WEEK 11, MODULE 2

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

QMIND EDUCATE

KAELAN LUPTON

TABLE OF CONTENTS

WEEK 11, MODULE 2: MACHINE LEARNING: NAÏVE BAYES

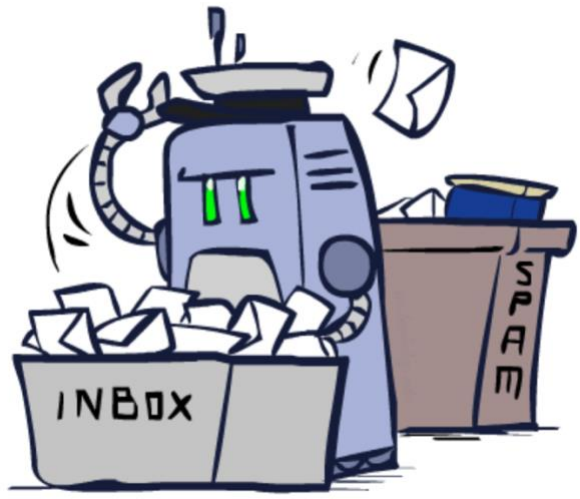
MACHINE LEARNING

To this point, we have spent significant time learning how to use a model to make optimal decisions. After experimenting with searches, Hidden Markov Models, and Bayes Nets, it's become apparent that the difference between an effective or ineffective performance by an agent is judged by how good of a model is used. However, it can be expensive (financially, temporally, and computationally) to come up with these good models ourselves. The next step we will take is learning to automate the process by having a computer do the "heavy lifting" by studying data or experience.

Some methods of going about this are…

- Learning the parameters involved (e.g. probabilities)
- Learning the structures involved (e.g. BN graphs, neural networks)
- Learning hidden concepts (e.g. clustering – an example of which is online news being clustered into subgenres, or classifying people into groups and targeting them with advertisements)

In this module, we'll be focusing on **model-based classification using Naïve Bayes.** Worry not if that sounds like a bunch of jargon – we'll clarify it all!



CLASSIFICATION EXAMPLES

Classification in machine learning involves *sorting a large data set into various groups to achieve a goal.* The most common example of a *classification* problem in day-to-day life is the spam filter on your e-mail. This model judges every incoming e-mail and determines whether it is spam. To break it down:

Input: E-mail

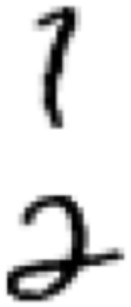
Output: One of two options: **spam** or **safe**. *This could be well modelled by a binary system – 0 for safe, 1 for spam.*

Process: The setup to building a *spam filter* involves using a large collection of example emails, each labelled *spam* or *safe*, for our model to *train itself on* (we will discuss more about training down the line). It is important to note that a person somewhere had to hand-label these e-mails to create a data set that the model can use to train itself! The model will study this set of data and learn to predict the labels (spam/safe) of new future emails. The system will judge each e-mail on features like specific words often sound in spam ("Free!", "Prince", etc.), common text patterns ($, %, alternating capital letters, etc.), or non-text qualities (whether the sender is in your contacts list, etc.). *Often, your machine trying to gauge which of these features are valuable is the one of the toughest parts of machine learning!*

Another example of classification is *automated digit recognition*. A true game-changer for the machine learning community was the release of the **MNIST Database of handwritten digits** in the late 1990's for machines to train on to learn to recognize hand-written numeric values. Digit recognition classifiers can read a hand-written digit and tell us what the number is – or at least, what the best guess is. *If you're curious about the dataset, it is available for free online and to be experimented with in your own machine learning experiments, look it up!* For now, let's dive a little deeper into the specifics:

Input: an image (a grid of pixels) of a hand-written digit.

Output: an estimated matching digit 0-9.



Process: Provide your model with a large data collection of handwritten digits in image form. Each image you plan on using to train your model must be labelled with actual correct number, so the model can learn about the accuracy of its estimates and make changes accordingly. Features the model might look for are components of the number, bright or dark spots on the pixel grid, or loops (found in digits such as 8).

We will be returning to these examples repeatedly throughout this module, so make sure you understand them!

There should be a pretty obvious commonality here: large data sets are almost ubiquitously required for machine learning and model-based classification. *You must have enough pre-labelled data to train a model on for it to learn properly how to classify any inputs.*

Other interesting examples of classification problems are found in medicine (inputs of symptoms or abnormalities, outputs of diseases or tumors), fraud detection (inputs of account activity or history, output of binary fraud estimate), and machine vision (inputs of visual or image data, outputs of image contents such as face detection).

Model-Based Classification

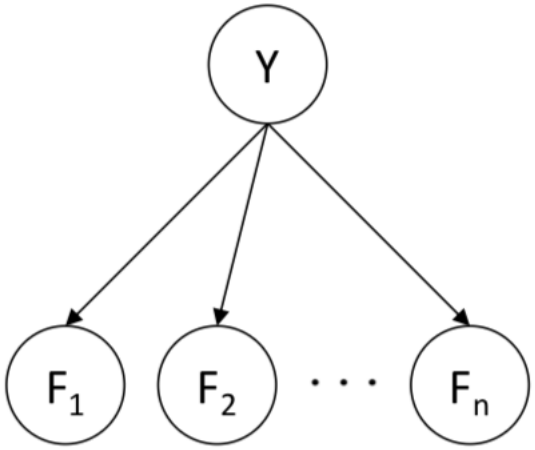
One way of *building a classifier* is by using a process called **model-based classification**, which is pretty much exactly what it sounds like; the goal of *model-based classification* is to build a function that maps observed features to possible outputs which can generalize well to new or unseen data. Each feature has a domain of some random variable, as does each output.

Let's break this definition down a little further. Start with the notion that you have something you're trying to predict, be it whether an e-mail is spam or safe, or what a handwritten digit is in a PNG file. We will be using a model to make this prediction; in this module, we will use a *Bayes Net* (specifically a *Naïve Bayes*, called Naïve because we're assuming *conditional independence* – which will be elaborated on soon). This Bayes Net will have features associated with it which represent the random variables, and a feature for the label or decision you are trying to predict (i.e. spam/safe or digit). To begin, we instantiate (set) any of the variables we have observed via inputting a sample. From there, *we can calculate a probability distribution conditioned on the features given.*

Naïve Bayes: Digit Example

Return to our digit example. We will focus on the features of a 10x10 PNG image of a digit, which we will structure simply: each feature will be a pixel, giving us 10x10=100 features. A pixel will have a binary value of 0 if it is empty, or 1 if it is full. You could get into greater specifics of the level of fill each pixel has, but in the interest of simplicity we will stick with a binary evaluation. *We discussed features such as loops in an image above, but we will only be looking at the values of each individual pixel with this example.*

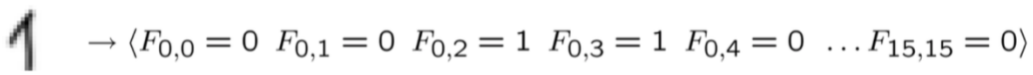
When using *Naïve Bayes*, we will always draw the *simplest* Bayes Net that could possibly work – one feature Y (our class label or category feature, with possible values y_i) with an arc to each observed feature F_i independently, as such:



Time for a crucial concept of Naïve Bayes: *each feature is assumed to be not necessarily independent (not affecting each other at all), but rather conditionally independent given the label Y.*

My favourite wording of this concept is as follows: "Imagine that the underlying variable Y gives rise to each feature independently." We are imagining the features don't affect each other whatsoever – so, the value of one pixel will not affect the value of a pixel beside it. In practice, we know this not to be true – a brush stroke works continuously, so the likelihood of a pixel being filled is greater if a neighbouring pixel is also filled. However, we will continue to assume conditional independence given Y, and it will become apparent that this method often works well regardless. *For a review of conditional independence, see Week 9, Module 1.*

Moving on, once given our input (PNG image), we must do some computation/examination to receive a vector or equation that characterizes our image, showing us all the features. An example for a digit would be:



($F_{0,0}$ is the pixel found at the grid position 0, 0 of the PNG file and is set to 0 (off), etc.)

We can now safely throw away our image, as this vector clarifies precisely what is going on in the image.

As an aside, if this were an equation for our spam/safe filter, a very simple F value could be based on the presence of a word, with 0 or 1 indicating its presence.

This is our Naïve Bayes model for the digit example:

$$P(Y|F_{0,0}\dots F_{15,15}) \propto P(Y)\prod_{i,j} P(F_{i,j}|Y)$$

Reads as: The conditional distribution of Y (or, the probability of Y given the occurrence of feature values $F_{0,0}$ through $F_{15,15}$) is proportional to your joint probability (the probability of Y multiplied by the probability of each feature given the class). We will better numerically represent this later.

Note: recall that Y represents our entire set of possible values of class labels. Each possible value of Y can be represented by y, specifically in lower case.

General Naïve Bayes

Naïve Bayes Notes

Generally, we need to learn a probability distribution. Here is the above formula in more general terms.

$$P(\mathsf{Y}, \mathsf{F}_1 \dots \mathsf{F}_n) = \quad P(\mathsf{Y})\prod_{i} P(\mathsf{F}_i|\mathsf{Y})$$

- How many parameters are on the right side of the equation? If *n* is the number of features, F is the domain size of each feature (for digits, domain is two: on or off), and Y is the number of class labels (for digits, there are 10), **there will be *n**F*Y parameters.** You can look at this as F*Y being the size of one table of joint probabilities, and there being one for each feature *n*.

- As a causal model, this assumption of conditional independence makes no sense – variables are going to affect other variables. Consider the digit example – brushstrokes work continuously such that if one pixel is set to "on", the pixels around it will have a greater probability of being set to on as well. However, this conditionally independent model does a great job with approximation, *and has the enormous bonus of being compact.*

- The **left side** of the equation above **does not** assume conditional independence, and clearly has the potential to increase exponentially – which isn't computationally or temporally feasible when dealing with large numbers. The **right side** is far simpler as it **assumes** conditional independence – we can simply evaluate the effect of each variable independently given the class. It is a distribution over the number of features F_i in our Bayes Net. This equation only increases linearly, which is more manageable than an exponential increase.

- Fun fact: the *right side* also boasts another kind of efficiency. Every time you have an experience, there is only so much you can learn from that experience statistically. So, if you're attempting to learn something over exponentially numbers, you can have exponentially many experiences, which results in an extraordinary amount of data. It's more cost-effective to learn things from less data by only having to specify how each feature depends on the class, not how each feature depends on each other. This is called **statistical efficiency**, and will come up a lot if you ever spend time studying data science.

- Another example: It will be easier to determine if an e-mail is spam by only checking for the word free than it will be if you're checking for the interactions between each word. Make sense? Tired of hearing about this point yet?

To use a Naïve Bayes approach, we need to…

- Run inference.
- Estimate all local conditional probabilities.

In the next two headings, we will cover each of these steps.

Inference

How do we run **inference** in a *Naïve Bayes* model? How do we actually predict which class label is correct?

Once we have the features, compute the *joint probability* of the label and the evidence for each label.

$$P(Y, f_1 \dots f_n) = \begin{bmatrix} P(y_1, f_1 \dots f_n) \\ P(y_2, f_1 \dots f_n) \\ \vdots \\ P(y_k, f_1 \dots f_n) \end{bmatrix} \Rightarrow \frac{\begin{bmatrix} P(y_1) \prod_i P(f_i|y_1) \\ P(y_2) \prod_i P(f_i|y_2) \\ \vdots \\ P(y_k) \prod_i P(f_i|y_k) \end{bmatrix}}{P(f_1 \dots f_n)}$$

There is one line in the center vector for each possible label. This is done by multiplying the **prior $P(y_i)$** by the **probability of each of feature given the class label $P(f_i|y_j)$**, which is seen to the right of the blue arrow. Whichever class label has the *highest score will be the predicted label*.

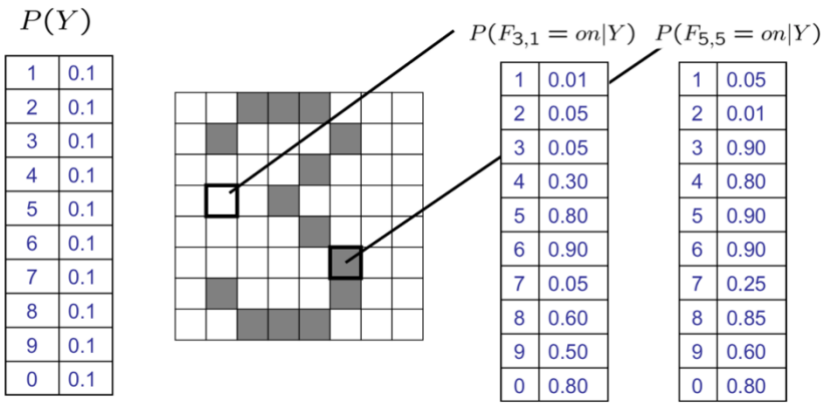
The prior is a probability assigned to each class label y_i in Y showing the probability of its occurrence without any other data. The sum of each prior probability together must equal 1, as it represents every possible class label that could be instantiated.

Additionally, if you sum each entry in the joint probability of the label and evidence then divide by the probability of the predicted label, you can get a *probability that you have selected the right label*.

Local Conditional Probability Tables

The last thing you will need to run Naïve Bayes is the *local conditional probabilities associated with the problem.* This means finding the *priors* (defined above) for each class label and the conditional *probabilities* for each feature F_i given each class label $y_i \in Y$. Essentially – finding all the probabilities of the problem we have generally been given so far. These probabilities are called the **parameters** of the model, **which we will henceforth refer to as θ**. We often find θ by looking at training data – something we will discuss with *machine learning* later.

The image below shows an example of possible local conditional probabilities for our digit problem. The leftmost table, labelled P(Y), is the set of *priors* for the model. Here it is assumed each number is equally likely to be written for simplicity.



1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
0	0.1

1	0.01
2	0.05
3	0.05
4	0.30
5	0.80
6	0.90
7	0.05
8	0.60
9	0.50
0	0.80

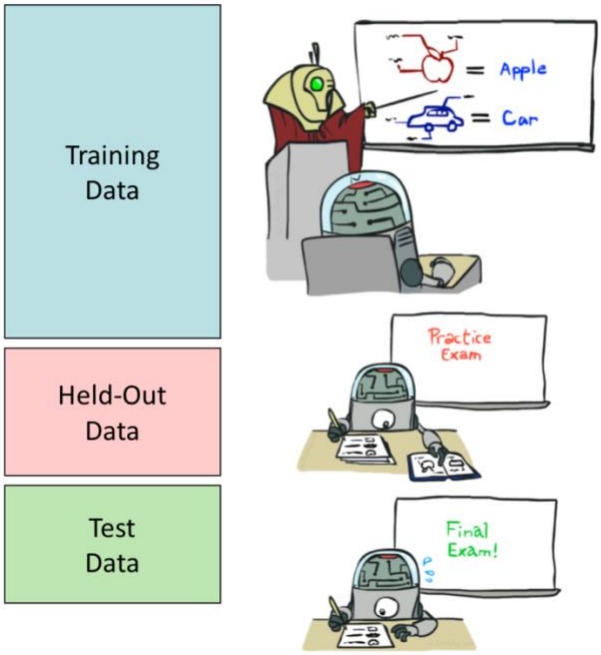
1	0.05
2	0.01
3	0.90
4	0.80
5	0.90
6	0.90
7	0.25
8	0.85
9	0.60
0	0.80

The graphed numeric 3 shows a 15 x 15 grid, representing a PNG image with various pixels turned on and off. The table labelled $P(F_{3,1} = on|Y)$ shows the probability of pixel 3,1 being turned on given each possible class label, while the $P(F_{5,5} = on|Y)$ shows the same for pixel 5,5. Pixel 3,1 appears to almost never be on for the number 1, but almost always on for the number 6. Once a table like this is made for each pixel square (in combination with knowing the priors) you have found the parameters θ for the model.

Finding the parameters for a text-based problem is a little different. Each feature W_i is the word at position *i.* We can predict a class label for a selection of text (like an e-mail) by conditioning the labels on the feature values – each word. A method often used with text is called the **Bag-of-words Naïve Bayes**, which assumes that each word W_i is *identically distributed* – as though you were just reaching into a bag of words and treating each word the same. This allows us to **not** have to learn a separate conditional probability table for each word position in an e-mail, and instead assume that word position doesn't *really* play a role. This is a dangerous assumption, but works well enough for a simple model.

Key Concepts in Machine Learning

Congratulations! We're *finally* getting into the good stuff – the processes and methods that allow machine learning to actually, you know, learn.



To this point, we've assumed most of our data, such as parameter values or probabilities, will be provided. In practice, however, we're often simply given a set of data and tasked with making some sense of it. This data is often a series of labeled instances, like the MNIST Data Set or emails marked spam/safe, which is to be split into three sets:

- Training set (Often about 70% of the total set)
- Held-Out Data (Often about 15%)
- Testing Data (Often about 15%).

The idea is that you will have your model *study* on your training data, and validate that studying on the held-out data, then test your model on the testing data.

The training data is where the bulk of your work is done, attempting to set the parameters of your machine learning algorithm – in our case, our *Bayes Net*. A method of *studying training data* could be looking at the **relative frequency of the occurrences of features** (we will look at other methods such as *perceptrons* later in the course). Once you've completed studying training data, you can *try* the held-out data to see how well the model works, like a practice exam. This step is often used best for selecting which model works best, if you have built multiple models and are judging which to move forward to the testing portion. *If your model(s) doesn't perform well, perhaps you return to your training data and add more features, find new features, or eliminate features. You can then retrain your model(s) and try the held-out data again.*

Once training and validating is completed, you can test your model using the testing data to see how well it performs, like a final exam. Computing the **accuracy** of your model (how many instances were predicted correctly divided by total instances) on the test data will give you a decent idea of how well your model performs.

Of note: it is very important to ***not expose your model to the test data until it is time to test.*** *This guarantees the model is performing on labelled data it's not seen before to gauge its predictive prowess.*

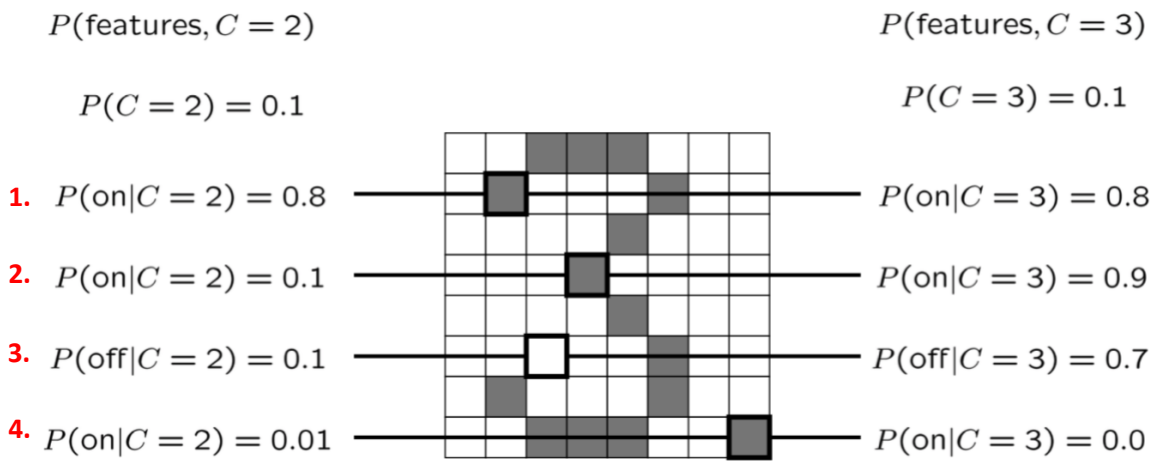
These three steps can often be reduced to the simple titles **Training, Validation, and Testing.**

Data is split up this way as a means of *seeking generalization and avoiding overfitting*; we've touched on overfitting in the past in reinforcement learning.

Overfitting and Generalization

Overfitting occurs when an algorithm prepares itself *too well* on the given training data, such that it begins matching features directly to what it has seen before, rather than making *actual predictions*. An *overfit model* will perform exceptionally on training data, but fail to translate that success to unfamiliar testing data.

Overfitting with our digit example would look something like this.



Here we see an analysis of a digit based on four different boxes. The left side is checking for the number two, while the right checks for number 3. The number is very clearly a three, though there is a small dot in the corner.

The priors for both are 0.1; it is equally likely it could be a two or a three before evaluating. If we stopped evaluating after lines 1, 2, and 3, our model would multiply the probabilities on the 2 side (0.8*0.1*0.1) for a probability of 0.008, and the probabilities on the 3 side (0.8*0.9*0.7) for a probability of 0.504 and determine the digit to be a 3.

However, once line 4 is considered, the model sees that nowhere in its training data has pixel (15,15) been coloured in for a 3, so the probability of such is set to 0. Therefore, the model determines the digit *must* be a 2.

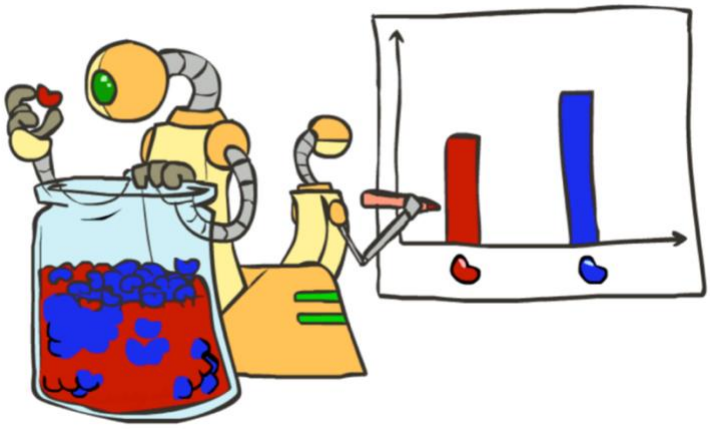
Some notes…

- As we see here, *relative frequency parameters often overfit*. As seen in our example, just because none of the training data showed pixel (15,15) coloured in for a 3, doesn't mean we won't see pixel (15,15) coloured in for a 3 in the testing data.
- There should almost *never* be a case we are assigning a zero probability to unseen events. Multiplying any probability by zero will just set that number to 0 moving forward, no matter what other evidence appears – this could *mean you miss out on accurate predictions*.
- To generalize better across all possibilities, we will need to **smooth or regularize** our estimates.

Parameter Estimation

What is **parameter estimation**?

We're attempting to estimate the distribution of a random variable to move away from setting unseen events to a zero probability – how can we do this?



- **Elicitation**: asking a human expert for input is always an option, but rarely will a someone know off the top of their head what probabilities to input where.
- **Empirically**: use training data (learning – this is what we've done so far). For each outcome *x*, we can look at the *empirical rate* of that value found in a sample. Say we pull three jelly beans from a bag of red and blue jelly beans. Two beans are red; one bean is blue. We can look at that sample, and say that the **maximum likelihood estimate** from the data in that sample is that red has 2/3 probability of being selected. The maximum likelihood estimate is proportional to the counts of occurrences – the number of times a variable *x* occurs divided by the number of total samples.

Below on the **left** is how we will write *empirical rates* formulaically, and to the **right** is another equation that can also be used to find the estimate that *maximizes the likelihood of the data*.

$$P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}} \qquad L(x, \theta) = \prod_i P_\theta(x_i)$$

L is the likelihood of the data x and parameters ϑ. This is equal to the product of each of the probabilities P_ϑ of each of the data points x_i. This equation would essentially break down to finding the value of theta that produces the largest (maximum) likelihood value, which can be done by taking derivatives on the right side of the formula.

It is worth noting that a sample of 3 is incredibly small, and so this estimate is likely wrong. It will improve with increased sample size.

Maximum likelihood estimates still don't directly give us an answer to the problem of dealing with unseen events. We will need to add something to the formula to handle this.

Maximum A Posteriori estimate: *a maximum a posteriori probability estimates factors in the consideration you may pre-emptively expect some outcomes to be more likely than others. This involves multiplying in another probability to find the ϑ value most likely conditioned on the data*.

SMOOTHING

Smoothing is a great way to avoid having your model evaluate unseen events to a zero probability.

Laplace Smoothing: a method of making sure you have seen every possible outcome at least once, so to avoid setting unseen events to a probability of zero. This is done by simply pretending you have seen every outcome once more than you have, and adding those "observations" to your total. The equation

$$P_{LAP}(x) = \frac{c(x)+1}{\sum_x [c(x)+1]}$$

$$= \frac{c(x)+1}{N+|X|}$$

for Laplace Smoothing is as follows:

So, returning to the example above, if we had two red jelly beans and one blue jelly bean, $P_{LAP}(x)$, where x = red, is now equal to 3/5.

The numerator of the above equation shows the count of variable x + 1, while the denominator shows the sum of all counts of variables x_i plus one. This gives a new relative frequency.

We can take this a step further by *customizing* the number of times we pretend to have seen every outcome, the number of which is denoted by *k*. The adjusted equation looks like:

$$P_{LAP,k}(x) = \frac{c(x)+k}{N+k|X|}$$

Variable *k* is called the **strength** of the prior. It is a variable we can tweak from model to model to tweak our findings. In some situations, it may make sense to have a lower or higher k value – it is up to the human to decide. *Note: Laplace smoothing with a k value of 0 would just be a maximum likelihood equation!*

TUNING

We've nearly come full circle on using Naïve Bayes to solve classification problems. Recall the three groups we separated our data in to: training, held-out, and test data. *Held-out data* was declared to be good for selecting a model from a group of models, but it also plays a large role in tuning your **hyperparameters** such as amount of smoothing to do or *k* value adjustment. This now gives us two types of unknowns: *hyperparameters* and *parameters* (the probabilities P(X|Y) and P(Y)).

There will be a sweet spot when adjusting your *k* value where your classifier performs the best. Too high a *k* value and the model will stay *very generalized*, not even performing well on test data. Too low and the model will begin to *overfit*. Tune your *k* value on your held-out data to figure out what number is best.

As an Aside…

Here is a link to a website that sheds light on how to use Python libraries to build a simple Naïve Bayes classifier: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

References

Artificial Intelligence: A Modern Approach, Global Edition. (2018). 3rd ed. Stuart Russell and Peter Norvig.

Ai.berkeley.edu. (2015*). Berkeley AI Materials*. [online] Available at: http://ai.berkeley.edu/course_schedule.html [Accessed 25 Mar. 2018]

Yann.lecun.com. (2018). *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. [online] Available at: http://yann.lecun.com/exdb/mnist/ [Accessed 26 May 2018].