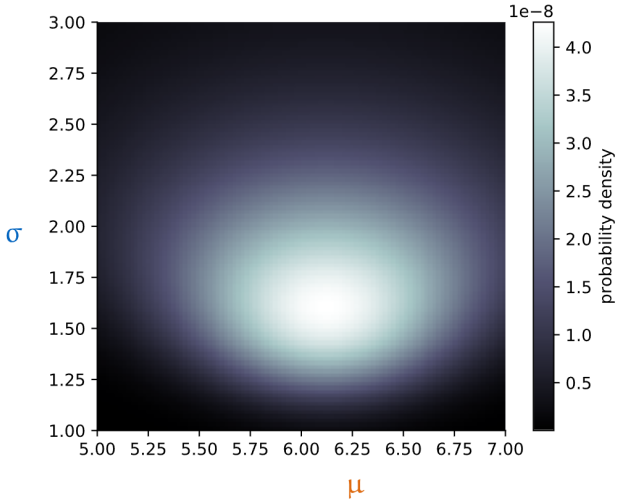


1.4 Maxmium likelihood for normal distribution

Formally we can express the maximum likelihood estimate for the mean of a normal distribution as follows.

$$\begin{aligned}
\arg\max_{\theta} \ln p(X \mid \theta) &= \arg\max_{\theta} \ln \prod_{x \in X} p(x \mid \theta) \\
&= \arg\max_{\theta} \sum_{x \in X} \ln p(x \mid \theta) \quad p(x \mid \theta) = N(x \mid \mu, \sigma) \\
&= \arg\max_{\mu,\sigma} \sum_{x} \ln \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right) \right] \\
&= \arg\max_{\mu,\sigma} \sum_{x} \ln \left[\frac{1}{\sqrt{2\pi\sigma^2}} - \frac{(x-\mu)^2}{2\sigma^2} \right]
\end{aligned}$$

We can graph the landscape of the log-likelihood function as follows.



Some notes regarding this landscape:

- The mean and variance that give us a model which best fits our data are in the bright spot.
- If we don't want to do an analytic solution we could find the optimum by using gradient descent or random search.

Removing parameters

If we remove each parameter we can further simplify the problem, for example the maximum likelihood estimate for just the mean is given by

$$\begin{aligned}
\arg\max_{\mu} \sum_{x} \ln \mathcal{N}(x \mid \mu, \sigma) &= \arg\max_{\mu} \sum_{x} -\frac{(x-\mu)^2}{2\sigma^2} \quad \text{(constant rule)} \\
&= \arg\max_{\mu} -\frac{1}{2\sigma^2} \sum_{x} (x-\mu)^2 \quad \text{(constant rule)} \\
&= \arg\max_{\mu} -\sum_{x} (x-\mu)^2 \quad \text{(constant rule)} \\
&= \arg\min_{\mu} \sum_{x} (x-\mu)^2
\end{aligned}$$

The maximum likelihood estimate for the mean is just the value that minimizes the sum of the squared differences between the data and the mean. The key insight here being that even if a likelihood function looks complex its often the case when you take the logarithm and maximize it the complexity dissapears.

1.5 Bayesian learning

We can break the bayesian learning formula into the following components:

$$\underbrace{p(\theta \mid D)}_{\text{posterior d.}} = \underbrace{p(D)^{-1}}_{\text{model evidence}} \underbrace{p(D \mid \theta)}_{\text{data d.}} \underbrace{p(\theta)}_{\text{prior d.}}$$

Where

- posterior distribution: a distribution over all possible models given some data we observed D
- prior distribution: our belief about the model before we see the data

Coinflip example

Using the coinflip example again the first thing we want to calculate is the prior distribution. We can express this as:

$$\begin{aligned}
p(D) &= p(D, \text{Bent}) + p(D, \text{Straight}) \\
&= p(D \mid \text{Bent})p(\text{Bent}) + p(D \mid \text{Straight})p(\text{Straight})
\end{aligned}$$

We can then sub the above result into the bayesian learning formula to get the posterior distribution for the Straight model as:

$$\begin{aligned}
p(\text{Straight} \mid D) &= \frac{p(D \mid \text{Straight})p(\text{Straight})}{p(D)} \\
&= \frac{p(D \mid \text{Straight})p(\text{Straight})}{p(D \mid \text{Bent})p(\text{Bent}) + p(D \mid \text{Straight})p(\text{Straight})}
\end{aligned}$$

As a reminder here we are simply expressing the probability of the coin being straight as a cause for the data we observed. In the case of both models being equally likely we can simplify the above expression to

$$p(\text{Straight} \mid D) = \frac{p(D \mid \text{Straight})}{p(D \mid \text{Bent}) + p(D \mid \text{Straight})}$$

1.6 Bayesian vs Frequentist applied

In the case of the bayesian approach we get a distribution over the model space. It tells us both that the Bent model is the more likely cause of our data and that the Straight model while less likely is still reasonably possible.
In the case of the frequentist approach we only get a point estimate. It tells us that the Bent model is the more likely cause of our data but it doesn't tell us anything about the Straight model. Thus making it a less useful analysis overall. The benefit of the frequentist approach is that it is often easier to compute and understand.

2 Naive Bayes Classifier

Definition 2.1 (Probalistic classifier)**.** A classifier that returns a probability over all classes.

A basic abstract example, given a random variable X for instances where $X = x_1, x_2, \cdots, x_n$ and Y as a random variable for a positive or negative class. We can express a probalistic classifier as follows:

$$p(Y = \text{pos} \mid X) = 0.1 \quad p(Y = \text{neg} \mid X) = 0.9$$

Definition 2.2 (Generative classifier)**.** A generative classifier focuses learning a distribution on the feature space given the class $p(X = s \mid Y)$, which is then combined with Bayes' rule to get the probability over the classes conditioned on the data.

There are 3 main types of generative classifiers:

- Bayes optimal classifier: Marginalizze over all classifiers in a model class. Provably optimal (given certain assumptions). Usually too expensive to compute.
- Bayes classifier: Learn single distribution $P(X \mid Y)$. Reasonable approach for low-dimensional data.
- Naive Bayes classifier: Assume conditionally independent features. Simple, cheap and effective for high-dimensional data.

Definition 2.3 (Discriminative classifier)**.** A discriminative classifier learns the function $p(Y \mid X = x)$ with X as input and class probabilities as output. It functions as a kind of regression, mapping x to a vector of class probabilities.

2.1 Bayes classifier

We can explain a basic binary bayes classifier as follows, say we want to caclulate the probability of some data beloning to a class pos given some instance X. We express this probability as the regular bayes rule:

$$p(\text{pos} \mid x) = \frac{p(x \mid \text{pos})p(\text{pos})}{p(x)} = \frac{p(x \mid \text{pos})p(\text{pos})}{p(x \mid \text{pos})p(\text{pos}) + p(x \mid \text{neg})p(\text{neg})}$$

We can see to then calculate $p(\text{pos} \mid x)$ we need to calculate $p(x \mid y)$ and $p(y)$ for both classes. So the task becomes to learn functions for those two probabilities.

Fitting Multivariate Normal Distribution (MVN)

If we want to fit a multivariate normal distribution to some dataset of N features we can work our the vector μ representing the sample mean and the matrix Σ representing the covariance matrix.

Summary steps

We start off by chosing a class of prob. distributions M (e.g. MVNs) which includes:

- Fit params μ_p, Σ_p to all positive points: $p(x \mid \text{pos}) = N(x \mid \mu_p, \Sigma_p)$
- Fit params μ_n, Σ_n to all negative points: $p(x \mid \text{neg}) = N(x \mid \mu_n, \Sigma_n)$

We then estimate $p(y)$ from the class frequencies in the training data, or use domain specific information. Finally we compute the clss probabilities using bayes rule.

2.2 Naive Bayes classifier

Here we assume that all features are conditionally independent on the given class. Its more simplistic than the Bayes classifier but computationally less expensive. Formally we can express this as:

$$p(X_1, X_2 \mid Y) = p(X_1 \mid Y)p(X_2 \mid Y)$$

This type of classifier is often used with categorical features.
For the full Naive bayes classifier we apply the following thought process:

$$\begin{aligned} p(Y \mid X_1, \ldots, x_n) &\propto p(X_1, \ldots, X_n \mid Y)p(Y) \\ &= p(X_1 \mid Y) \times \ldots \times p(X_n \mid Y)p(Y) \end{aligned}$$

Spam classification example

In our example we have a dataset with binary features. The instances are emails, the target classes are spam or ham, and each feature indicates the presence of a word. In a table this is expressed as:

"pill"	"meeting"	
T	T	spam
T	F	spam
T	T	ham
T	T	ham
F	T	ham
F	T	ham
F	T	ham
F	F	spam
T	F	spam
F	F	spam
F	F	ham

We want to built a generative classifier, so one which aims to learn the distribution of a feature space given the class. By naive bayes we can do this for each feature independently and just multiply the probabilities. For our example, assuming $X_1 = \text{pill}$ and $X_2 = \text{money}$ we express the probabilities as:

$$\begin{aligned} p(X_1 = T \mid \text{ham}) &= \frac{2}{6} \\ p(X_1 = F \mid \text{ham}) &= \frac{4}{6} \\ p(x_2 = T \mid \text{spam}) &= \frac{3}{5} \\ p(x_2 = F \mid \text{spam}) &= \frac{2}{5} \end{aligned}$$

Now we can use these probabilities to calculate the probability of a new email being spam or not, for example say we had a new email with the features 'pill' and 'meeting'.
The probability of it being ham is proportional to the probability of seeing a ham email with the observed features and the probability of it being a ham email to begin with. Formally we can express this as:

$$\begin{aligned} p(\text{ham} \mid X_1 = T, X_2 = T) &\propto p(X_1 = T, X_2 = T \mid \text{ham}) p(\text{ham}) \\ &= p(X_1 = T \mid \text{ham}) p(X_2 = T \mid \text{ham}) p(\text{ham}) \\ &= \frac{2}{6} \times \frac{5}{6} \times \frac{6}{11} \end{aligned}$$

Laplace smoothing

Naive bayes can run into issues with for some features a particular value does not occur thus giving a probability of 0. Its a problem because it causes the entire naive bayes probability to be 0. We can solve this by using smoothing.

Definition 2.4 (pseudo-observations)**.** Pseudo-observations are a type of smoothing where for each possible value, we add one instance where all features have that value. In a large dataset the impact of this is minimal and there exist further techniques to minimize the impact even more.

unsmoothed

$$p(X_1 = T \mid Y = \text{spam}) = \frac{\text{freq. of T in spam data}}{\text{total \# of spam instances}}$$

smoothed

$$p(X_1 = T \mid Y = \text{spam}) = \frac{\text{freq. of T in spam} + 1}{\text{total \# of spam instances} + v}$$

λ-smoothed

$$p(X_1 = T \mid Y = \text{spam}) = \frac{\text{freq. of T in spam} + \lambda}{\text{total \# of spam instances} + \lambda v}$$

We can use λ-smoothing to further minimize the impact of the pseudo-observations by reducing their weight among the other observations. Comparing unsmoothed, smoothed, and λ-smoothed using our spam classification we get the following expressions.

3 Logistic regression

Definition 3.1 (Logistic regression)**.** Logistic regression is a type of *discriminative* classifier that learns to map features directly to class probabilities without using bayes rule.

3.1 Logistic sigmoid function

The logistic sigmoid function is a function that maps any real number to the range $[0, 1]$. It is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

Some important properties of this function are:

- symmetry: $\sigma(-x) = 1 - \sigma(x)$
- derivative: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$

Both of these are properties which make analysis easier.

3.2 Logistic regression model

In a basic logistic regression model we seem to try and treat a binary classification problem as a regression problem. All classes are assigned probabilities (e.g. probability of being positive) which constraints them to the range $[0, 1]$.
The decision boundary also has to be constrainted into this range which is where the logistic sigmoid function comes in. Given a hyperplane $\mathbf{w}^T\mathbf{x} + b$ we constrain it using the sigmoid function s follows:

$$c(x) = \sigma(\mathbf{w}^T\mathbf{x} + b)$$

From this for a good choice of $\mathbf{w}$ and b we will get a probability distribution that assigns high probabilities to positive instances and low probabilities to negative instances. To see how well our model is we need a loss function.

3.3 Logarithmic loss

Definition 3.2 (Logarithmic loss)**.** Logarithmic loss (log loss/(binary) cross-entropy loss) is a loss function based on the maximum likelihood estimator.

Similar to the maximum likelihood estimator we some data $\mathbf{x}$, our two parameters $\mathbf{w}$ and b. Where we are looking for a combination that maximumizes the probability of seeing the data we saw. We can call the probability distribution the classifier produces for $\mathbf{x}$, $q_{\mathbf{x}}$. So the probability of a class C conditioned on the data $\mathbf{x}$ is given by:

$$q_{\mathbf{x}}(C) = p(C \mid \mathbf{x})$$

In the case of binary classification for a positive and negative class we have for example

$$q_{\mathbf{x}}(\text{Pos}) = 0.1 \quad q_{\mathbf{x}}(\text{Neg}) = 0.9$$

We can then express the probability of seeing our data D as a function of the probability distribution $q_{\mathbf{x}}$ as follows:

$$p(D) = \prod_{x, C \in D} q_x(C)$$

Since we assume the instances in our data are independent the probabilities of all class labels are just the product of the probabilities of the individual class labels given the data. Now to create an equation for the log loss. Firstly the main idea here is that we want to maximize the log-probability of the class labels given the data. We can express this as:

$$\arg\max_q \prod_{C,\mathbf{x}} q_{\mathbf{x}}(C) = \arg\max_q \log \prod_{C,\mathbf{x}} q_{\mathbf{x}}(C)$$

Since we generally want to minimize the loss we multiply the above expression by -1 to get the following:

$$\arg\max_q \log \prod_{C,\mathbf{x}} q_{\mathbf{x}}(C) = \arg\min_q -\log \prod_{C,\mathbf{x}} q_{\mathbf{x}}(C)$$

We can then move out the multiplication of the logarithm to turn the expression into a sum

$$\arg\min_q \sum_{C,\mathbf{x}} -\log q_{\mathbf{x}}(C)$$

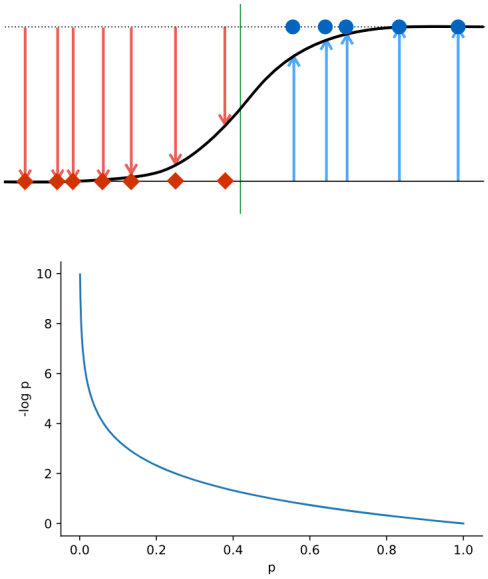
Finally we separate the data into positive and negative instances and express the loss as:

$$\arg\min_q \sum_{x \in \mathbf{x}_{\text{Pos}}} -\log q_x(\text{Pos}) - \sum_{x \in \mathbf{x}_{\text{Neg}}} -\log q_x(\text{Neg})$$

This loss function says that

- For the positive points we want to maximize the log-probability of being positive
- For the negative points we want to maximize the log-probability of being negative

In practice the log loss tries to maximize the sum of the logarithms (minimize the negative logarithm).



In the bottom we see the effect of taking the negative logarithm:

- In the case of a low probability the loss is quite high
- In the case of a high probability, so where the rod in the upper diagram is close to the instance, the corresponding loss is low.

Intuitively this means that for the loss function low probabilities are penalized more than high probabilities, analagously to how with least squares we penalize large errors more than small errors by squaring the residuals between the predicted and actual values.

Minimizing loss

To minimize loss for logistic regression we use *gradient descent* on the loss function. As a quick recap this is our loss function:

$$\text{loss}(\mathbf{w}, b) = -\sum_{x \in \mathbf{x}_{\text{Pos}}} \log q_x(P) - \sum_{x \in \mathbf{x}_{\text{Neg}}} \log(q_x(N))$$

First we can apply the sum rule to yield the following

$$\frac{\partial \text{loss}(\mathbf{w}, b)}{\partial w_i} = \sum_{x \in \mathbf{x}_{\text{Pos}}} -\frac{\partial}{\partial w_i} \log q_x(P) + \sum_{x \in \mathbf{x}_{\text{Neg}}} -\frac{\partial}{\partial w_i} \log q_x(N)$$

First we can look at the positive term. To simplify the derivation we assume $y = \mathbf{w}^T\mathbf{x} + b$.

$$\begin{aligned}
-\frac{\partial}{\partial w_i} \log q_x(P) &= -\frac{\partial}{\partial w_i} \log \sigma(y) \\
&= -\frac{\partial \log \sigma(y)}{\partial \sigma(y)} \times \frac{\partial \sigma(y)}{\partial y} \times \frac{\partial y}{\partial w_i} \quad \text{(chain rule 2x)} \\
&= -\frac{1}{\sigma(y)} \times \sigma(y)(1 - \sigma(y)) \times x_i \quad \text{(basic derivatives)} \\
&= -(1 - \sigma(y))x_i = -q_x(N)x_i \quad \text{(simplification and rewriting)}
\end{aligned}$$

In the context of graident descent this value is the one we want to subtract from the current w_i to better fit the classifier to the particular point x.

As an example we can assume the clasifier does badly, so to a positive point x it assigns a large prob. for the negative class, that is $q_x(N)$ is large. If x_i is a large positive value, then gradient descent will subtract a large negative number $-q_x(N)x_i$ from w_i. This increases the sum $\mathbf{w}^\intercal\mathbf{x}$ and thus the probability of the positive class. The same logic applies in the converse case.

If a classifier does well then $q_x(N)$ is close to 0, and this particular instance has little influence on the gradient descent step.

Deriving the second term we get the same form but with $q_x(P)$ instead of $q_x(N)$ and a minus. This gives us the partial derivative of the loss function with respect to the weights.

$$\frac{\partial}{\partial w_i}\text{loss}(\mathbf{w}, b) = -\sum_{x\in\mathbf{x}_{\text{Pos}}} q_x(N)x_i + \sum_{x\in\mathbf{x}_{\text{Neg}}} q_x(P)x_i$$

3.4 Logistic regression application summary

The summary of how to apply logistc regression is as follows:

1. Use the sigmoid function to turn a linear classifier into a discriminative probalistic classifier
2. Use log loss, so the maximize the log-likelihood of the data given the model
3. Derive the gradient of the loss function and use gradient descent to minimize the loss, that is, search for good weights

One thing of note regarding the desicion boundary is that its still a hyperplane in the 2D case. We just changed the loss function by using it to fit the curved sigmoid function through the probability values in the data. So its kind of like regression but the actual hyperplane $w^\intercal x + b$ is what ends up being the decision boundary.

3.5 Logistic regression overall summary

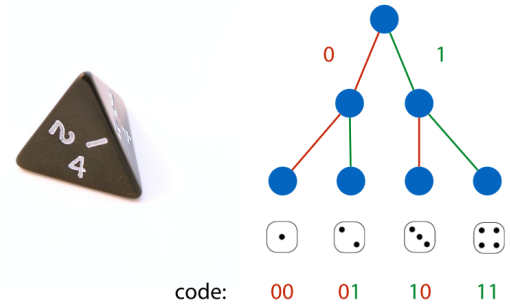
- Use the logistc sigmoid to provide class probabilities from a linear classifier.
- Use $-\log p(\text{class} \mid \text{features})$ as a loss function.
- Points near the decision boundary get more influence that points away
- Log loss generalises naturally to multi-class classification

4 Information theory

4.1 Probabilities and codes

We can simulate things like a 6 sided die via a coinflip. We do this by using different combinations of coinflips to represent the different sides of a die. This in turn then also associates each encoded die roll with a corresponding probability depending on the combination of coinflips.

If we represent these combinations in conjunction with including reset branches we can represent any probability distribution using this encoding method.



Definition 4.1 (Prefix-free tree)**.** A prefix-free tree is a tree which encodes some probability distribution. The resulting codes are prefix-free due to the fact that no code word will be a prefix of any other code word, which comes with the implication that given a prefix-free tree no delimiters are needed to separate the code words.

There is a relation between the length of a code we assign its outcome and its probability. For example the more coinflips we require to get to a particular outcome, the lower the probability that we will get there, and the longer the code.

4.2 Codelengths

If we define $L(x)$ as the length of the code for outcome x we can first see that we need to for example flip a coin 3 times to get to a codeword of length 3, generalizing this:

$$\begin{aligned} p(x) &= \frac{1}{2} \times \cdots \times \frac{1}{2} \\ &= \left(\frac{1}{2}\right)^{L(x)} \\ &= 2^{-L(x)} \\ \implies L(x) &= -\log p(x) \end{aligned}$$

The key takeaway here is that the negative logarithm of the probability of an outcome is the length of the code for that outcome.

For any distribution L we can find a prefix-free code so that the value $-\log p(x)$ and the code-length $L(x)$ differ by no more than 1 bit for any outcome x. Formally this is expressed as

$$|-\log_2 p(x) - L(x)| \leqslant 1$$

4.3 entropy

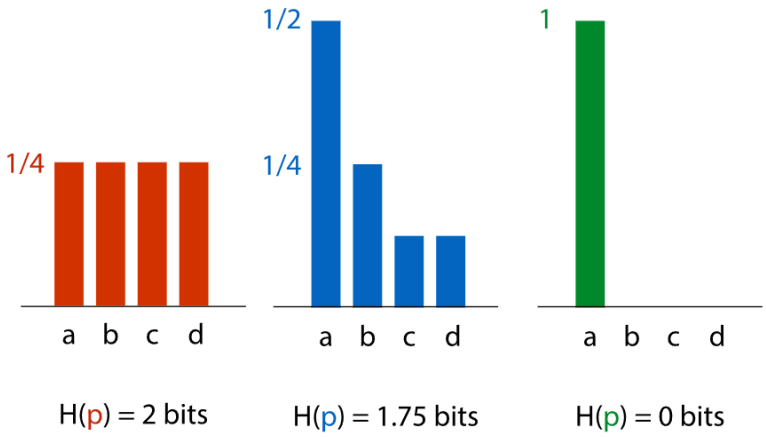
Definition 4.2 (Entropy)**.** Entropy is the measure of the expected number of bits we will have to use per outcome. Its the codelength of each outcome multiplied by its probability, summed over all outcomes.

Formally, if we encode X with the corresponding code p, our expected codelength is given by:

$$\begin{aligned} H(p) &= E_p[L(x)] \\ &= \sum_{x \in X} p(x) L(x) \\ &= -\sum_{x \in X} p(x) \log p(x) \end{aligned}$$

Uncertainty

Entropy is a measure of uncertainty about the outcome of a distribution. In other words it measures how uniformly spread out the probability mass is among the outcomes.



4.4 Cross-entropy

Definition 4.3 (Cross-entropy)**.** Cross-entropy is the expected codelength if we use q, but then the data comes from $p(x)$. So instead of using a code that corresponds to our data p to encode it we use some other code based on a distribution $q(x)$. Formally this is expressed as:

$$\begin{aligned} H(p, q) &= E_p[L_q(x)] \\ &= -\sum_{x \in X} p(x) \log q(x) \end{aligned}$$

Some notes regarding cross-entropy:

- The code corresponding to p provides the best expected codelength out of all possible prefix-free codes.
- The cross-entropy is a good way to quantify the distance between two distributions.

Kulback-Leibler divergence

Definition 4.4 (KL divergence)**.** This is a function which is a measure of how far apart two distributions are. It's not symmetric and is defined as the difference between the cross-entropy and the entropy. Formally

this is expressed as:

$$\begin{aligned} KL(p, q) &= H(p, q) - H(p) \\ &= -\sum_{x \in X} p(x) \log \frac{q(x)}{p(x)} \end{aligned}$$

4.5 Generalizing to continous and discrete space

We can create a general expression for entropy and KL divergence using the expected operator which implies we integrate for continous spaces and sum for discrete spaces. Formally this is expressed as:

$$H(p) = -\int p(x) \log p(x) dx = -E_p[\log p(x)]$$

$$KL(p, q) = -\int p(x) \log \frac{q(x)}{p(x)} dx = -E_p[\log \frac{q(x)}{p(x)}]$$

4.6 Log loss as cross-entropy loss

We can rewrite the log-loss function using the function for cross entropy loss as follows

$$\begin{aligned} \text{loss}(q) &= \sum_{x \in X} H(p_x, q_x) \\ &= \sum_{x \in X} p_x(P) \log q_x(P) + p_x(N) \log q_x(N) \\ &= -\sum_{x \in X} \log q_x(P) - \sum_{x \in X} \log q_x(N) \end{aligned}$$

Some interpretations we can make about this relation:

- If we minimize $-q_x(P)$ in logistic regression, we are also minizing the amount of bits we would need to transmit to comunicate that x is of the positive class.

4.7 Minimum Description Length principle (MDL)

Compression is similar to learning:

1. Look at some data
2. Try to isolate recorruding patterns in the data, using coding and entropy

Sender and receiver framework

The SR framework can be described as follows:

1. Sender sees some data
2. Sender and receiver can come up with any scheme to commmunicate the data
3. Data has to be sent using the scheme
4. Data must be perfectly reconstructed/decoded by the receiver

Two part coding

The two part coding scheme can be described as follows:

1. Tje semder amd receiver agree beforehand on a family of models (e.g. normal distribution, indexed by parameters μ and σ)
2. Once the sender has seen the data, they send the model best suited for the data (e.g. by sending the parameters μ and σ for the chosen model)
3. The sender uses the model to encode the data and sends it over
4. The receiver uses the model to decode the data

The best choice of model is the one that minimizes the cost of communication the model and the data given the model.