1. Mean squared error

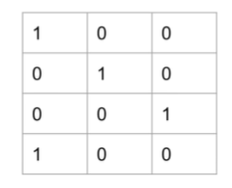
* Understand MSEE from a probabilistic perspective
* Helps prepare us for the cross-entropy loss
* Why is it squared?
* We want the errors to be positive – if they are not always positive, they cancel out
* Why not absolute value? In fact, this is perfectly fine!
* But not usually what we are talking about when we talk about linear regression
* **Maximum likelihood estimation (MLE)**
* E.g. we model the heights of the students in our class as a Gaussian distribution (bell curve)
* Prob density function
* The best estimation for
  + Data points collected:
* First step is to establish the likelihood function
* Note: x’s are constants (data points collected), variable is
* Maximize L with respect to
* Calculus
* Normally, we would set -> hard
* Instead, we first take the log
* It works because the log is a *monotonically increasing* function
* The derivative
* Solve for after setting the derivative to 0
* We get back our original answer
* Notice that these constants do not affect the final answer
* Why not set
* This is what we are equivalently trying to **maximize:**
* Maximizing the likelihood is the same as minimizing the squared error (where the error is the difference between
* Back to regression
* For regression (y is the random variable, is the mean of a Gaussian distribution
* An equivalent perspective: when we use the squared error, we are making the assumption that the error of our model w.r.t the data is Gaussian-distributed with mean 0

1. Binary cross-entropy

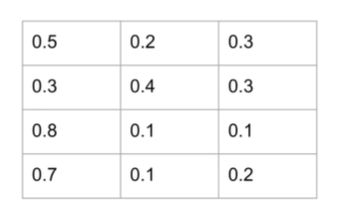
* Correct loss function to use for binary classification
* Probabilistic perspective
* The outcomes are binary
* Distribution describing binary events -> Bernoulli
* Problem set up
* Data collected (binary):
* Prob Mass Function of Bernoulli (we use PMF for discrete random variables, PDF for continuous random variables)
* Likelihood
* Log likelihood
* Set the derivative to zero
* **Negative log-likelihood**
* Binary classification (y is the random variable, is the probability of 1)
* Dividing by N
* This makes our loss invariant to the number of samples (N)
* If we have lots of samples, the sum of errors will be large only due to the large number of samples
* Dividing by N makes the value more meaningful
* Summary
* For both classification and regression, our loss function is grounded in prob
* Regression: MSE, Gaussian distribution
* Binary classification: binary cross-entropy, Bernoulli distribution
* Common pattern: in both cases, the error is just the **negative log-likelihood**
* Therefore, the solution (for the params) is the **maximum likelihood** solution

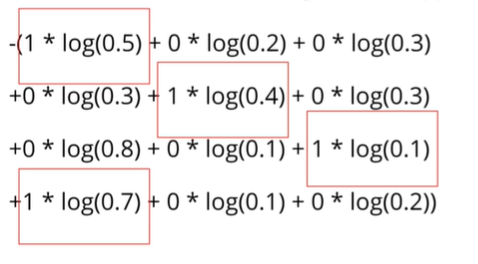
1. Categorical cross entropy

* Categorical cross-entropy is the negative log-likelihood of some distribution
* For binary outcomes, the distribution is Bernoulli
* Multiple categorical outcomes?
* Categorical distribution
* E.g. Die roll
* 6 possible outcomes (more are possible)
* Categorical PMF
* The one function is the indicator function -> returns 1 if its argument is true, 0 otherwise
* Easy to confirm that
* Problem setup
* Data collected
* Likelihood
* Log-likelihood
* Categorical cross-entropy
* No more indicator function (although we could use instead)
* is a one-hot encoded value
* Class label can only have one value for each sample
* Only one of the K values can be 1
* One-hot encoding is inefficient
* Suppose we have N = 4 samples and K = 3 classes (cat, dog, bird)
* The targets (y) are: {cat, dog, bird, cat}
* Numerically: [1,2,3,1]
* One-hot encoding is



* Suppose my prediction probabilities are





* Only take the log of highlighted values and add them together
* Can I do that without one-hot encoding?
* Numpy double indexing (no one-hot encoding)
* We want
* Same as:
* In general:
* Note: in code, we start indexing at 0
* Sparse Categorical cross-entropy
* The regular categorical cross-entropy uses the full N x K target
* Requires N x K multiplications / additions
* The sparse categorical cross-entropy uses the original target 1D array
* Requires only N multiplications / additions
* More efficient to use this