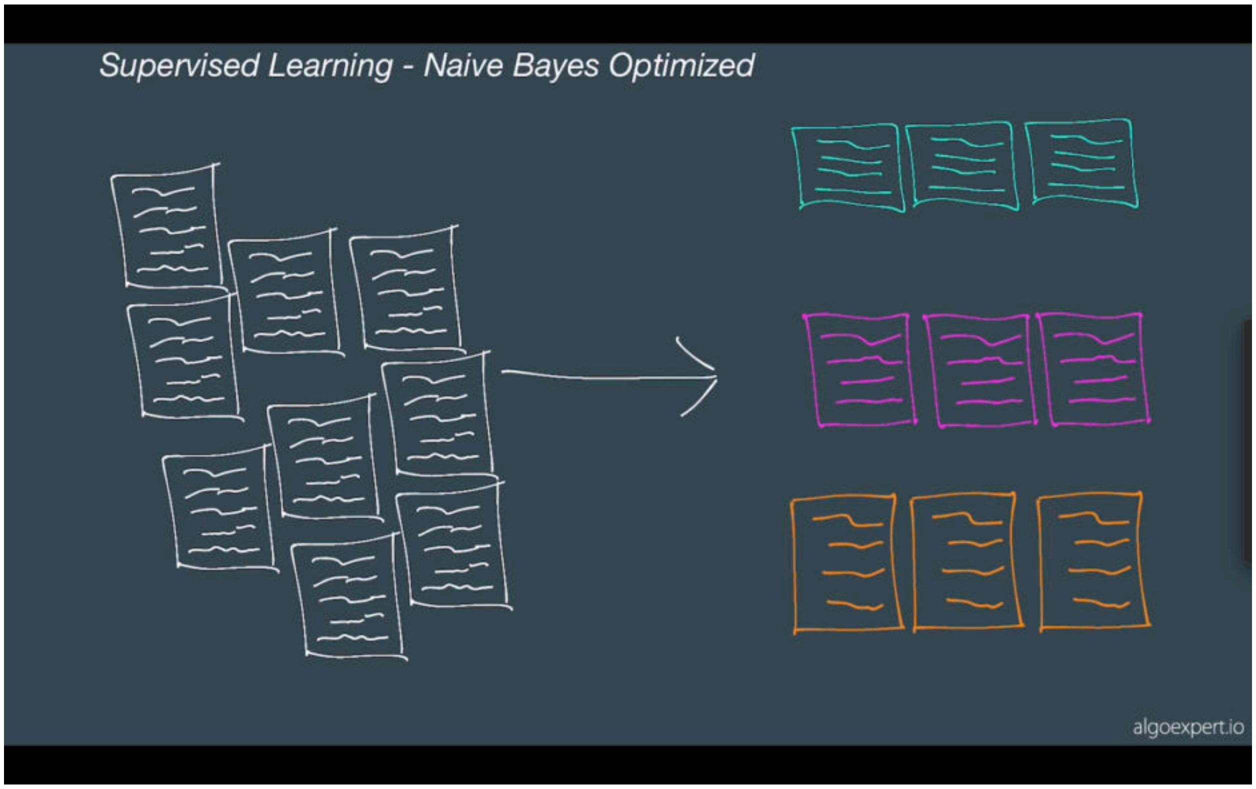
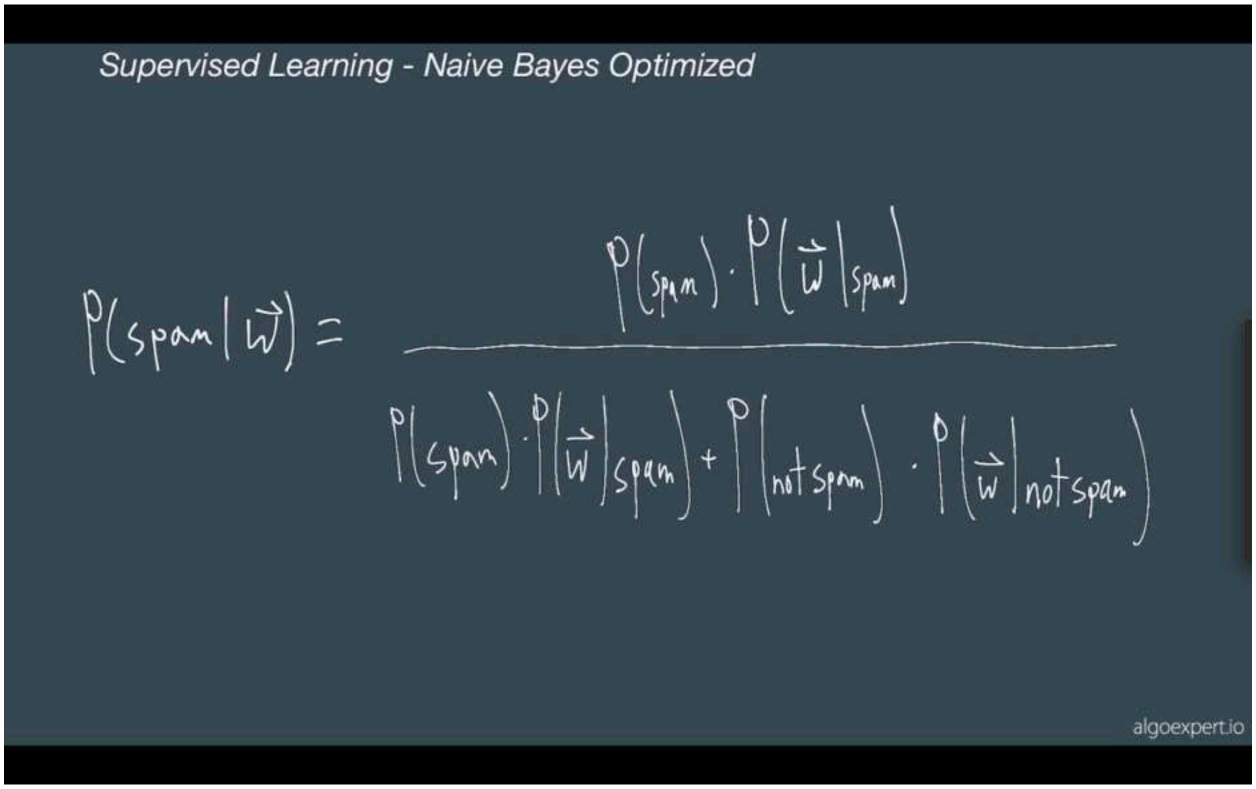
Naive Bayes Optimized

Welcome back to ML Experts Machine Learning Crash course. In this session, we're gonna build on our understanding of the Naive Bayes model, as well as expand our definition of supervised learning. So far, we've only been able to classify between two classes, either a spam or not spam, but what if there's more classes?



For instance, let's say we work for the social journaling company medium, and they have a bunch of content creators that create articles, what if we want to automatically classify articles that they create into classes or tags such as finance, politics or lifestyles slash travel blogs automatically without them having to do anything or without us having to hire people to read the articles and classify them for us.

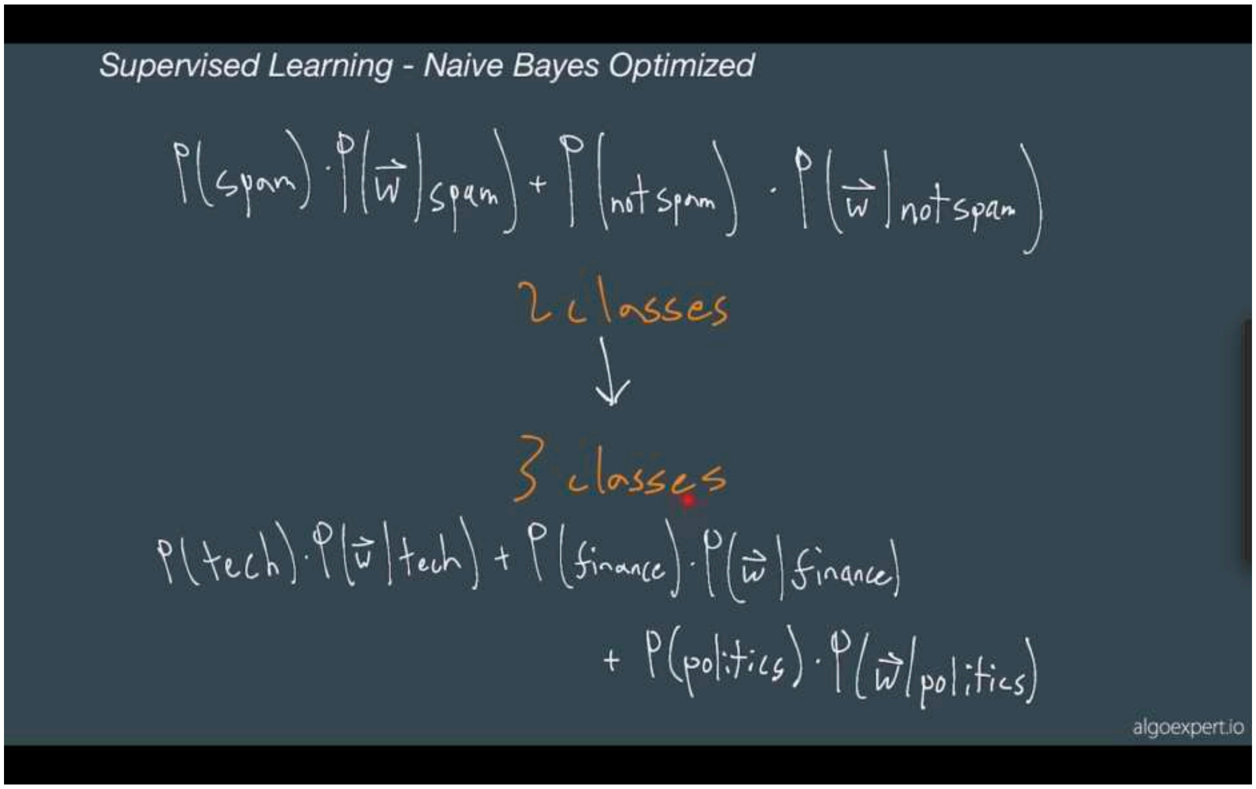


Supervised Learning - Naive Bayes Optimized

$$P(\text{spam} \mid \vec{W}) = \frac{P(\text{spam}) \cdot P(\vec{W} \mid \text{spam})}{P(\text{spam}) \cdot P(\vec{W} \mid \text{spam}) + P(\text{not spam}) \cdot P(\vec{W} \mid \text{not spam})}$$

Well, we can actually use Naive Bayes for this as well. Let's revisit the old spam model that we used. In the denominator, you can see that we actually only have two terms, right? We have one term for each class to evaluate it as if it were spam and as if it were not spam. And if you need to refresh, just go ahead and look at the previous Naive Bayes video.

We need to update this model to incorporate more than just these two classes. So somewhat obviously, right? We can just go from two classes to three classes by incorporating three terms instead of two. So we'd have a term for tech, a term for finance and a term for politics.



Supervised Learning - Naive Bayes Optimized

$$P(\text{spam}) \cdot P(\vec{w} \mid \text{spam}) + P(\text{not spam}) \cdot P(\vec{w} \mid \text{not spam})$$

2 classes

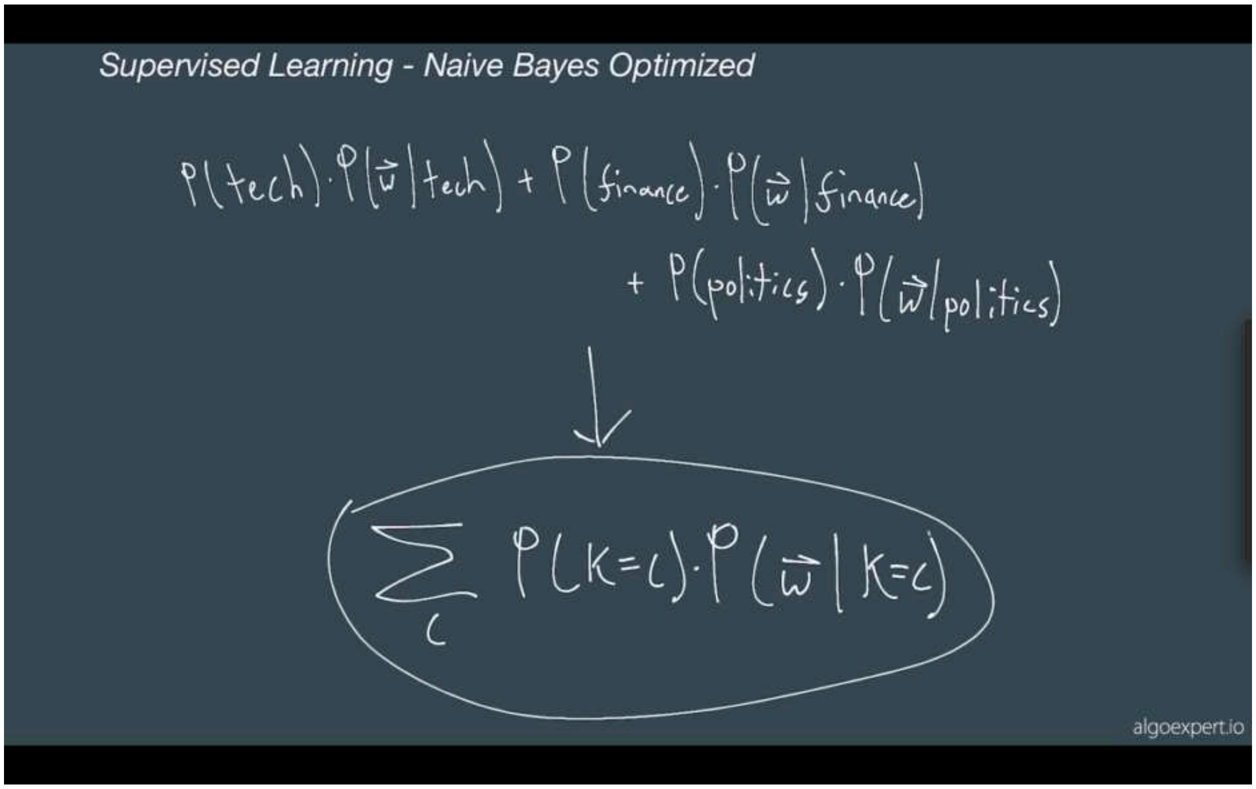
$$\downarrow$$

3 classes

$$P(\text{tech}) \cdot P(\vec{w} \mid \text{tech}) + P(\text{finance}) \cdot P(\vec{w} \mid \text{finance})$$
$$+ P(\text{politics}) \cdot P(\vec{w} \mid \text{politics})$$

algoexpert.io

We can actually generalize these summations here with the symbol so that we would actually just sum through every single class C. So if we had five classes, we could just write this and see would be five classes. All right, so now we can actually rewrite a condensed form of our Naive Bayes model with this summation here in the denominator.



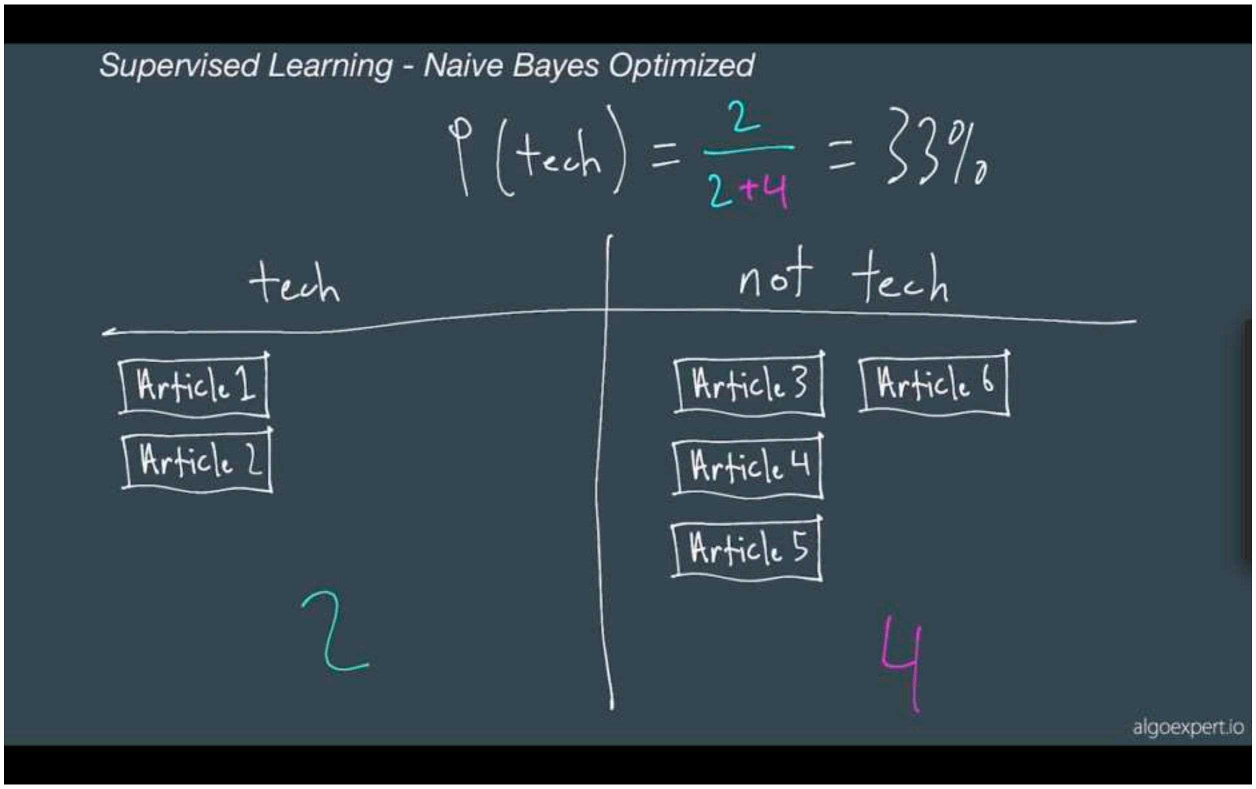
Supervised Learning - Naive Bayes Optimized

$$P(tech) \cdot P(\vec{w} \mid tech) + P(finance) \cdot P(\vec{w} \mid finance) + P(politics) \cdot P(\vec{w} \mid politics)$$

$$\downarrow$$

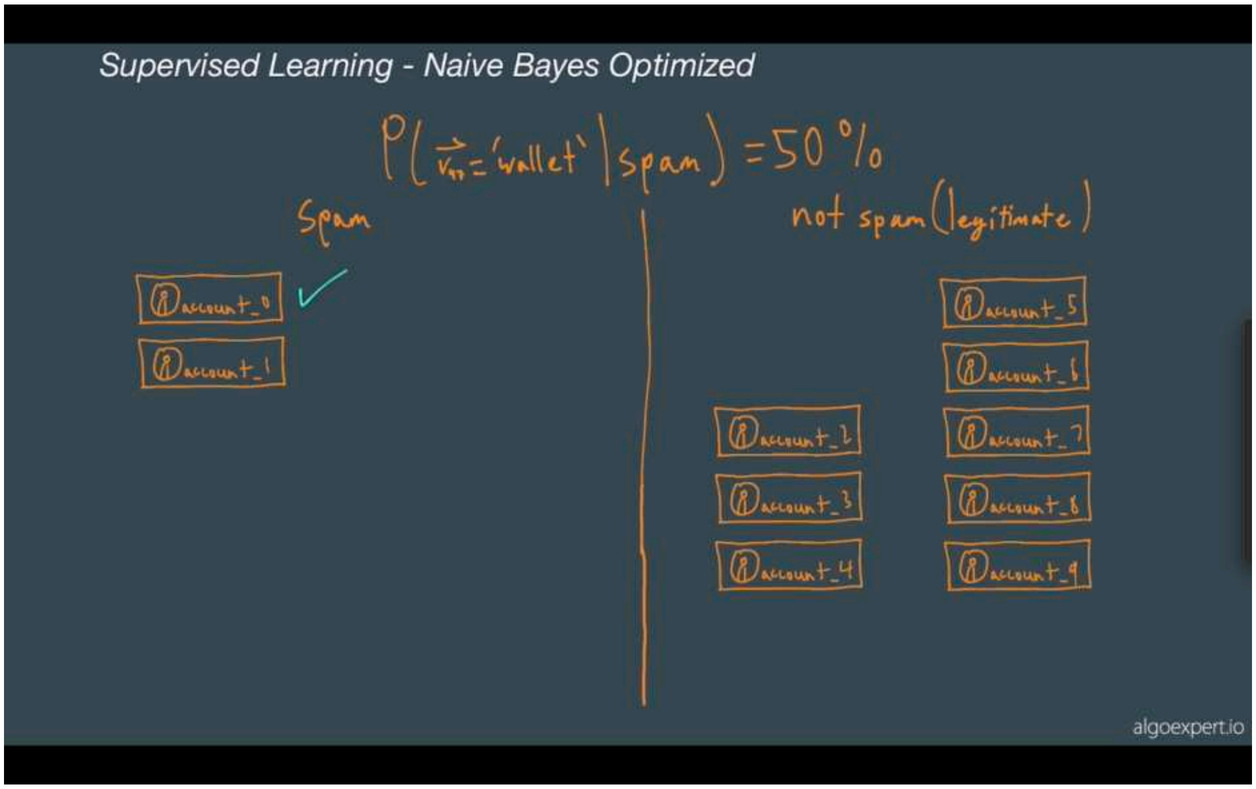
$$\sum_{c} P(K=c) \cdot P(\vec{w} \mid K=c)$$

Let's go through the other terms to see what has changed and what hasn't. Let's start first with the prior of sum class K. Let's say that we wanted to evaluate the prior of tech. Similar to the spam example, all we'd have to do is count the number of tech articles and count the number of non-tech articles, and then just take the ratio of the two. So here we have two tech articles, for non-tech articles that would give us a prior of tech of 33%.

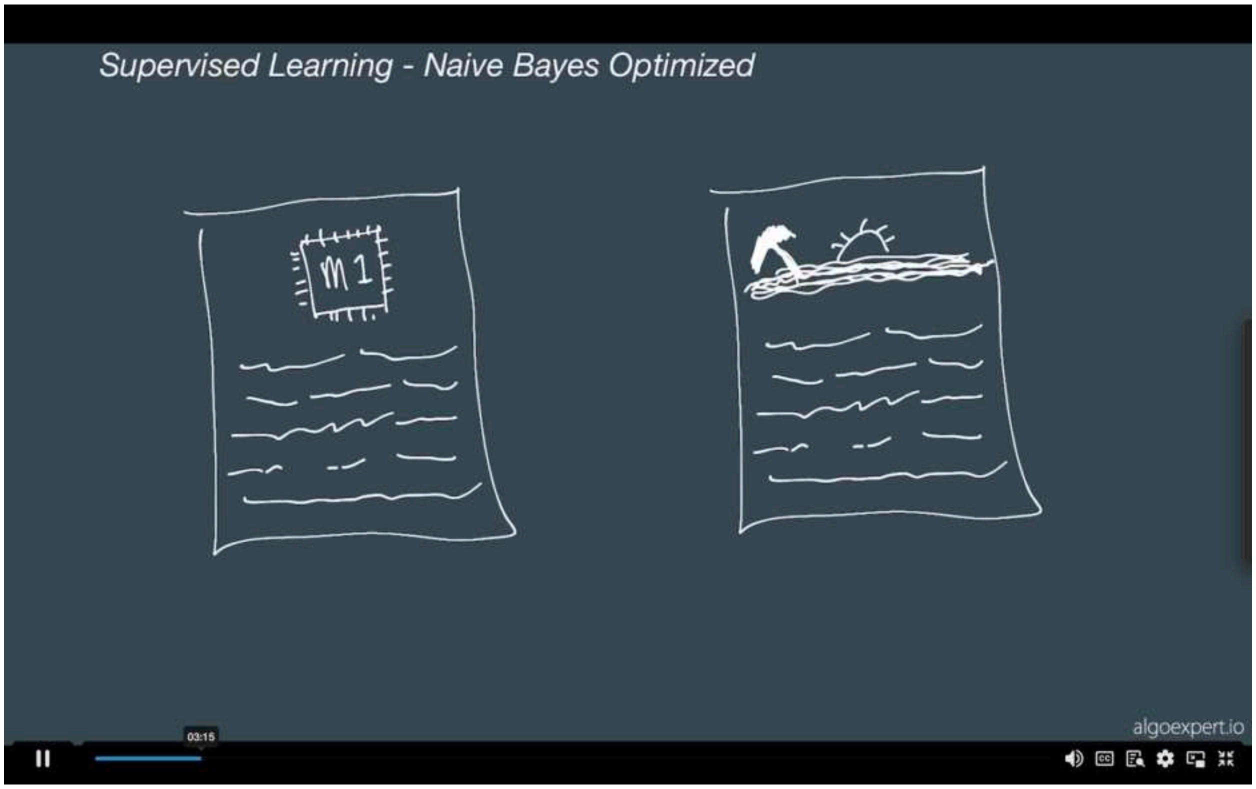

Supervised Learning - Naive Bayes Optimized

$P(\text{tech}) = \frac{2}{2+4} = 33\%$

Moving on to the likelihood term, we have a couple of options. We could treat this the same exact way we treated the spam likelihoods, okay?

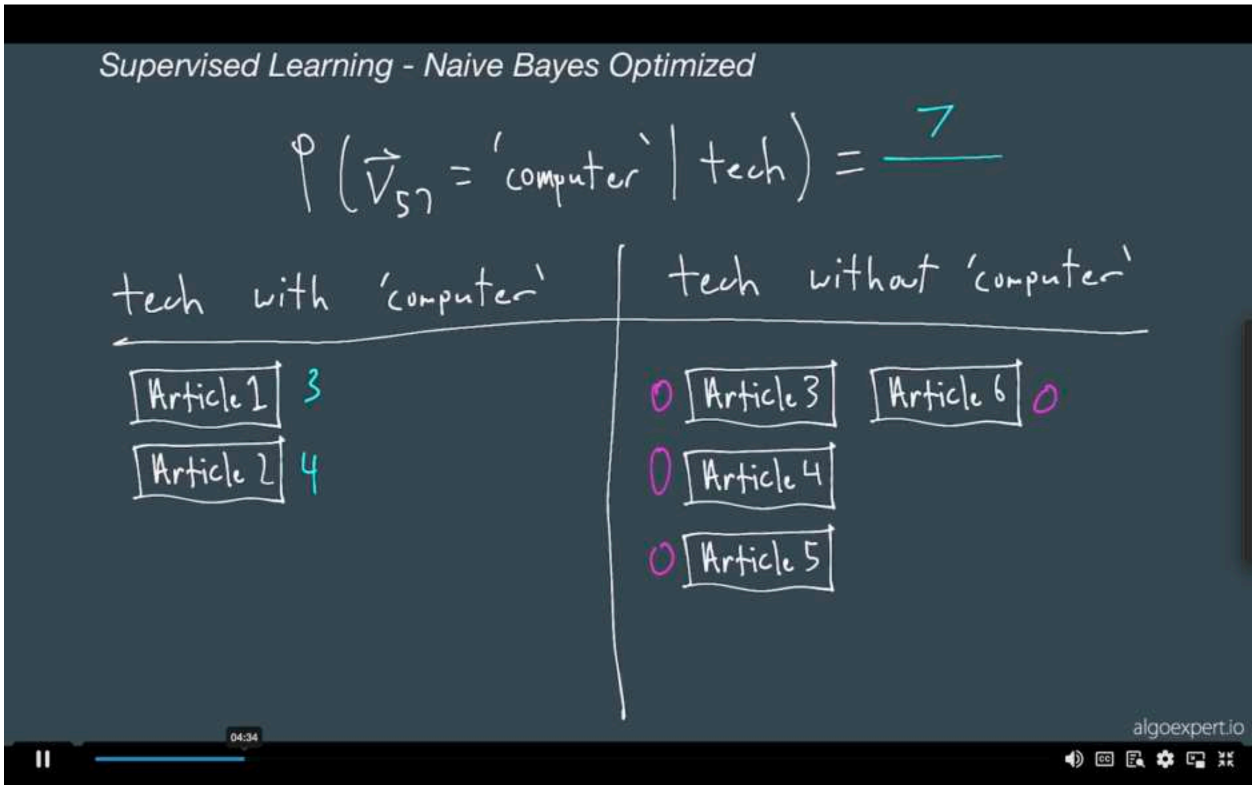


As a refresher, what we did there was we just counted the number of spam messages that contained a particular term here, wallet and divided by the total number of spam messages. So in this case, the word wallet appeared in this message one, but not this message, the other, so we got a 50% likelihood there of the word wallet appearing in a spam message. There's a case for not wanting to do this, with longer messages or articles.

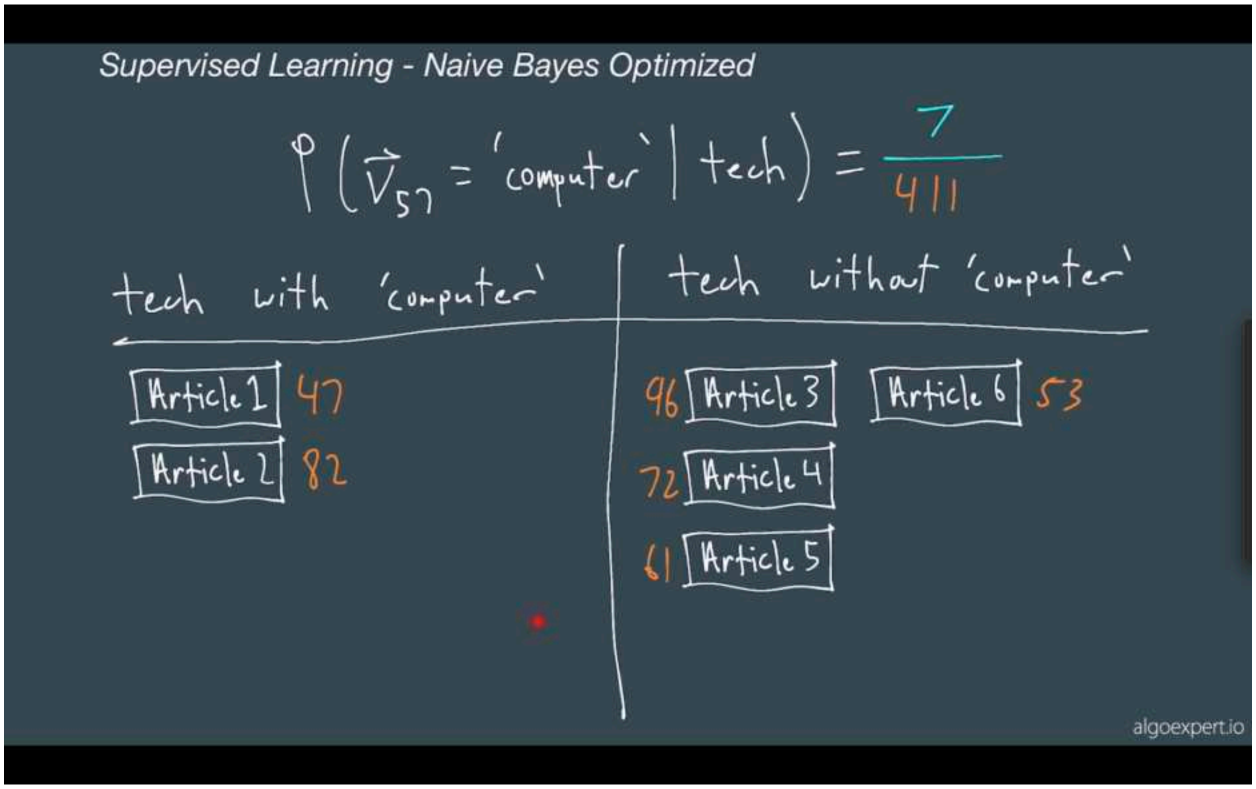


Let's say we have two articles, one on the left is a tech based article. It talks about how Apple is using their own M1 chips, and in the article, it mentions TSMC, which is Taiwan Semiconductor Manufacturing Company and it mentions Taiwan a single time. The article on the right, however, is a travel blog, this blogger had gone through Asia and mentions Taiwan several times in their article. The problem is that if we only model the presence or absence of words, then these two articles would appear to be the same in terms of the word Taiwan to our **Bernoulli model**, which is the model we used in the spam classification. However, if we don't use the Bernoulli model anymore, and we switched to a **multinomial model**, what we'll be modeling instead is the counts of each word in an article instead of the mere presence or absence of a word.

Let's say that we wanted to get the likelihood that the word computer appears in a tech article. What we'd do is we would take all of the tech articles with the word computer, that would be article one and two, and then separate them from all of the tech articles without the word computer.

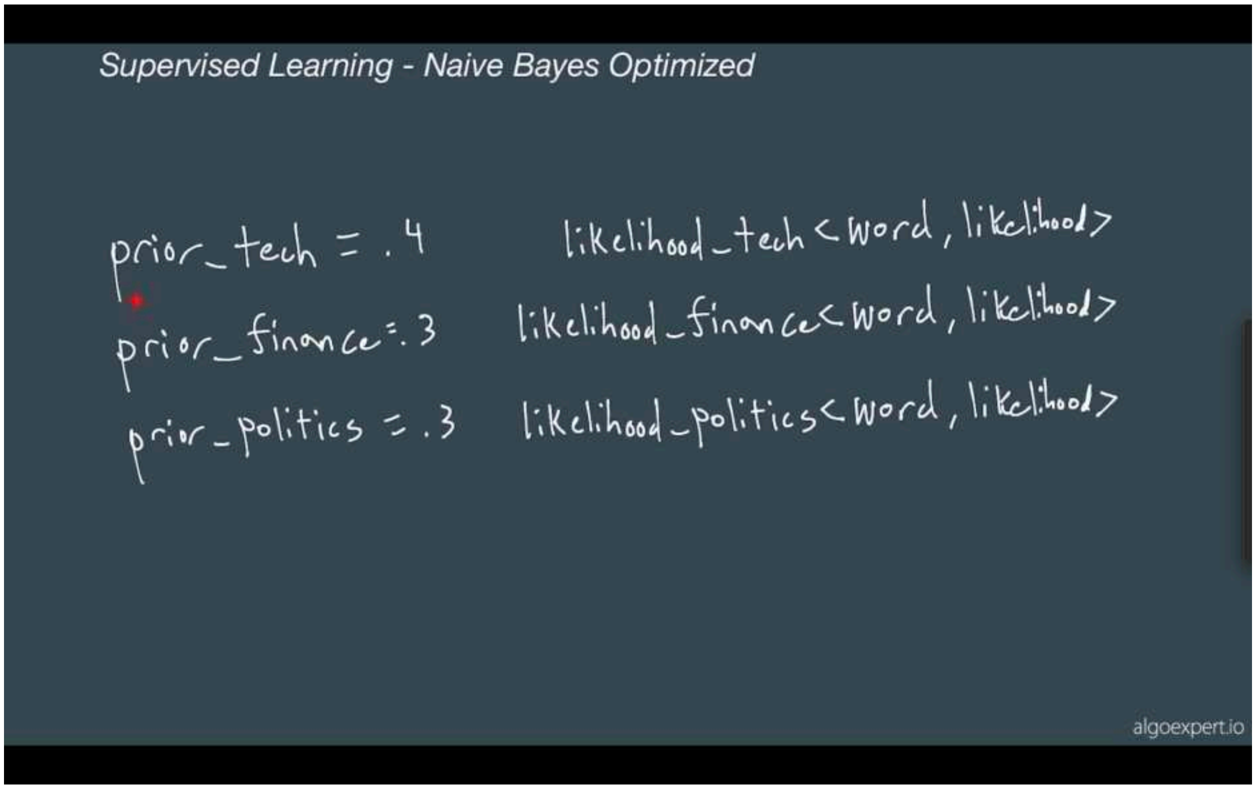


Then we would count the number of times that the word computer appeared in the articles with the word computer in them, and that would go in the numerator of our calculation and in the denominator, we would put the total number of words across all of the articles within the tech tag. So here article one had 47 words, article two had 82 words, article three had 96 words, we would sum all of those up and put them in the denominator of our probability calculator.



So if you remember, just to review what we did with the spam model, we would take our input message, we would map that to be in terms of our known vocabulary. This would be our model and we'd model either the absence or the presence of a particular word, and we'd have our priors of spam and priors of not spam, and then we'd have our spam likelihood maps and our non-spam likelihood maps.

For this case, since we have multiple classes, we're gonna have multiple priors and we're going to have a likelihood map for each class as well.



Supervised Learning - Naive Bayes Optimized

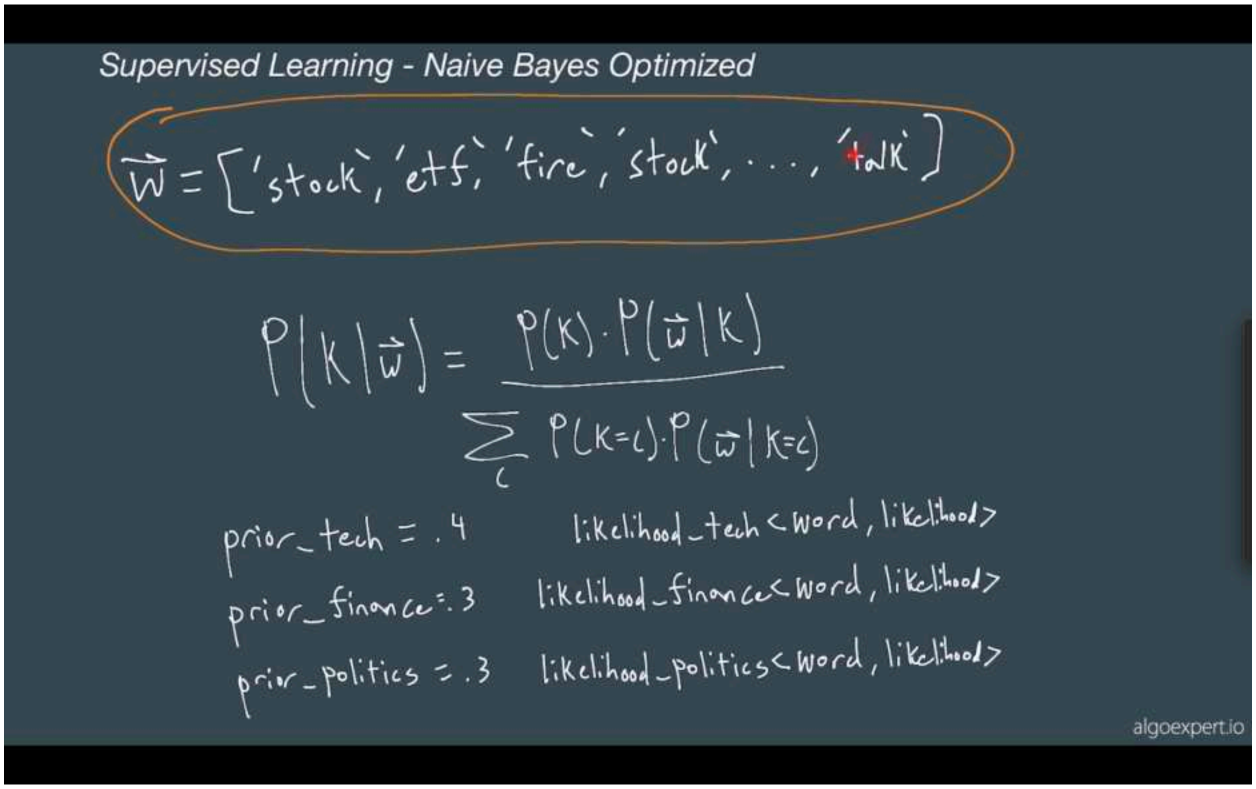
prior_tech = .4 likelihood_tech<word, likelihood>

prior_finance = .3 likelihood_finance<word, likelihood>

prior_politics = .3 likelihood_politics<word, likelihood>

algoexpert.io

The model is going to look the exact same in terms of like the denominator in the prior, the likelihood, however, will change. Specifically, let's look at how that will change with regards to an input message. So we'll have our input message and we'll calculate the likelihoods in this way.



Supervised Learning - Naive Bayes Optimized

$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$

$$P(K|\vec{w}) = \frac{P(K) \cdot P(\vec{w}|K)}{\sum_{c} P(K=c) \cdot P(\vec{w}|K=c)}$$

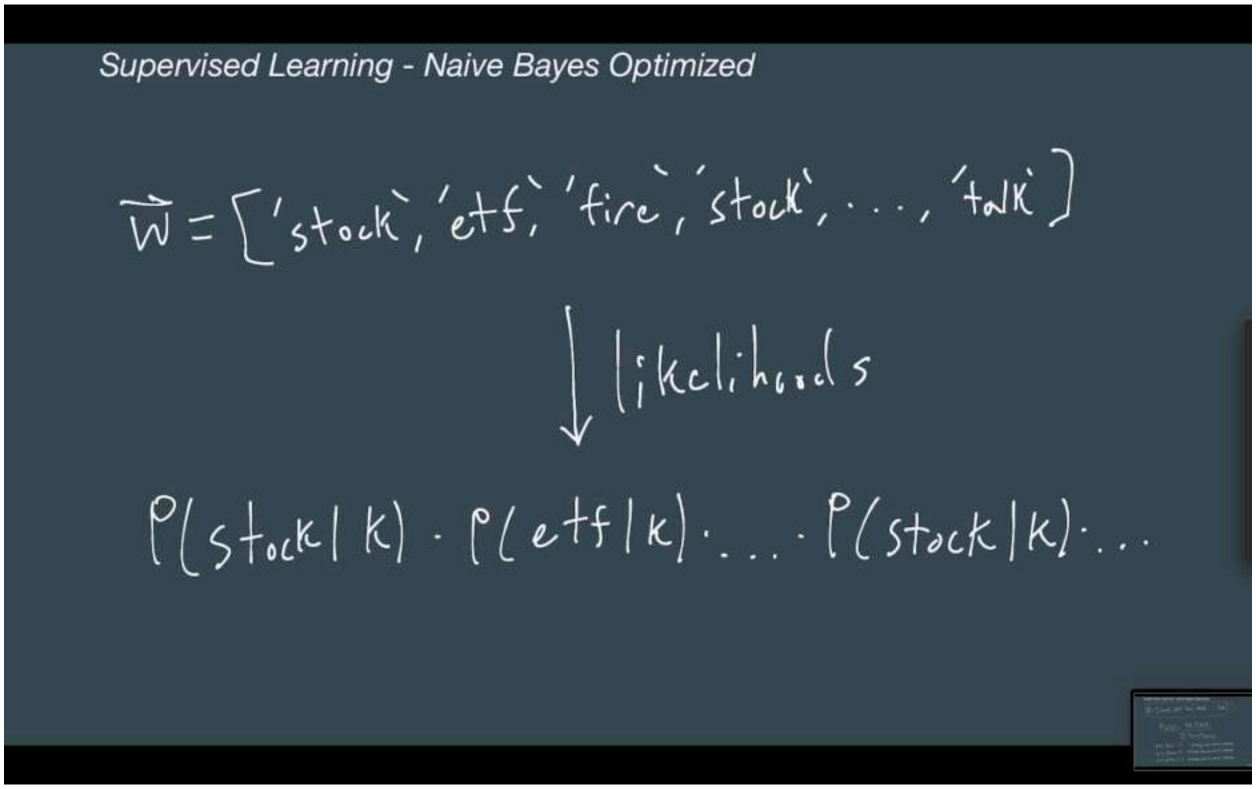
prior_tech = .4 likelihood_tech<word, likelihood>

prior_finance = .3 likelihood_finance<word, likelihood>

prior_politics = .3 likelihood_politics<word, likelihood>

algoexpert.io

So stock will have its own likelihood of the probability of stock given some class. ETF will also have its own term, fire will have some.



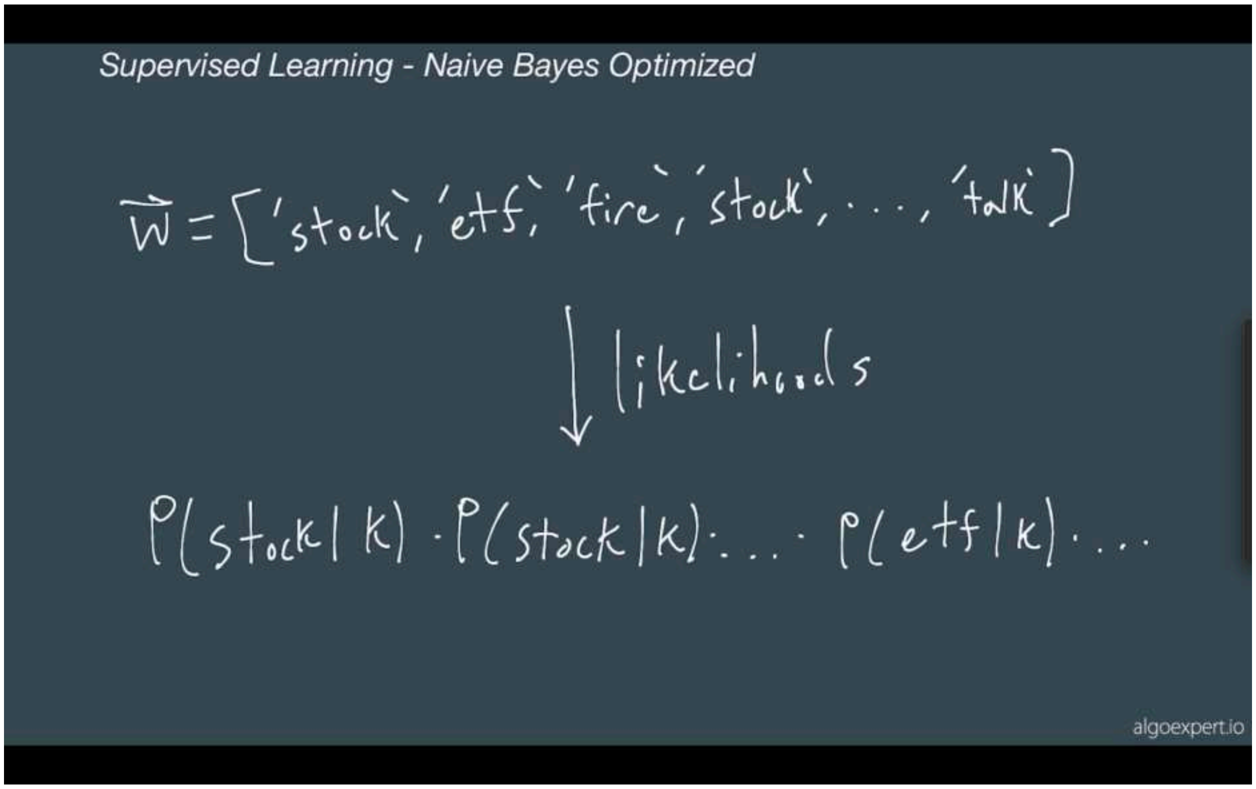
Supervised Learning - Naive Bayes Optimized

$$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$$

$$\Big\downarrow \text{likelihoods}$$

$$P(stock \mid k) \cdot P(etf \mid k) \cdot \ldots \cdot P(stock \mid k) \cdot \ldots$$

This looks very similar to how we modeled it in the Bernoulli way, however, the difference here is that if we reorder these, we can see that this term actually appears twice since the word stock is in our message twice we can see that we actually hit that probability twice.



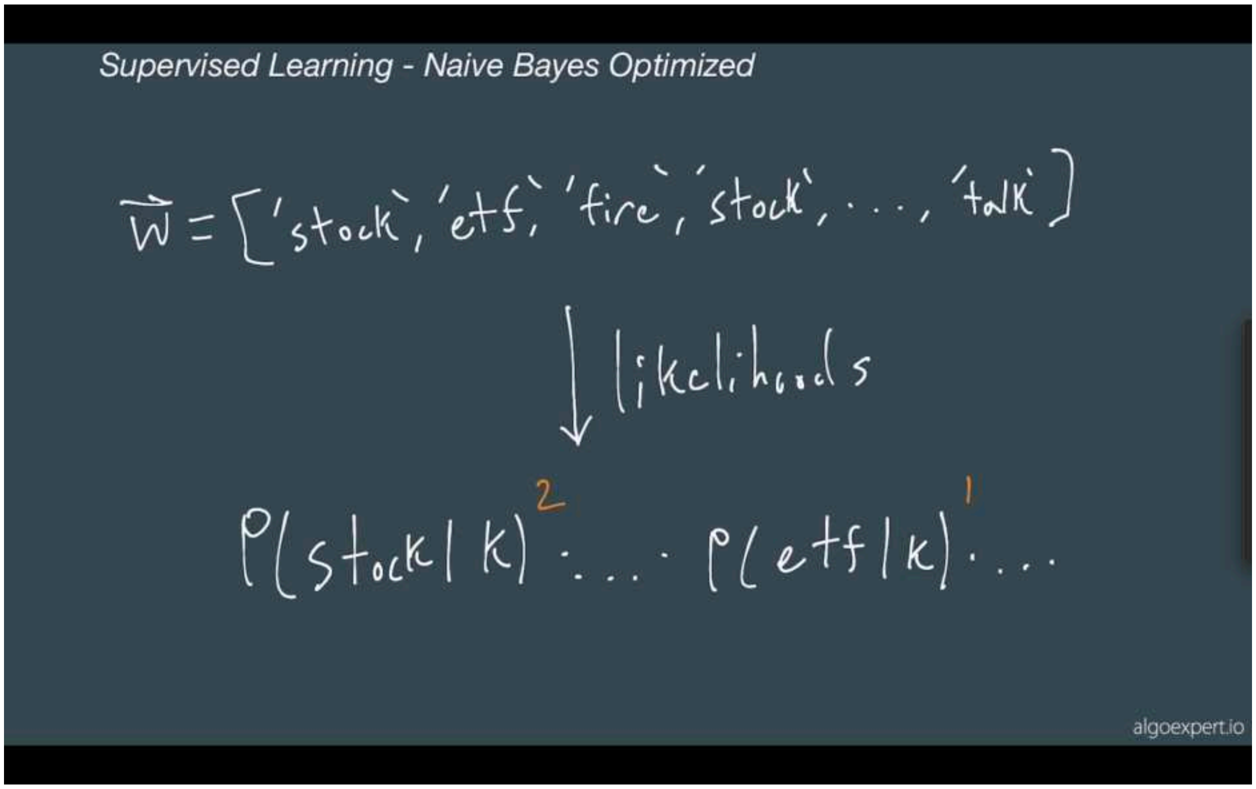
Supervised Learning - Naive Bayes Optimized

$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$

$\big\downarrow$ likelihoods

$P(stock \mid k) \cdot P(stock \mid k) \cdot \ldots \cdot P(etf \mid k) \cdot \ldots$

Okay, so we can actually combine them into powers. So the probability of stock given sum class times the probability of stock given sum class, it's just the probability of stock given class to the second power. ETF only appeared one time, so it would only be to the power of one.



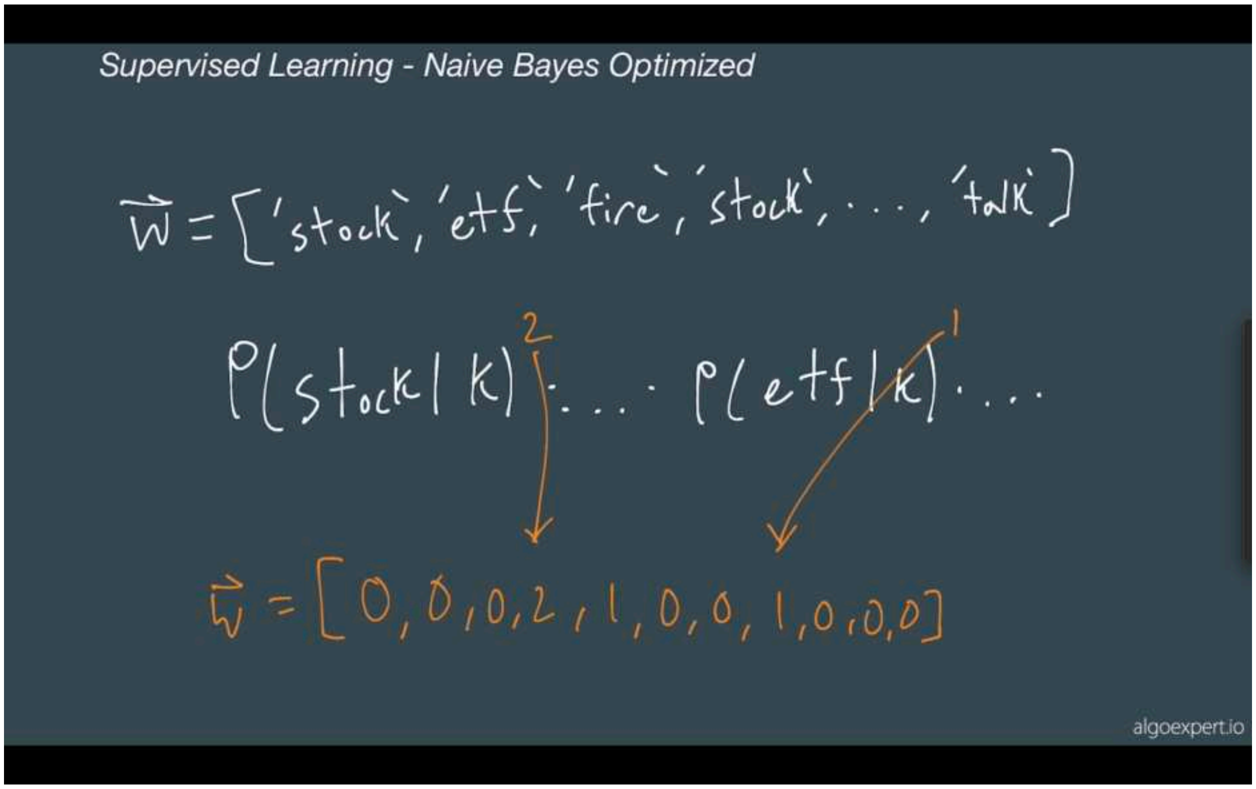
Supervised Learning - Naive Bayes Optimized

$$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$$

$$\Big\downarrow \text{likelihoods}$$

$$P(\text{stock} \mid K)^{2} \cdot \ldots \cdot P(\text{etf} \mid K)^{1} \cdot \ldots$$

What these really are, is just a count of the number of words with respect to the element in our vocabulary, right? So if we wanted to model the input message W we would say, okay, the zero with word in our vocabulary occurs zero times, the first zero times, this element in our vocabulary appears twice and we would model that as just the power of the probability of the likelihood. Okay, so now we have the representation of our input message in terms of their counts.



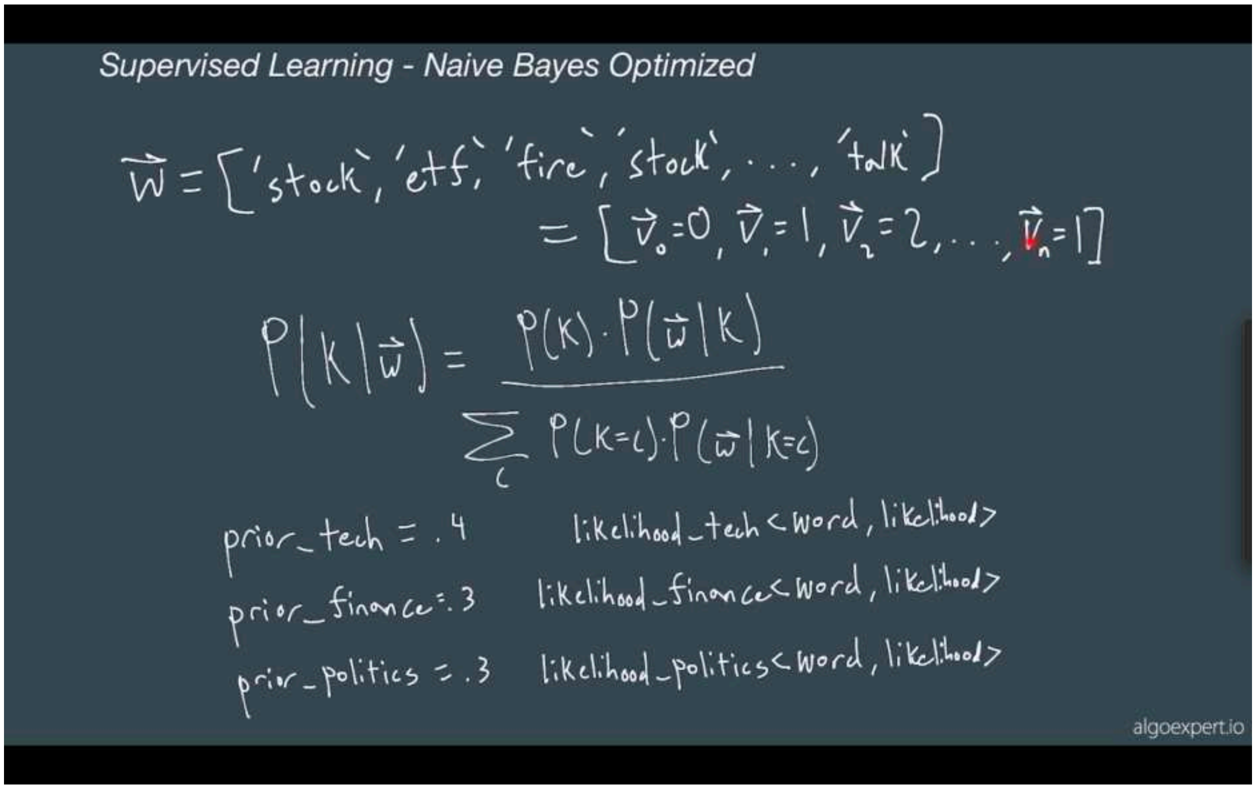
Supervised Learning - Naive Bayes Optimized

$$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$$

$$P(stock \mid k)^{2} \cdot \ldots \cdot P(etf \mid k)^{1} \cdot \ldots$$

$$\vec{w} = [0,0,0,2,1,0,0,1,0,0,0]$$

So let's say that we wanted to evaluate this message in terms of it being in the category of finance.



Supervised Learning - Naive Bayes Optimized

$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$

$= [\vec{v}_0 = 0, \vec{v}_1 = 1, \vec{v}_2 = 2, \ldots, \vec{v}_n = 1]$

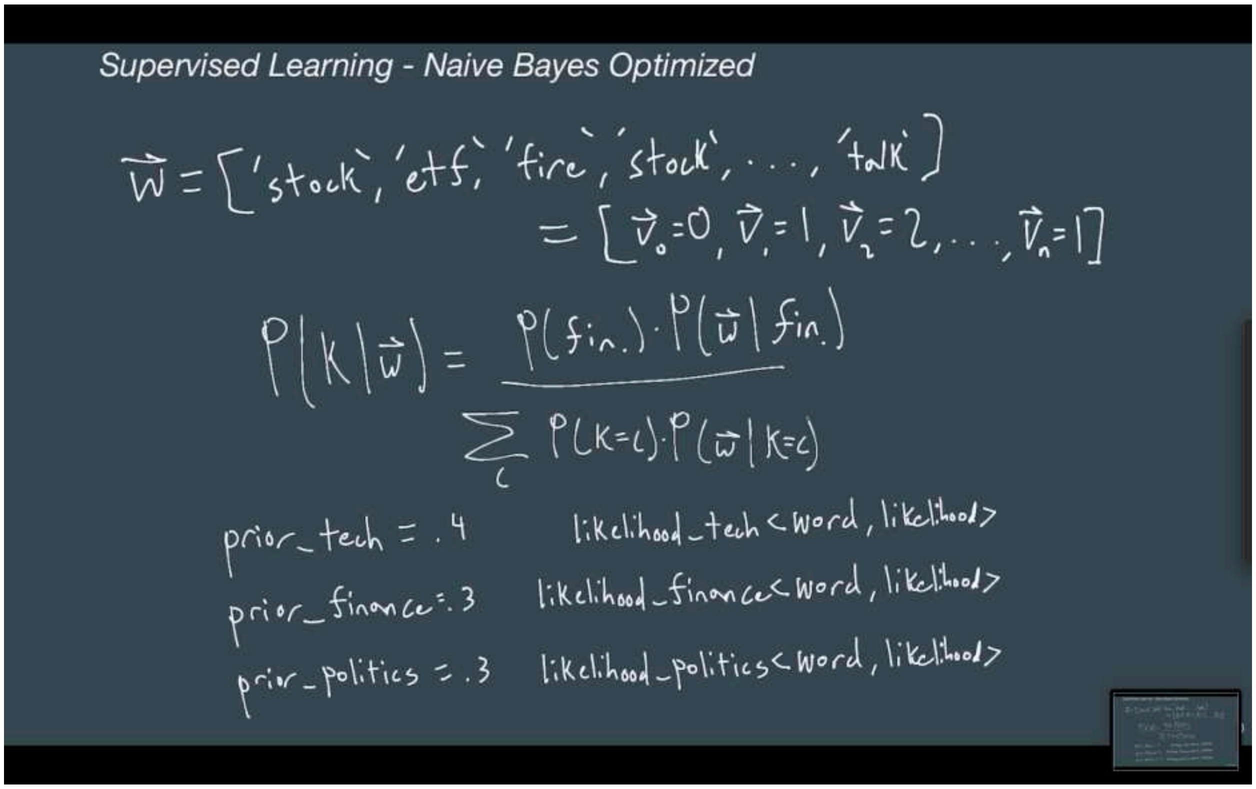
$$P(K|\vec{w}) = \frac{P(K)\cdot P(\vec{w}|K)}{\sum_c P(K=c)\cdot P(\vec{w}|K=c)}$$

prior_tech = .4 likelihood_tech<word, likelihood>

prior_finance = .3 likelihood_finance<word, likelihood>

prior_politics = .3 likelihood_politics<word, likelihood>

algoexpert.io



Supervised Learning - Naive Bayes Optimized

$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$

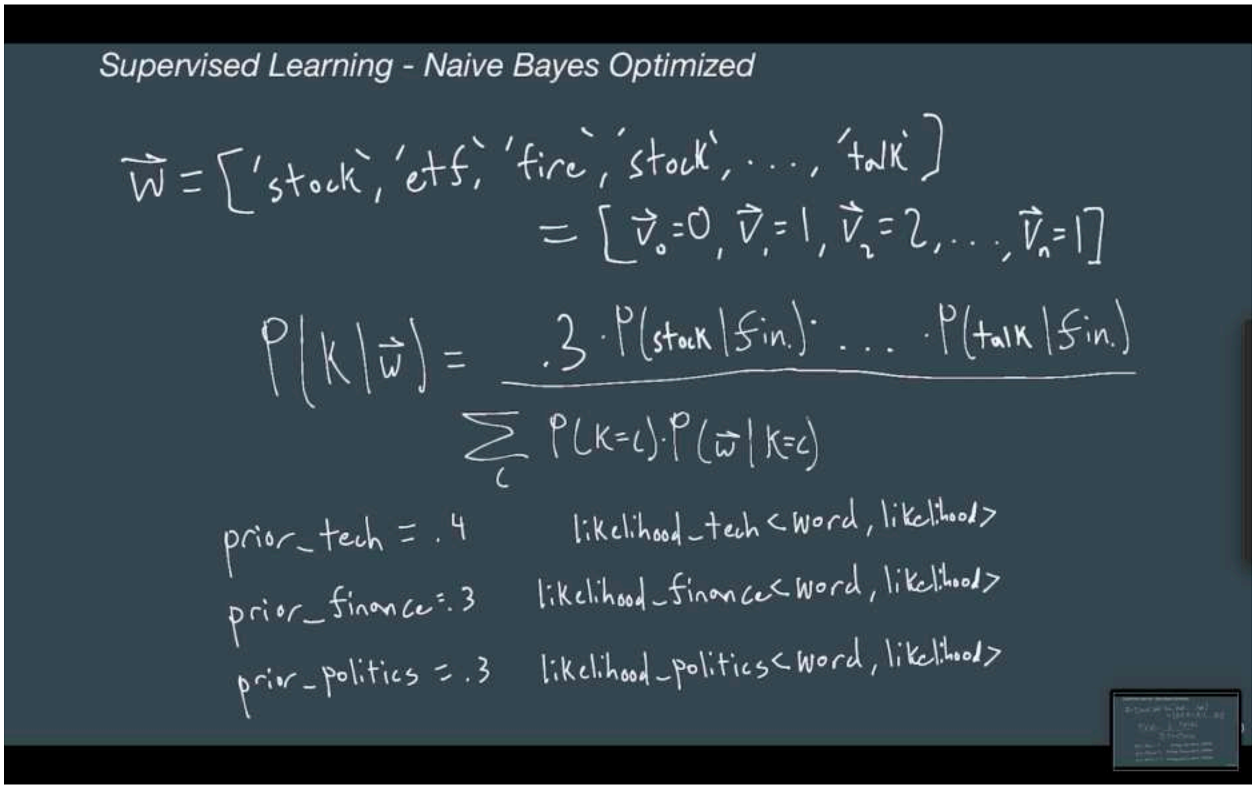
$= [\vec{v}_0 = 0, \vec{v}_1 = 1, \vec{v}_2 = 2, \ldots, \vec{v}_n = 1]$

$$P(K|\vec{w}) = \frac{P(\text{fin.})\cdot P(\vec{w}|\text{fin.})}{\sum_c P(K=c)\cdot P(\vec{w}|K=c)}$$

prior_tech = .4 likelihood_tech<word, likelihood>

prior_finance = .3 likelihood_finance<word, likelihood>

prior_politics = .3 likelihood_politics<word, likelihood>



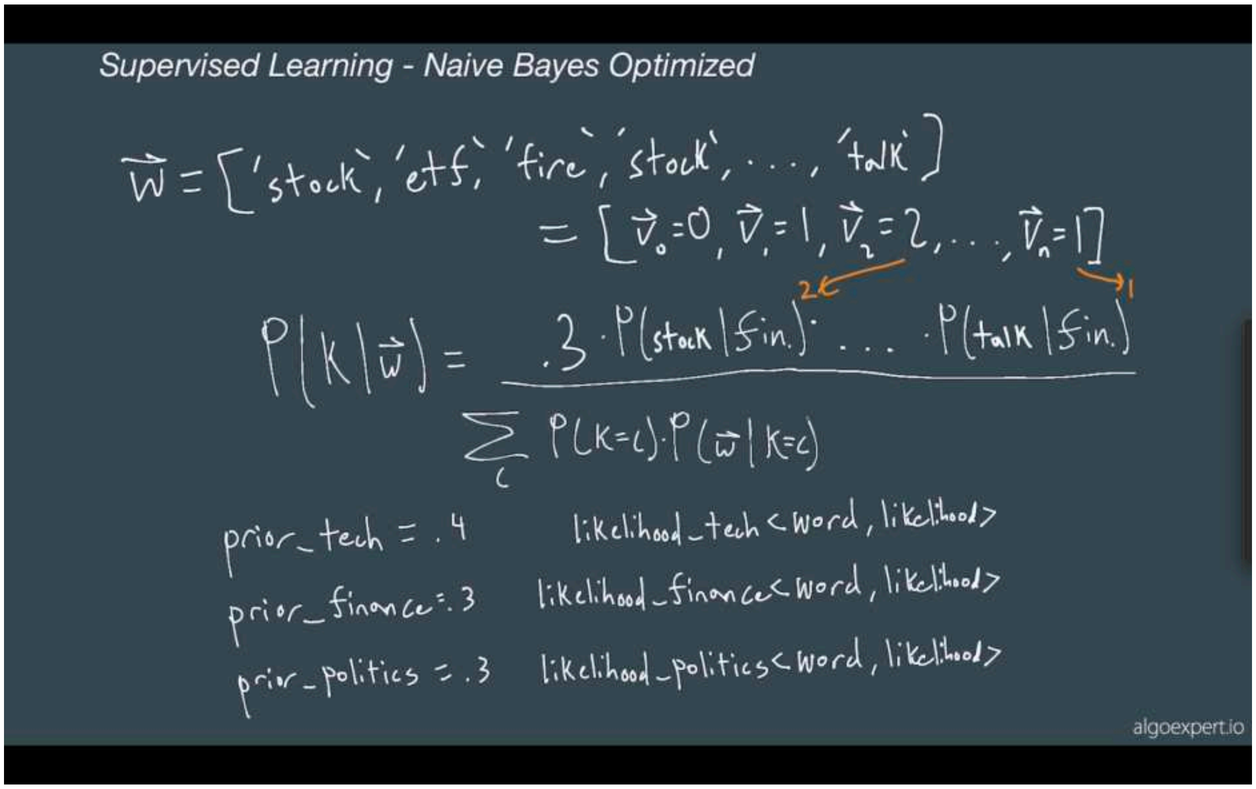
Supervised Learning - Naive Bayes Optimized

$$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$$

$$= [\vec{v}_0 = 0, \vec{v}_1 = 1, \vec{v}_2 = 2, \ldots, \vec{v}_n = 1]$$

$$P(K|\vec{w}) = \frac{.3 \cdot P(\text{stock}|\text{fin.}) \cdot \ldots \cdot P(\text{talk}|\text{fin.})}{\sum_c P(K=c) \cdot P(\vec{w}|K=c)}$$

prior_tech = .4	likelihood_tech<word, likelihood>
prior_finance = .3	likelihood_finance<word, likelihood>
prior_politics = .3	likelihood_politics<word, likelihood>



Supervised Learning - Naive Bayes Optimized

$$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$$

$$= [\vec{v}_0 = 0, \vec{v}_1 = 1, \vec{v}_2 = 2, \ldots, \vec{v}_n = 1]$$

$$2 \leftarrow \qquad \qquad \rightarrow 1$$

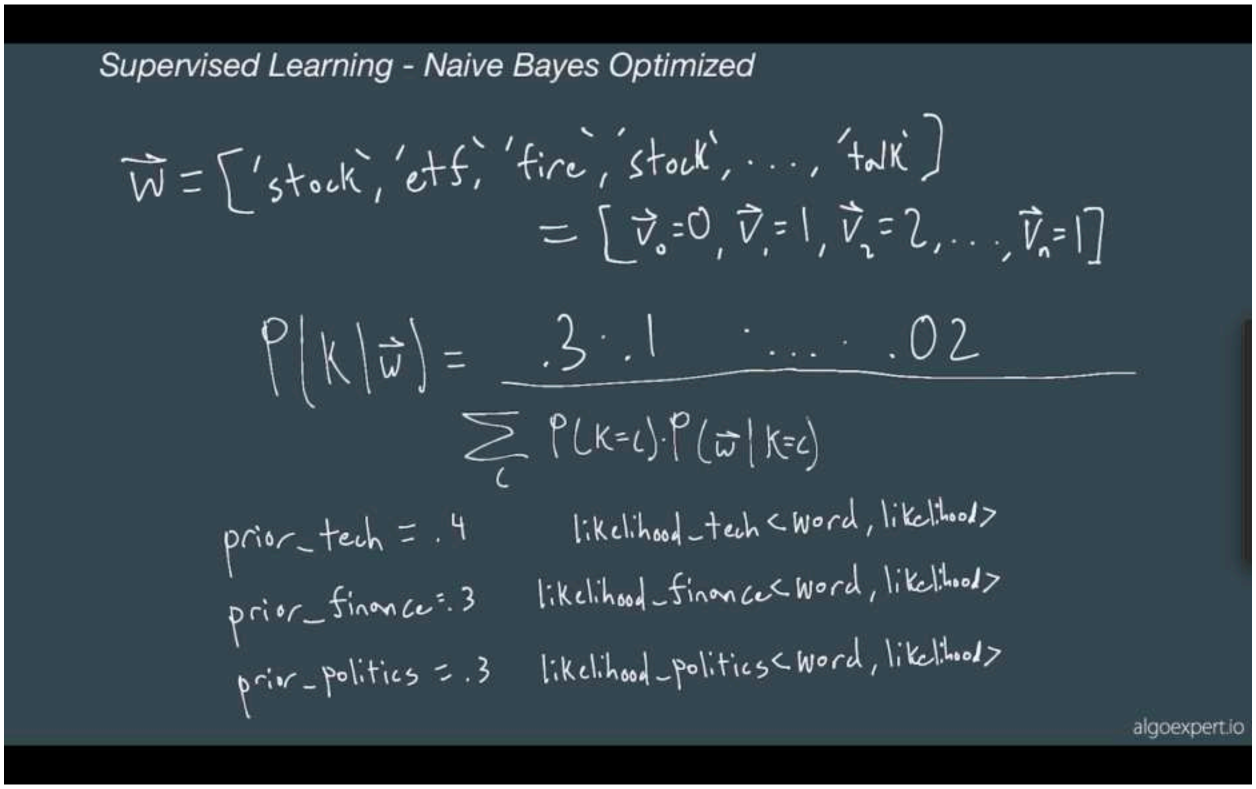
$$P(K|\vec{w}) = \frac{.3 \cdot P(\text{stock}|\text{fin.}) \cdot \ldots \cdot P(\text{talk}|\text{fin.})}{\sum_c P(K=c) \cdot P(\vec{w}|K=c)}$$

prior_tech = .4	likelihood_tech<word, likelihood>
prior_finance = .3	likelihood_finance<word, likelihood>
prior_politics = .3	likelihood_politics<word, likelihood>

algoexpert.io

Okay, so first we would replace this prior here, and then we would expand these likelihood terms out, and then all we would do is apply our vocabulary representation as powers to the likelihoods themselves.

So if you have a count of zero, then your power zero, and you would just have a value of one. However, since stock appears twice in our message, then the probability of stock given some finance article, we actually put a two in the power and for talk, since talk only occurs a single time that would just have a power of one. But we can run those calculations.



Supervised Learning - Naive Bayes Optimized

$$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \dots, \text{'talk'}]$$

$$= [\vec{v}_0 = 0, \vec{v}_1 = 1, \vec{v}_2 = 2, \dots, \vec{v}_n = 1]$$

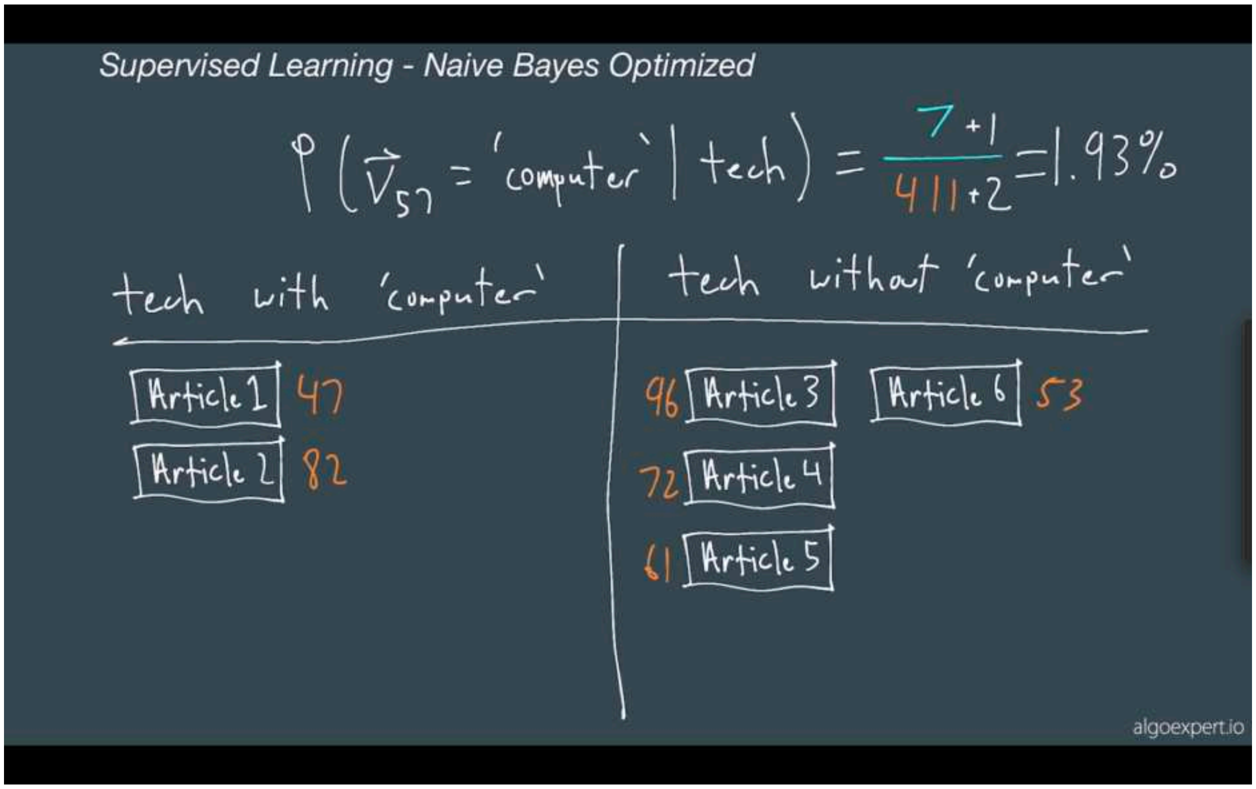
$$P(K|\vec{w}) = \frac{.3 \cdot .1 \cdot \dots \cdot .02}{\sum_c P(K=c) \cdot P(\vec{w}|K=c)}$$

prior_tech = .4 likelihood_tech<word, likelihood>

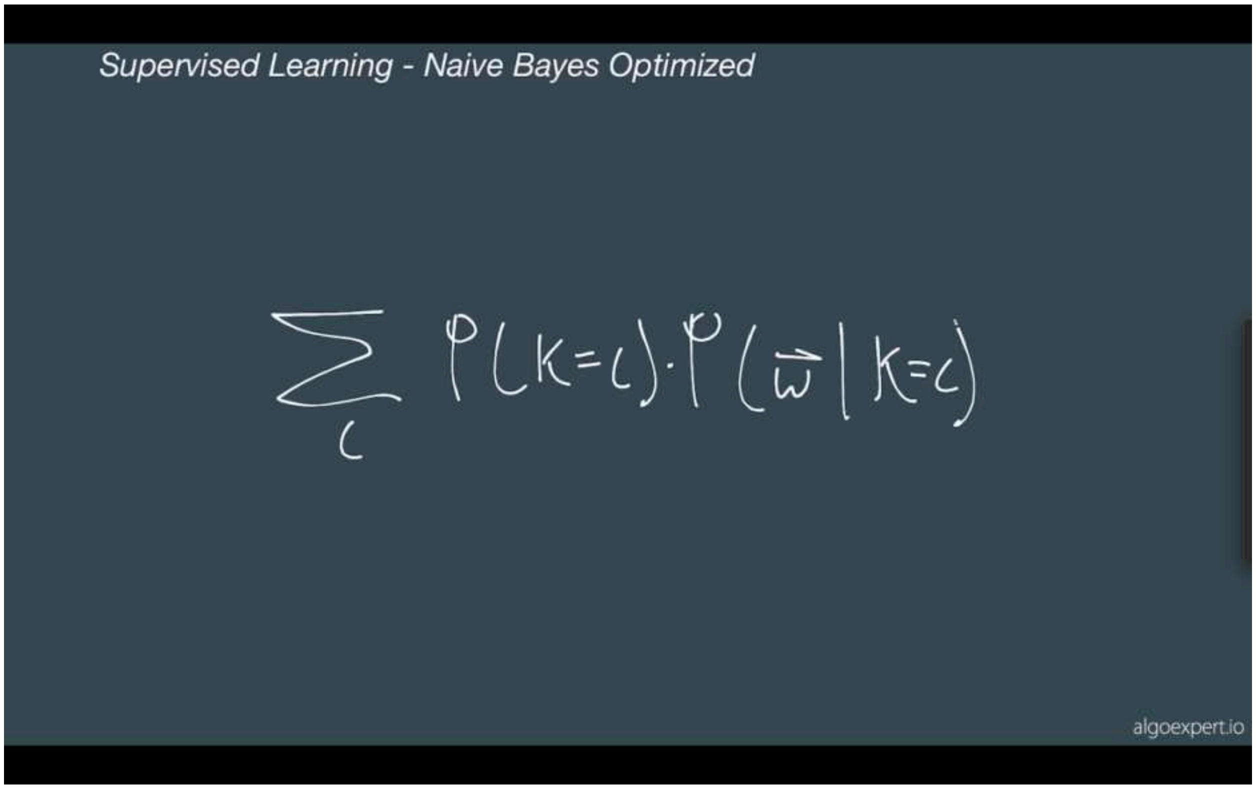
prior_finance = .3 likelihood_finance<word, likelihood>

prior_politics = .3 likelihood_politics<word, likelihood>

Now, you may have noticed that there, we could still run into the issue of having a zero element in the numerator. We talked about this with the spam example, and we use something called Laplace smoothing, we'll take a similar approach here.



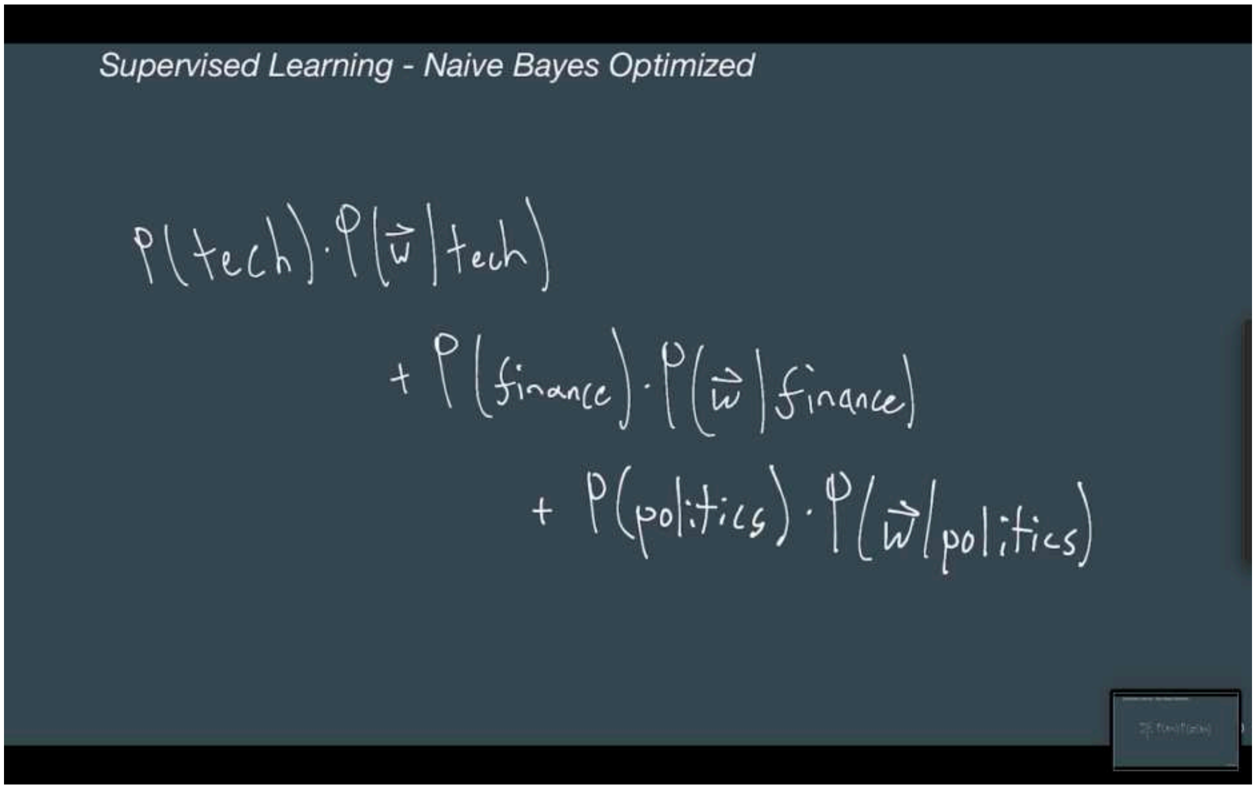
So going back to the likelihood of the word computer appearing in a tech article, nothing would change in terms of our counts or our calculation, except for the fact that we would add one to the numerator and two to the denominator to include Laplace smoothing.



Supervised Learning - Naive Bayes Optimized

$$\sum_{c} P(K=c)\cdot P(\vec{w} \mid K=c)$$

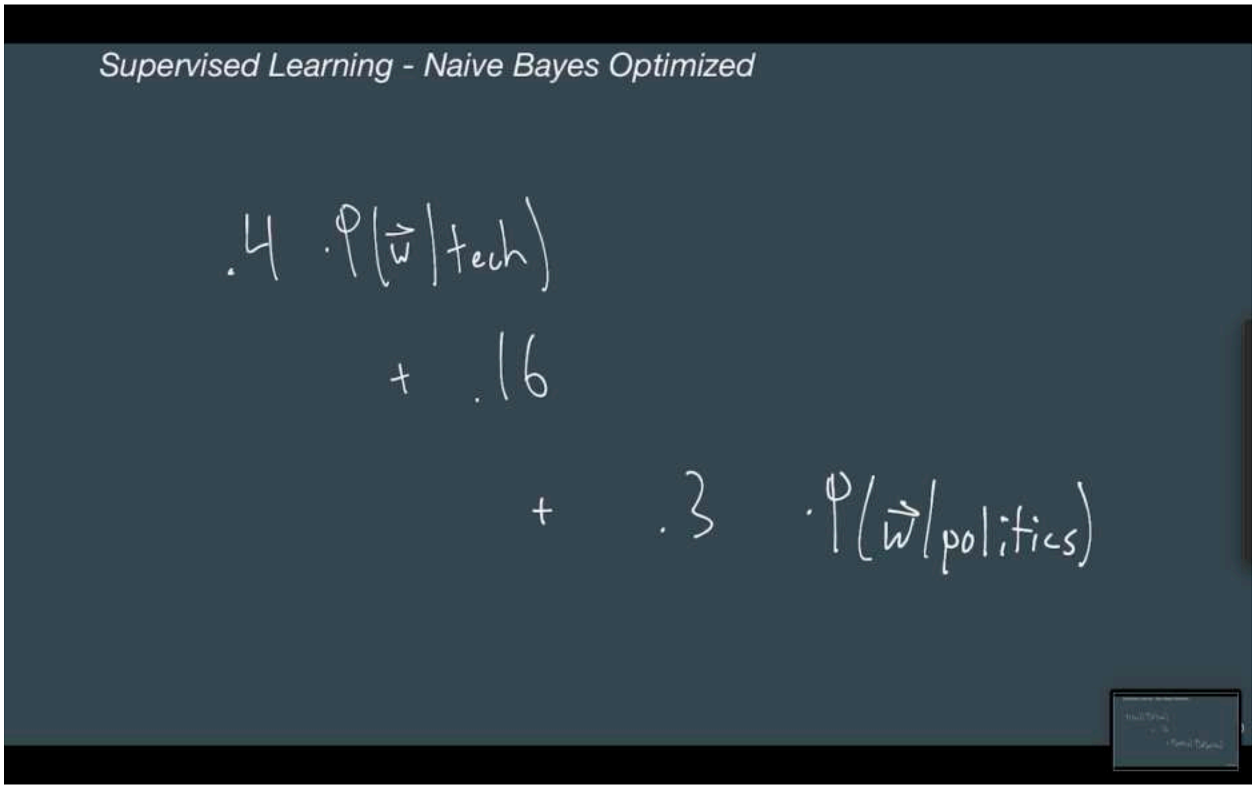
This would ensure that we don't get those zero probabilities in our likelihood calculations. So now let's go ahead and look at the denominator. Let's expand this condensed form.



Supervised Learning - Naive Bayes Optimized

$$P(tech) \cdot P(\vec{w} \mid tech) + P(finance) \cdot P(\vec{w} \mid finance) + P(politics) \cdot P(\vec{w} \mid politics)$$

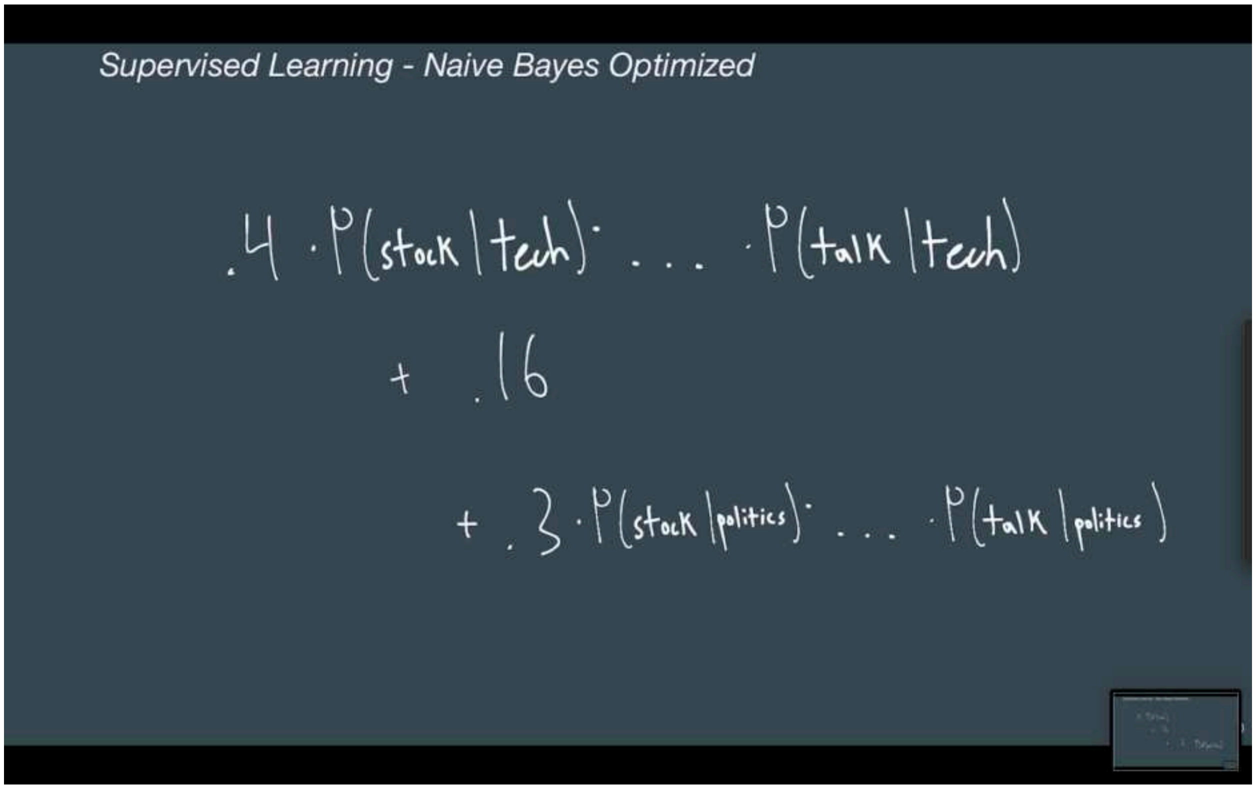
So we're actually iterating through every class. We've already calculated this finance terms, so we can go ahead and get rid of that.



Supervised Learning - Naive Bayes Optimized

$$.4 \cdot P(\vec{w} | tech) + .16 + .3 \cdot P(\vec{w} | politics)$$

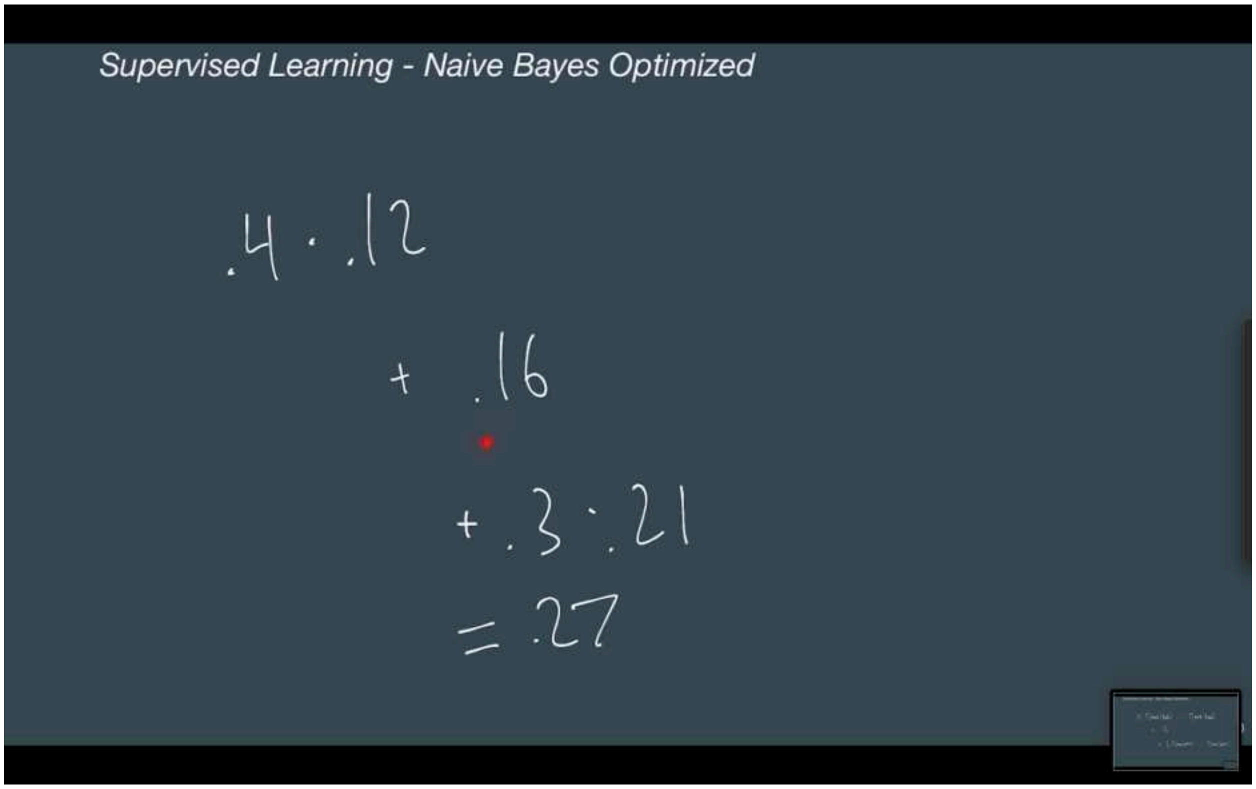
We can look up our priors for tech and politics, we can expand these terms in terms of our vocabulary as well.



Supervised Learning - Naive Bayes Optimized

$$.4 \cdot P(\text{stock} \mid \text{tech}) \cdot \ldots \cdot P(\text{talk} \mid \text{tech})$$
$$+ .16$$
$$+ .3 \cdot P(\text{stock} \mid \text{politics}) \cdot \ldots \cdot P(\text{talk} \mid \text{politics})$$

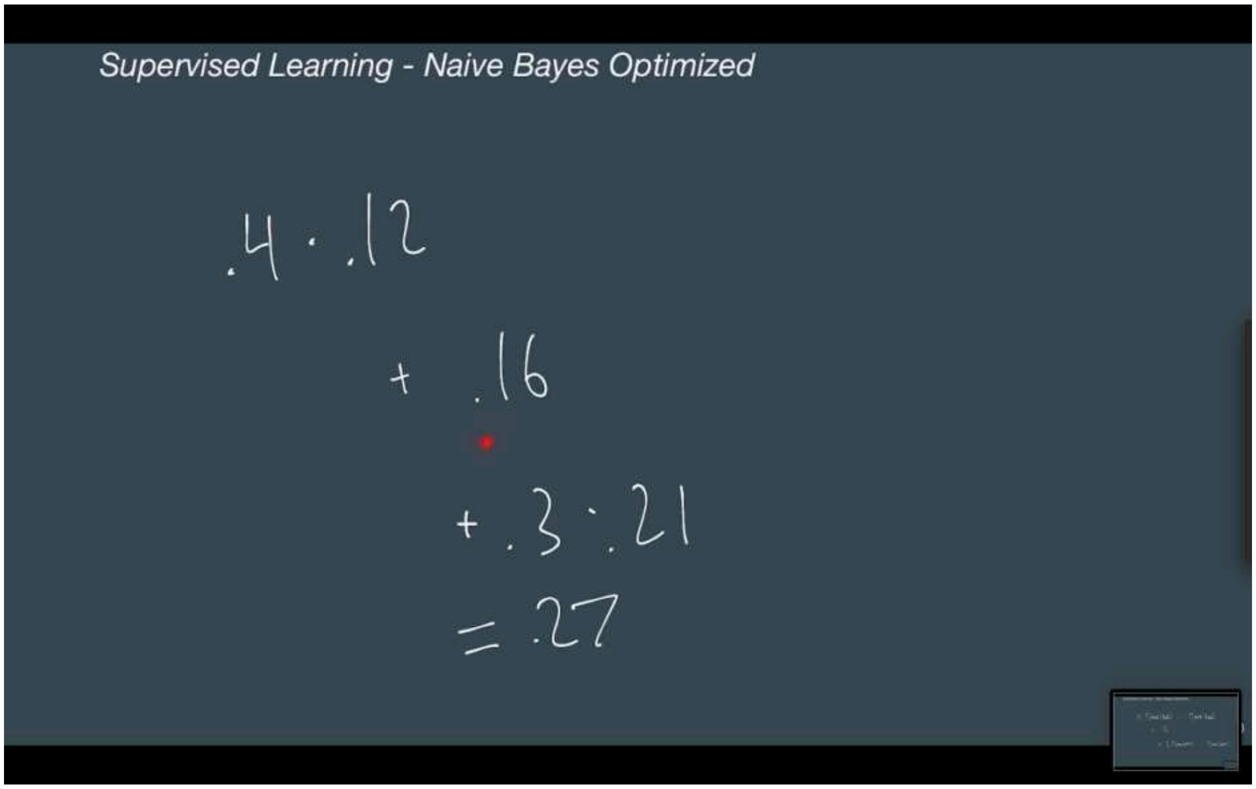
We would apply the powers rule here as well with the counts.



Supervised Learning - Naive Bayes Optimized

$$.4 \cdot .12$$
$$+ .16$$
$$+ .3 \cdot .21$$
$$= .27$$

So we can crunch those numbers and we get 0.27 for the denominator.



Supervised Learning - Naive Bayes Optimized

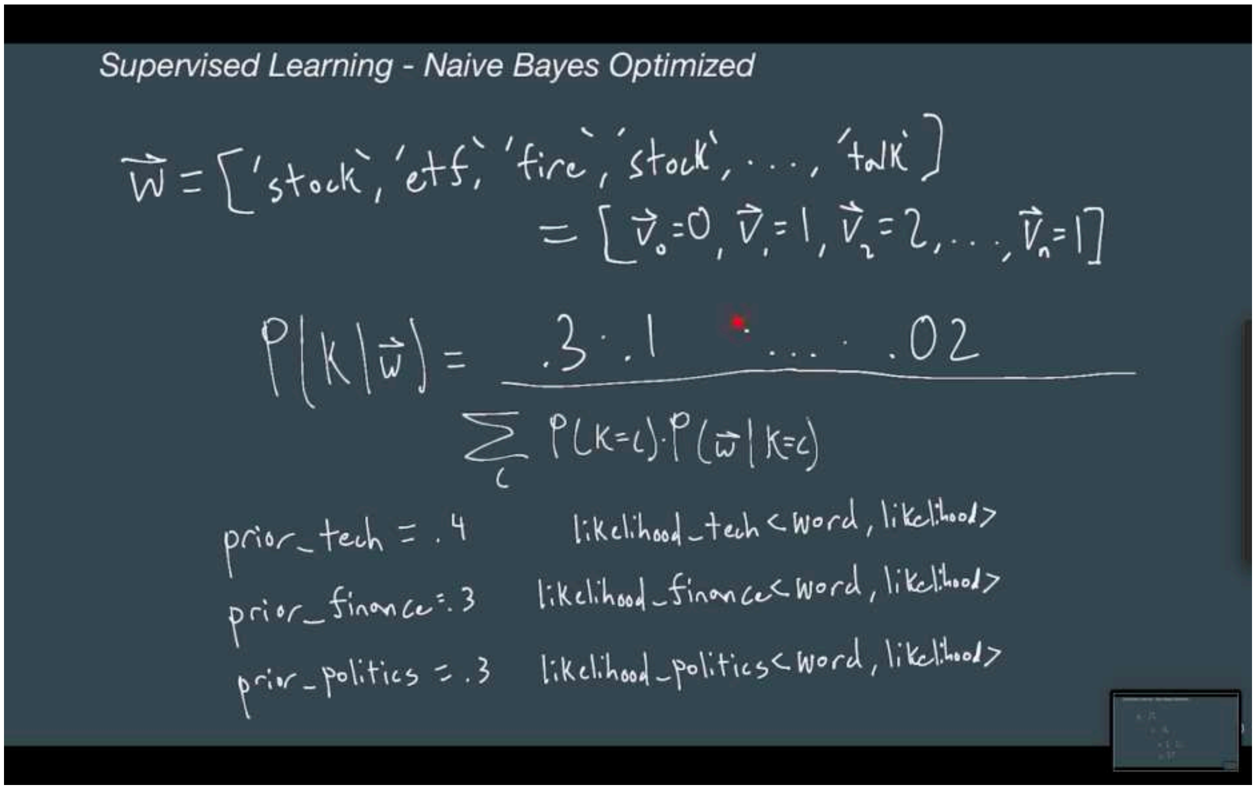
$.4 \cdot .12$

$+ .16$

$+ .3 \cdot .21$

$= .27$

So if we crunch these numbers up top, we get 0.16, we can substitute in the 0.27 for the denominator, this results in a 59% probability that this input article should belong in the finance category or receive the finance tag.

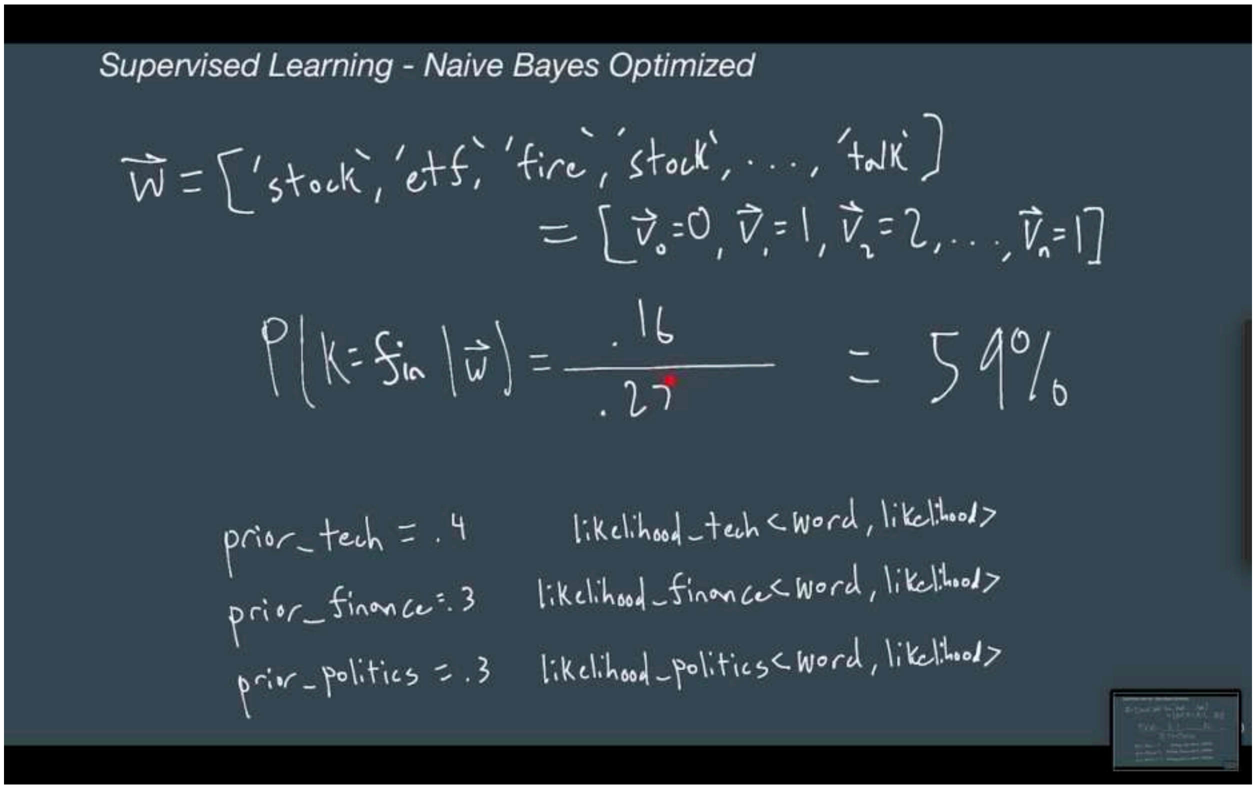


Supervised Learning - Naive Bayes Optimized

$$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$$
$$= [\vec{v}_0 = 0, \vec{v}_1 = 1, \vec{v}_2 = 2, \ldots, \vec{v}_n = 1]$$

$$P(K|\vec{w}) = \frac{.3 \cdot .1 \cdot \ldots \cdot .02}{\sum_c P(K=c) \cdot P(\vec{w}|K=c)}$$

prior_tech = .4	likelihood_tech<word, likelihood>
prior_finance = .3	likelihood_finance<word, likelihood>
prior_politics = .3	likelihood_politics<word, likelihood>



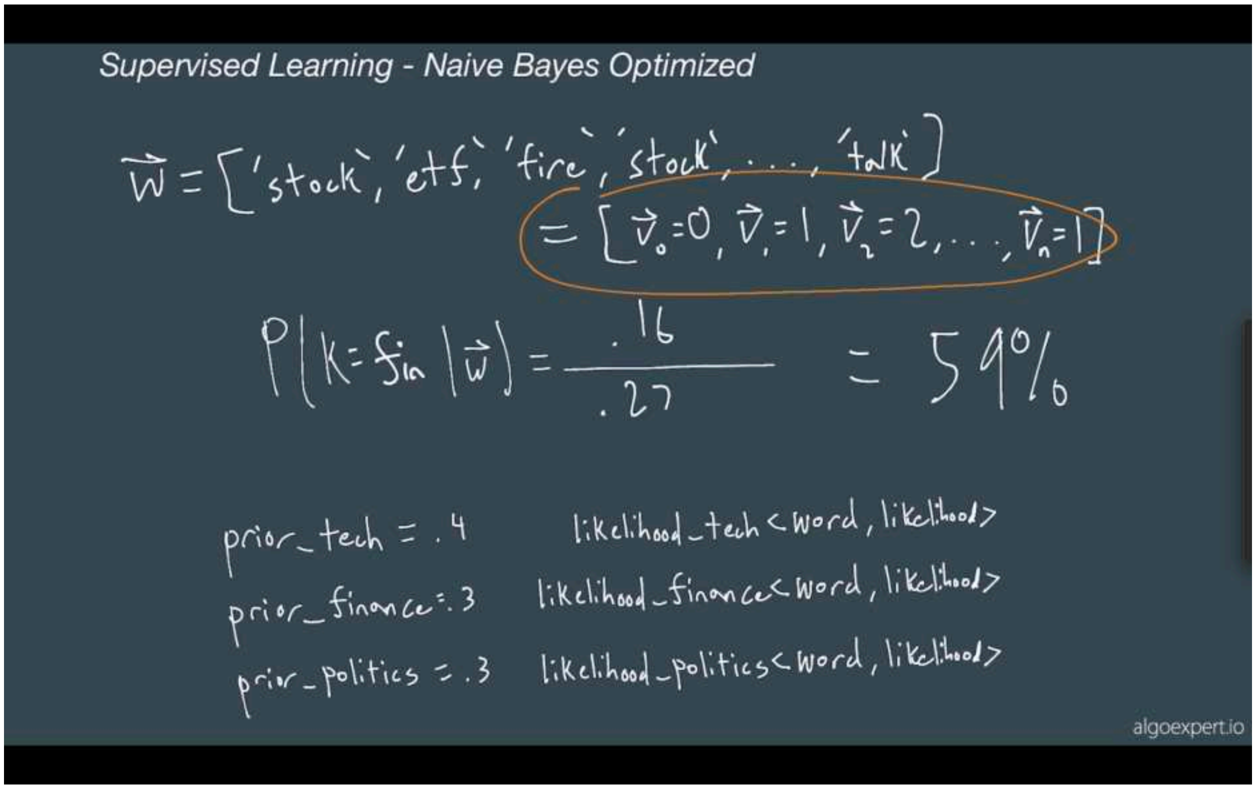
Supervised Learning - Naive Bayes Optimized

$$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$$
$$= [\vec{v}_0 = 0, \vec{v}_1 = 1, \vec{v}_2 = 2, \ldots, \vec{v}_n = 1]$$

$$P(K=\text{fin}|\vec{w}) = \frac{.16}{.27} = 59\%$$

prior_tech = .4	likelihood_tech<word, likelihood>
prior_finance = .3	likelihood_finance<word, likelihood>
prior_politics = .3	likelihood_politics<word, likelihood>

To improve our model performance we can actually do something a bit more clever with this vocabulary representation of the input message.



Supervised Learning - Naive Bayes Optimized

$$\vec{W} = [\text{'stock'}, \text{'etf'}, \text{'fire'}, \text{'stock'}, \ldots, \text{'talk'}]$$

$$= [\vec{v}_0 = 0, \vec{v}_1 = 1, \vec{v}_2 = 2, \ldots, \vec{v}_n = 1]$$

$$P(K = fin \mid \vec{w}) = \frac{.16}{.27} = 59\%$$

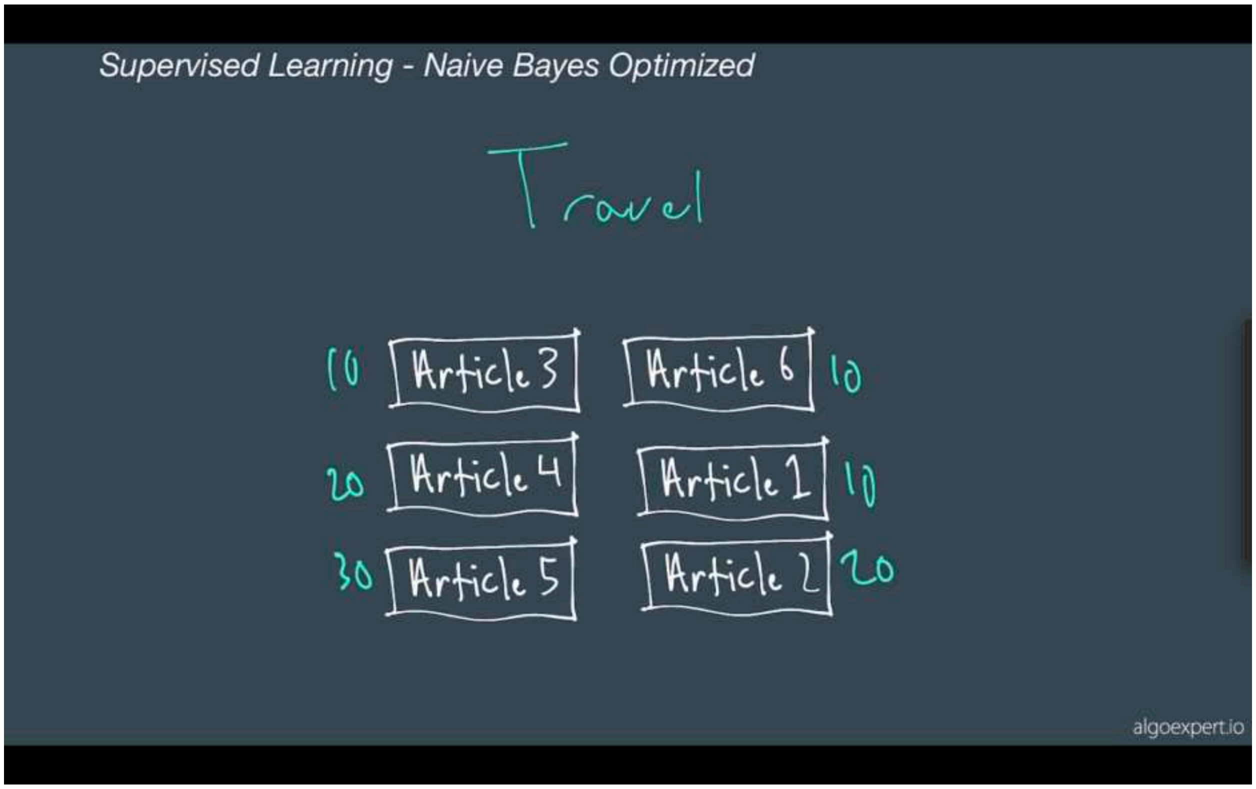
prior_tech = .4 likelihood_tech<word, likelihood>

prior_finance = .3 likelihood_finance<word, likelihood>

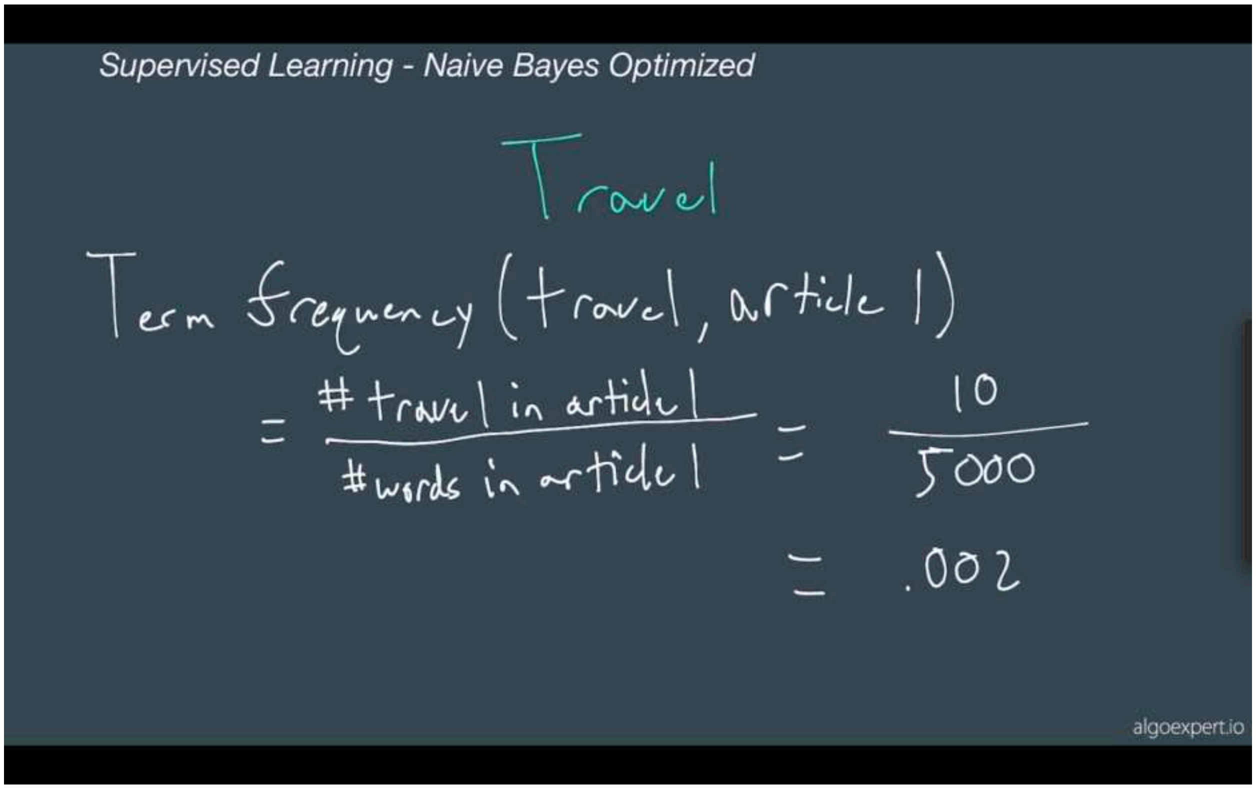
prior_politics = .3 likelihood_politics<word, likelihood>

algoexpert.io

Let's say that we had the word travel, okay? And let's say we had six articles, for instance, article three contains the word travel 10 times and so on for all the other articles, we can do something called a **term frequency calculation**.



All right, so we count the number of times the word travels appears in say article one, and we divide by the number of total words in article one, and we can get the term frequency.



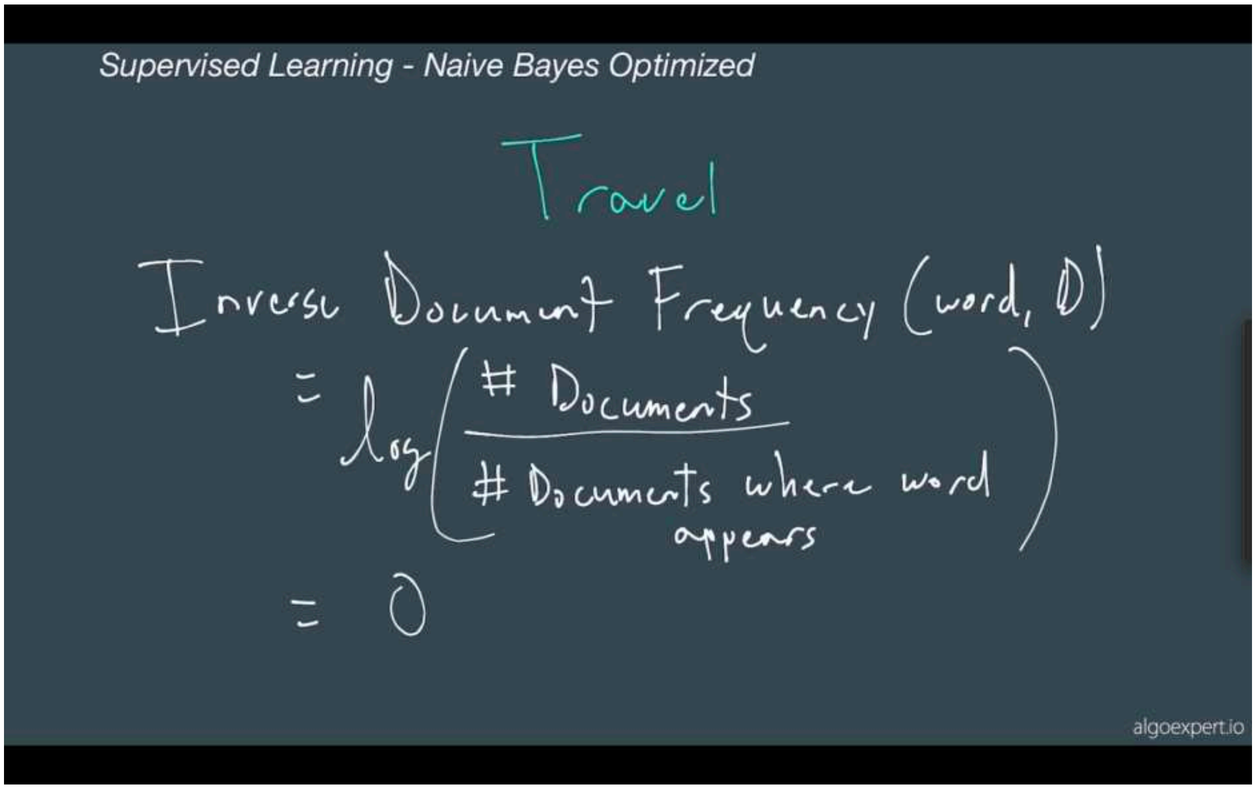
Supervised Learning - Naive Bayes Optimized

Travel

$$\text{Term frequency(travel, article 1)} = \frac{\#\text{travel in article 1}}{\#\text{words in article 1}} = \frac{10}{5000} = .002$$

If we leave that aside for just a second, then we calculate the **inverse document frequency**, which is the document here is equivalent to the articles.

So there were six articles, so six divided by the number of documents where the word travel appears, which was also six, the log of one is zero, okay? So we can calculate something called a **TF-IDF score**, which is a term frequency, inverse document frequency term.



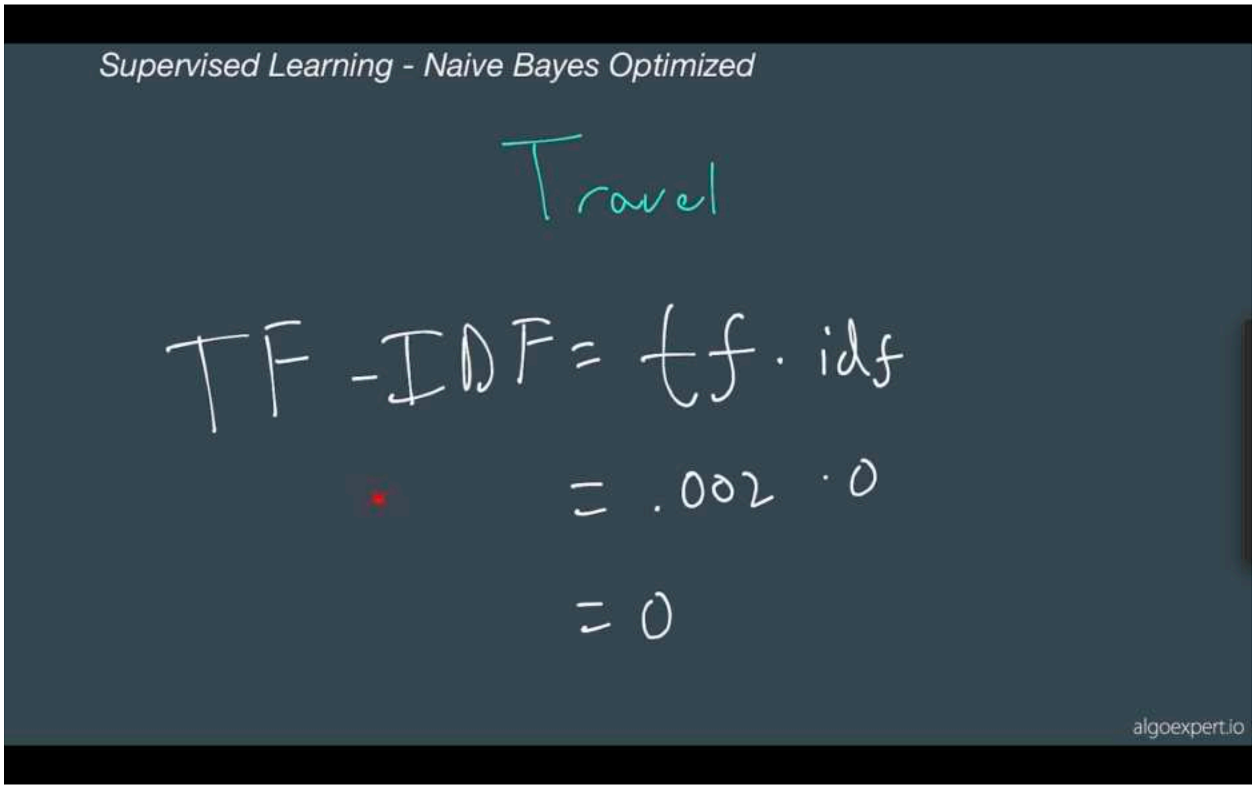
Supervised Learning - Naive Bayes Optimized

Travel

$$\text{Inverse Document Frequency}(\text{word}, D) = \log\left(\frac{\#\text{ Documents}}{\#\text{ Documents where word appears}}\right) = 0$$

algoexpert.io

We just multiply TF and IDF together. So 0.002 and zero to get a final TF-IDF of zero.



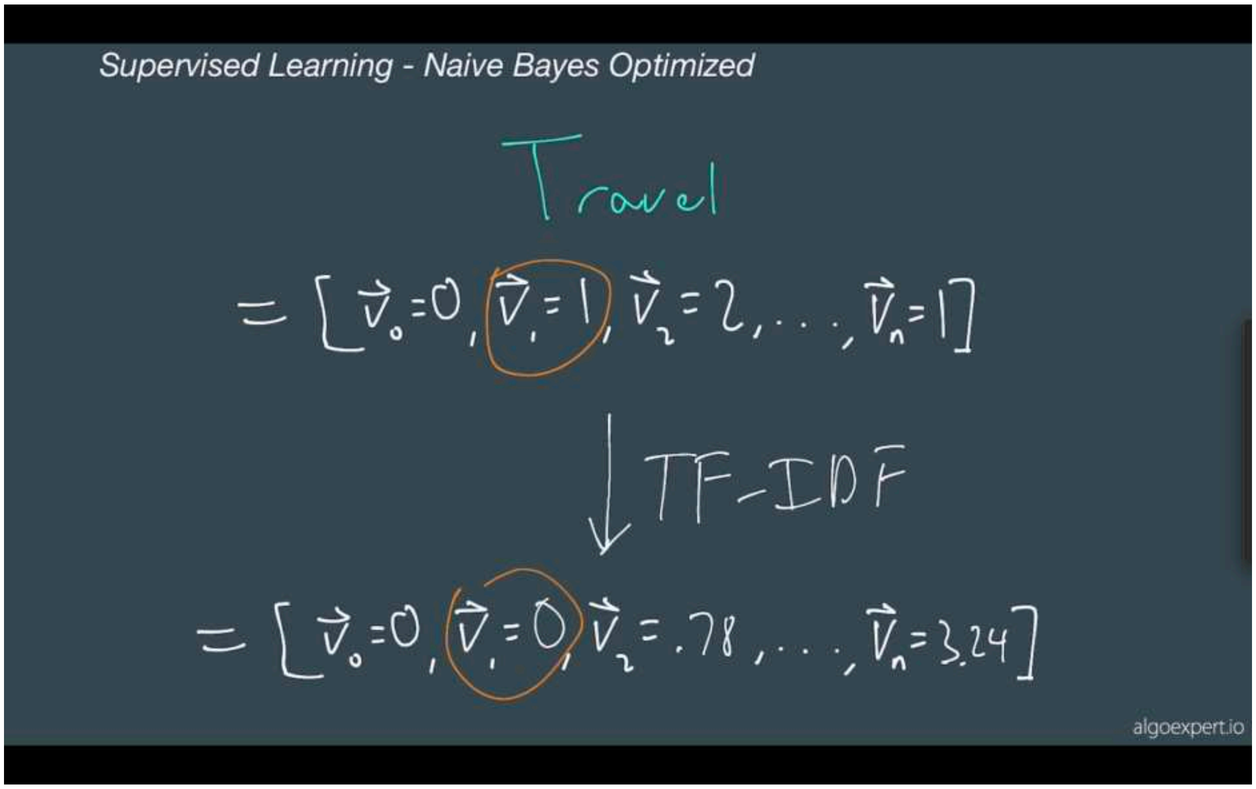
Supervised Learning - Naive Bayes Optimized

Travel

$$\text{TF-IDF} = tf \cdot idf$$
$$= .002 \cdot 0$$
$$= 0$$

So what does TF-IDF tell us actually, well, you remember the stop word list we removed a lot of the words like the, or for, because we realized that those words were so common that they weren't informative. Well, in our specific example, travel just happened to be not informative at all. Every single article contained that at least once so here at TF-IDF has zeroed out the value of travel. So it basically is a way to express **importance or unimportance of a particular word**.

So how can we use this to our advantage? Well, let's say that we have our vocabulary representation of an input message, this circled word here is travel, if we apply TF-IDF to the entire vector, what it would do, it would zero out our word of travel.


Supervised Learning - Naive Bayes Optimized

Travel

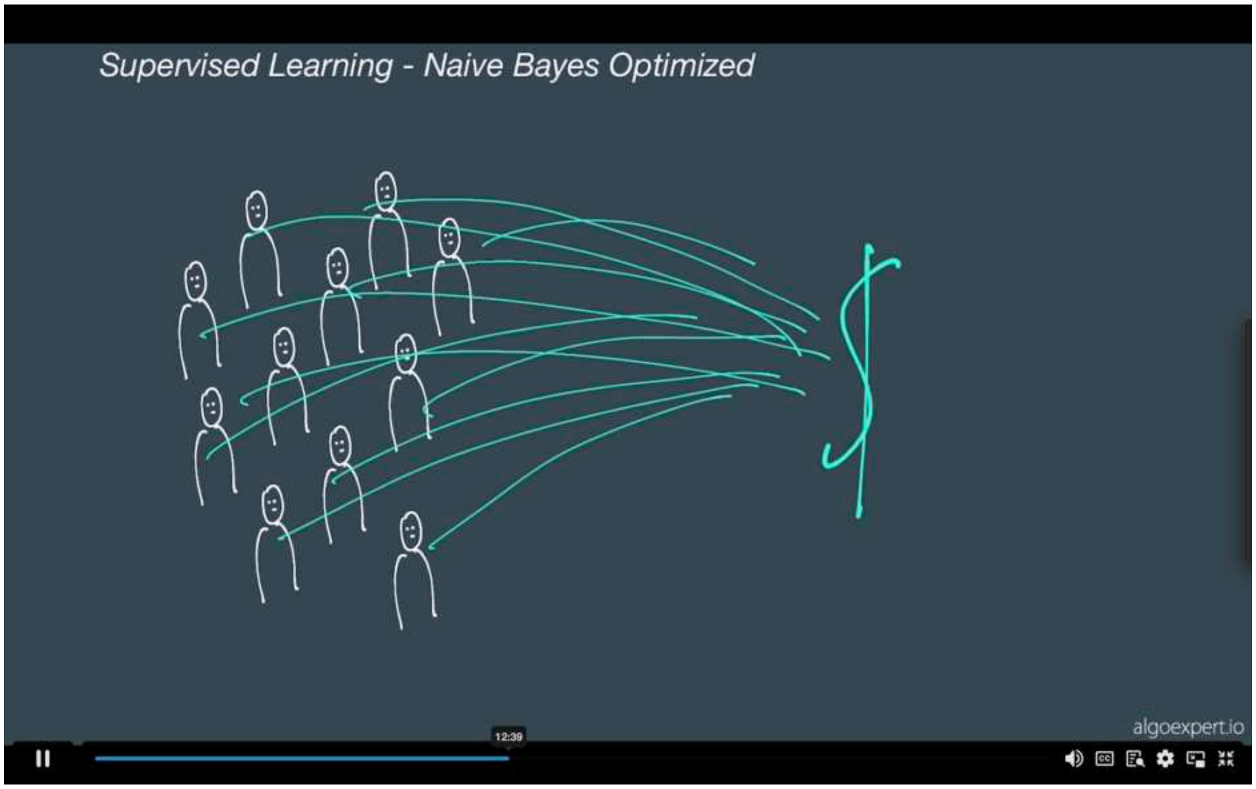
$$= [\vec{V}_0 = 0, \vec{V}_1 = 1, \vec{V}_2 = 2, \ldots, \vec{V}_n = 1]$$

$$\downarrow \text{TF-IDF}$$

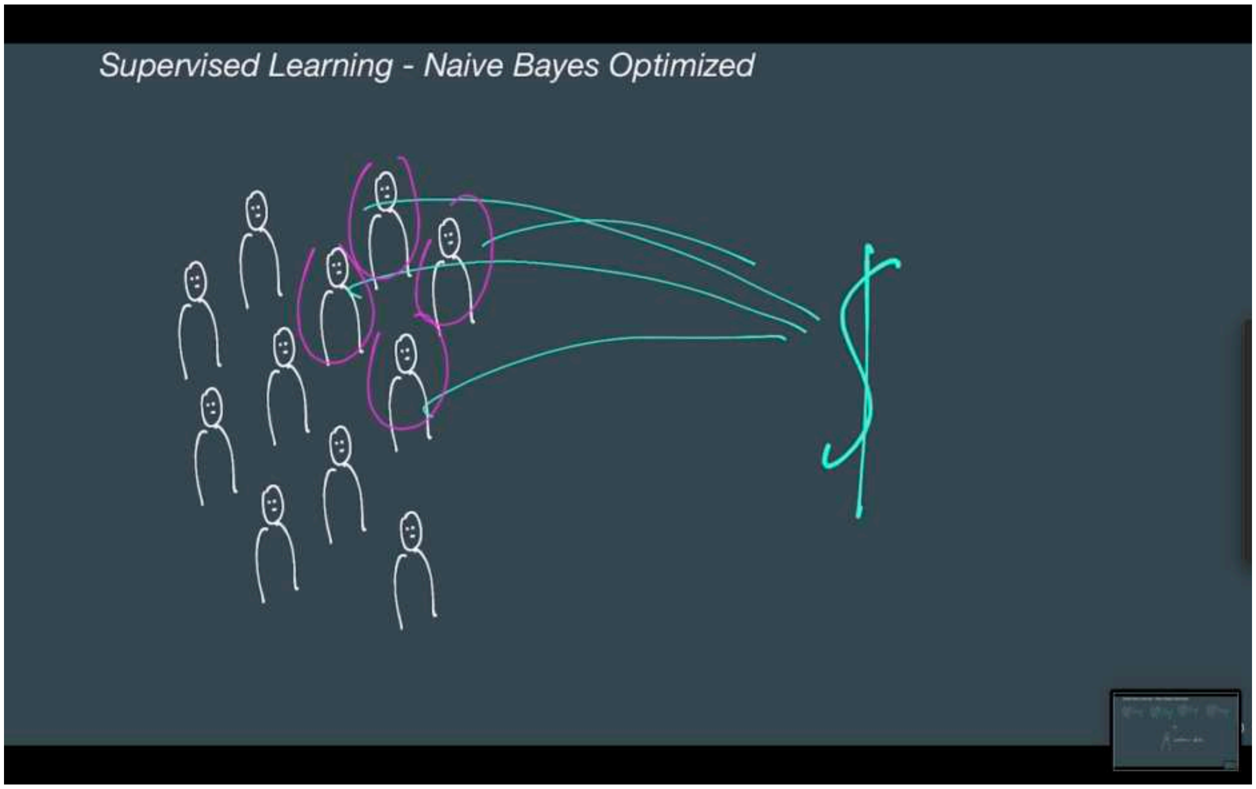
$$= [\vec{V}_0 = 0, \vec{V}_1 = 0, \vec{V}_2 = .78, \ldots, \vec{V}_n = 3.24]$$

For reference, we wouldn't change how we actually use this, we would still go through and raise every single term to the power of whatever the value is.

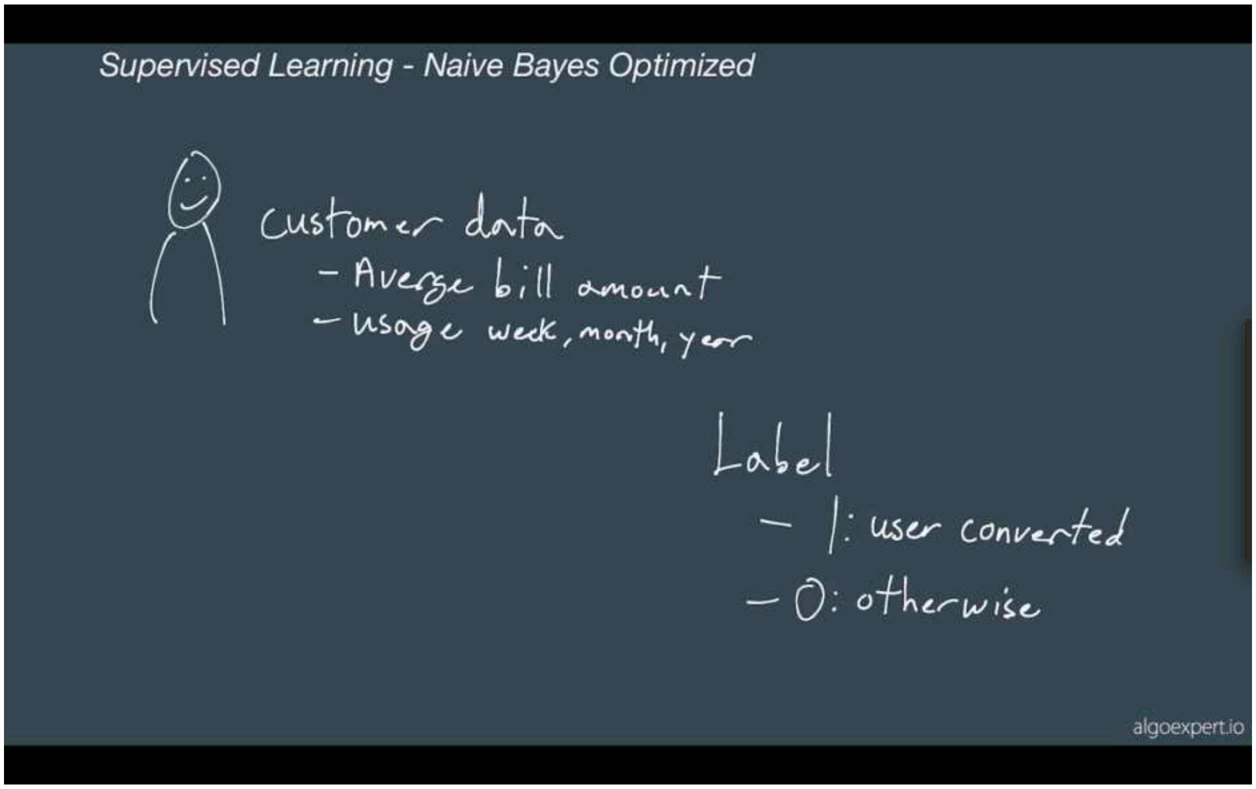
So what we've done so far is we've changed our Naive Bayes model to include multiple classes and to also represent the counts of words instead of the presence of words, which means we switched from Bernoulli to multinomial and **we've used this to categorize articles into different classes.** What this does is it allows users to create content and automatically have tags added so that people can discover the articles through search better, and we can also refine recommendation engines with these features as well.



So moving on, let's pretend that we are working for a food delivery company, **Uber Eats**, and their goal is to create a marketing campaign where they will send promotional discounts out to customers. Option one is to send money to everyone equally. So all of the customers would get, say $5 off their next order, as long as they order within the next week or so. Maybe what we can do is use past promotional data along with newer customer data to **selectively** decide who should get the promotional discounts, such that we can maximize the probability that those people will convert to what we'll call a habitual user, someone who orders at least three times a week.



In order to create a model like this, we need some features. So for the customer data, we're going to have their average bill amount, their usage by week, by month and by year, and we're gonna label these customers according to past promotions, our customer was converted to a habitual user after receiving the promotion, then the label is a one, if this customer did not convert to a habitual user, then the label would be a zero.

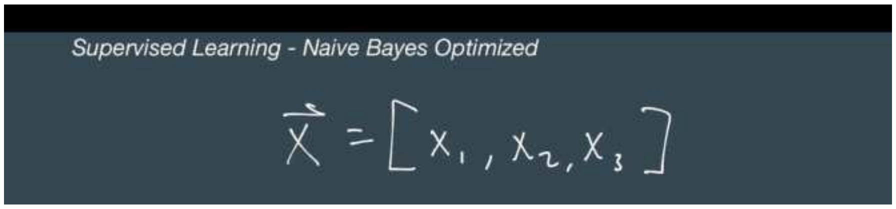

Supervised Learning - Naive Bayes Optimized

customer data
- Averge bill amount
- usage week, month, year

Label
- 1: user converted
- 0: otherwise

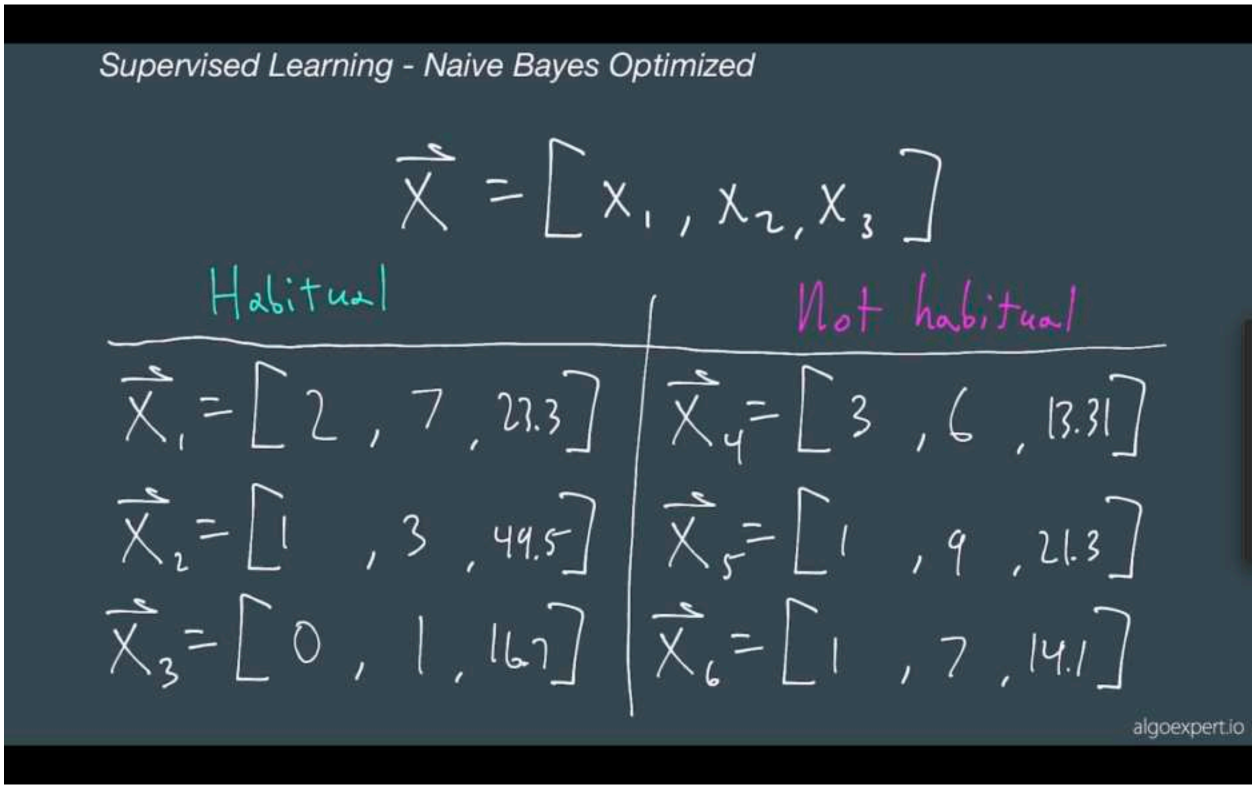
algoexpert.io

What we've done is we have a single vector X, which represents a customer. X one is going to be their usage within the past week, X two is going to be their usage within the past month, and X three is going to be their average spend amount.


Supervised Learning - Naive Bayes Optimized

$$\vec{X} = [x_1, x_2, x_3]$$

You'll notice that their yearly usage isn't included, we didn't find that feature to be very predictive, so we've stuck with these **two features** in terms of usage and this one in terms of average spend amount.

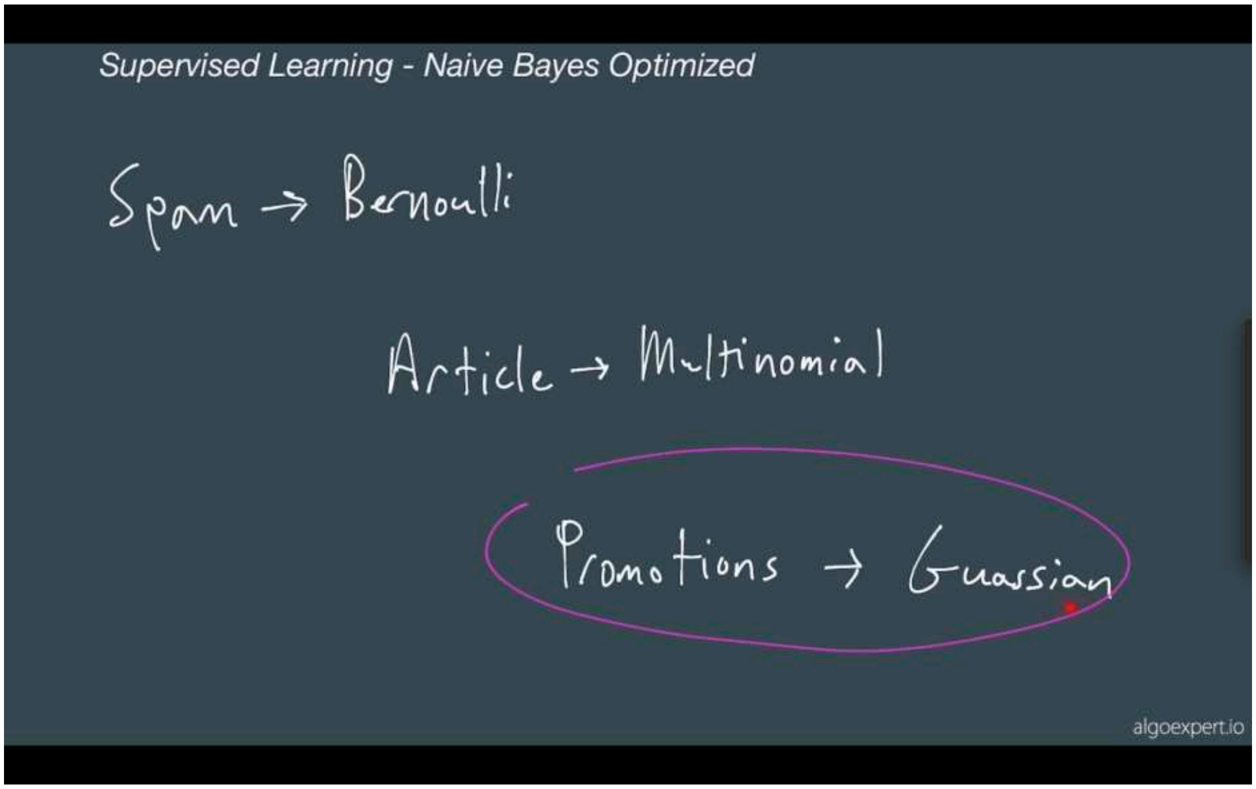

Supervised Learning - Naive Bayes Optimized

$$\vec{X} = [x_1, x_2, x_3]$$

Habitual	Not habitual
$\vec{X}_1 = [2, 7, 23.3]$	$\vec{X}_4 = [3, 6, 13.31]$
$\vec{X}_2 = [1, 3, 44.5]$	$\vec{X}_5 = [1, 9, 21.3]$
$\vec{X}_3 = [0, 1, 16.7]$	$\vec{X}_6 = [1, 7, 14.1]$

Now there's several customers. So here we have six customers X and four that customer used the service three times in the past week, they used the service six times within the past month and their average spend amount was $13 and 31 cents. Each of these customers will actually have a **label tied to them**. Customer one here, they used the service two times in the last week, seven times within the last month and their average spend amount per order was about $23. We put them in the habitual category because after receiving a promotional discount and taking advantage of it, this user converted to a habitual user within 30 days. However, this user X five, they didn't convert to a habitual user within 30 days. So we put them as a not habitual label.

How do we actually use this with a naive Bayes model? Well, for spam, we used the Bernoulli representation where we modeled the presence or absence of words.

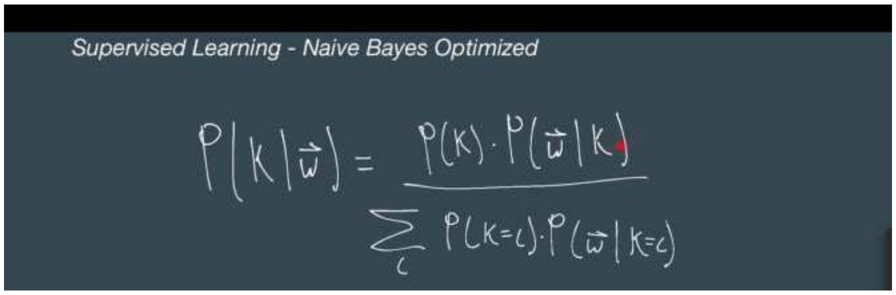

Supervised Learning - Naive Bayes Optimized

Spam → Bernoulli

Article → Multinomial

Promotions → Guassian

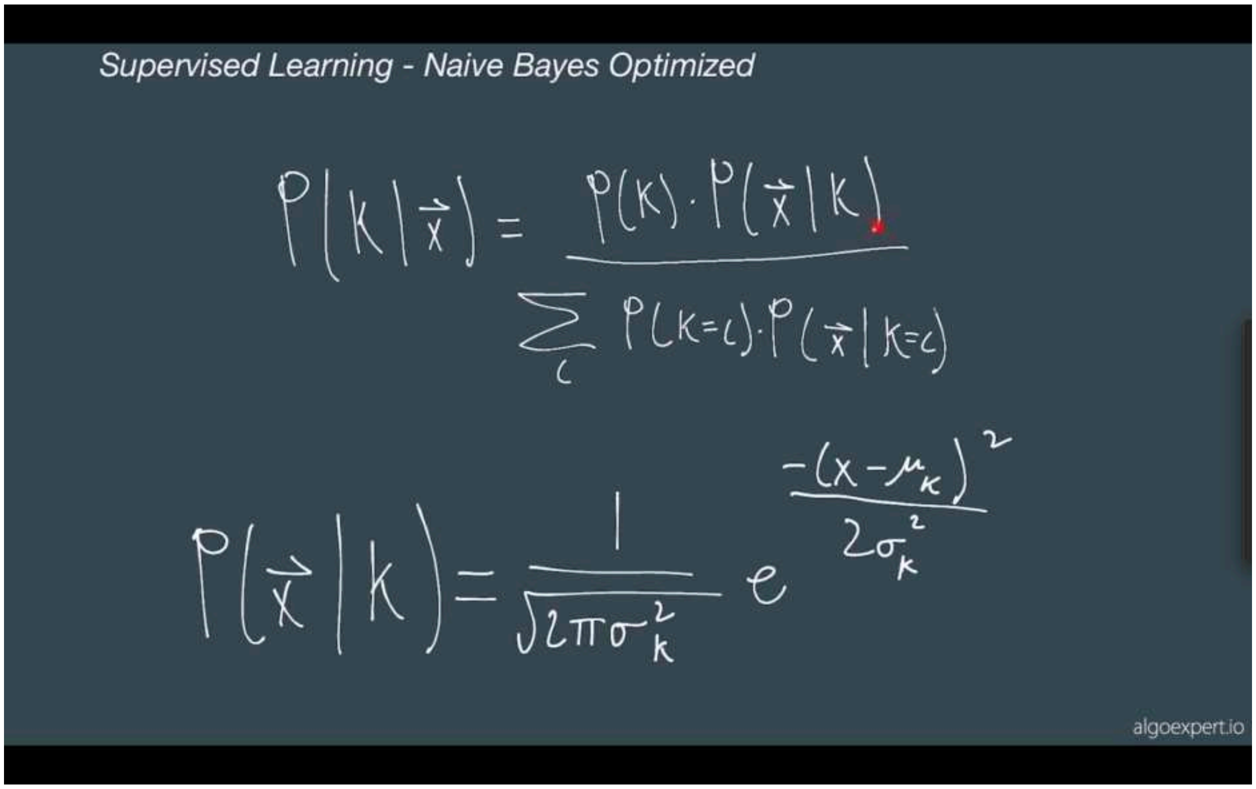
For articles, for medium we used the multinomial model, which modeled the count of the words. For promotions, we're actually going to use the **Gaussian model**.


Supervised Learning - Naive Bayes Optimized

$$P(K|\vec{w}) = \frac{P(K)\cdot P(\vec{w}|K)}{\sum_{c} P(K=c)\cdot P(\vec{w}|K=c)}$$

Now let's take a look at exactly what that is. Here's our typical model that where W is the word vector. We're going to replace that with X to represent the customer, and these likelihoods are going to be the only thing that changes.

With Bernoulli, these likelihoods were one thing, with multinomial, these likelihoods were another, and with Gaussian these likelihoods will be yet another, but everything else will be more or less the same. Let's take a look at what exactly this is.



Supervised Learning - Naive Bayes Optimized

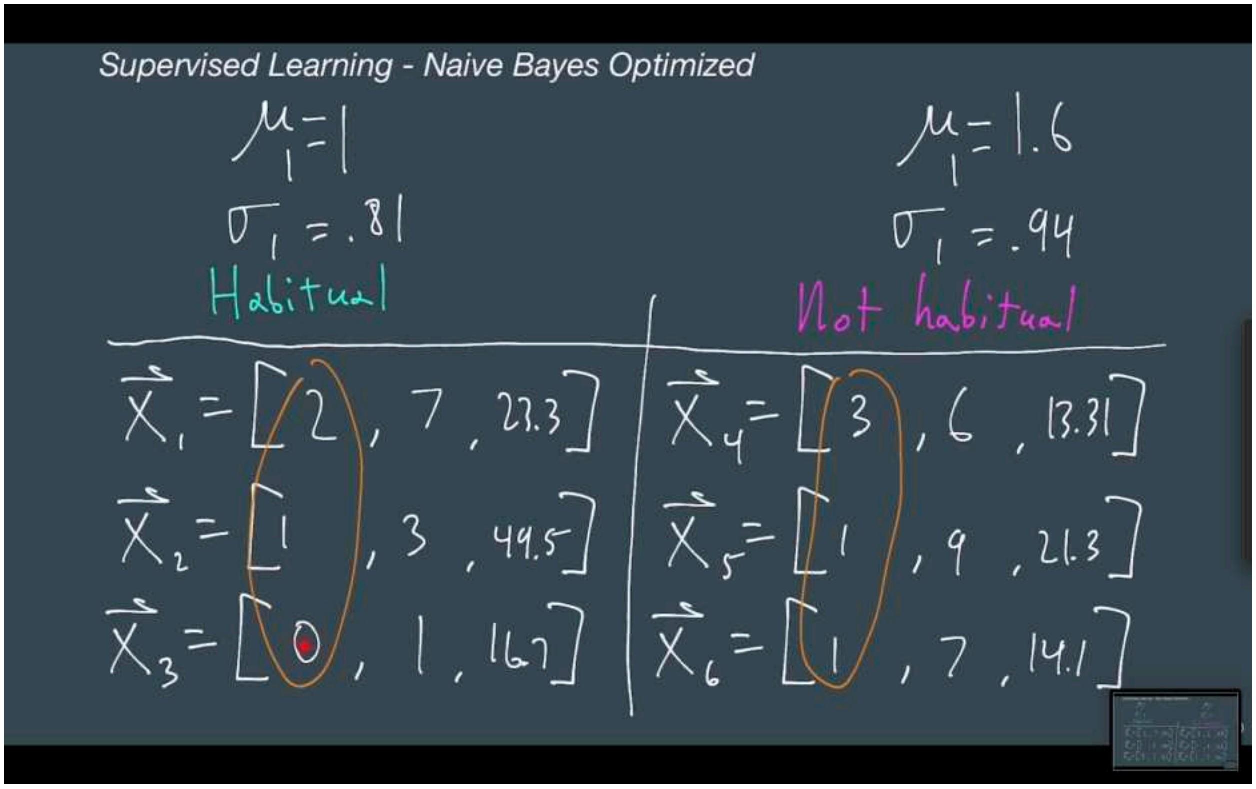
$$P(k \mid \vec{x}) = \frac{P(k) \cdot P(\vec{x} \mid k)}{\sum_{c} P(k=c) \cdot P(\vec{x} \mid k=c)}$$

$$P(\vec{x} \mid k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x-\mu_k)^2}{2\sigma_k^2}}$$

algoexpert.io

We actually only care about the **means** and the **standard deviations** of our customary data. Let's take a look at how to calculate this and see how this all connects.

We have our habitual customers and non-habitual customers, now, what we're gonna do is we're going to define a mean and a standard deviation for the first set of features per class.



Supervised Learning - Naive Bayes Optimized

$\mu_1 = 1$

$\sigma_1 = .81$

Habitual

$\vec{X}_1 = [2, 7, 23.3]$

$\vec{X}_2 = [1, 3, 44.5]$

$\vec{X}_3 = [0, 1, 167]$

$\mu_1 = 1.6$

$\sigma_1 = .94$

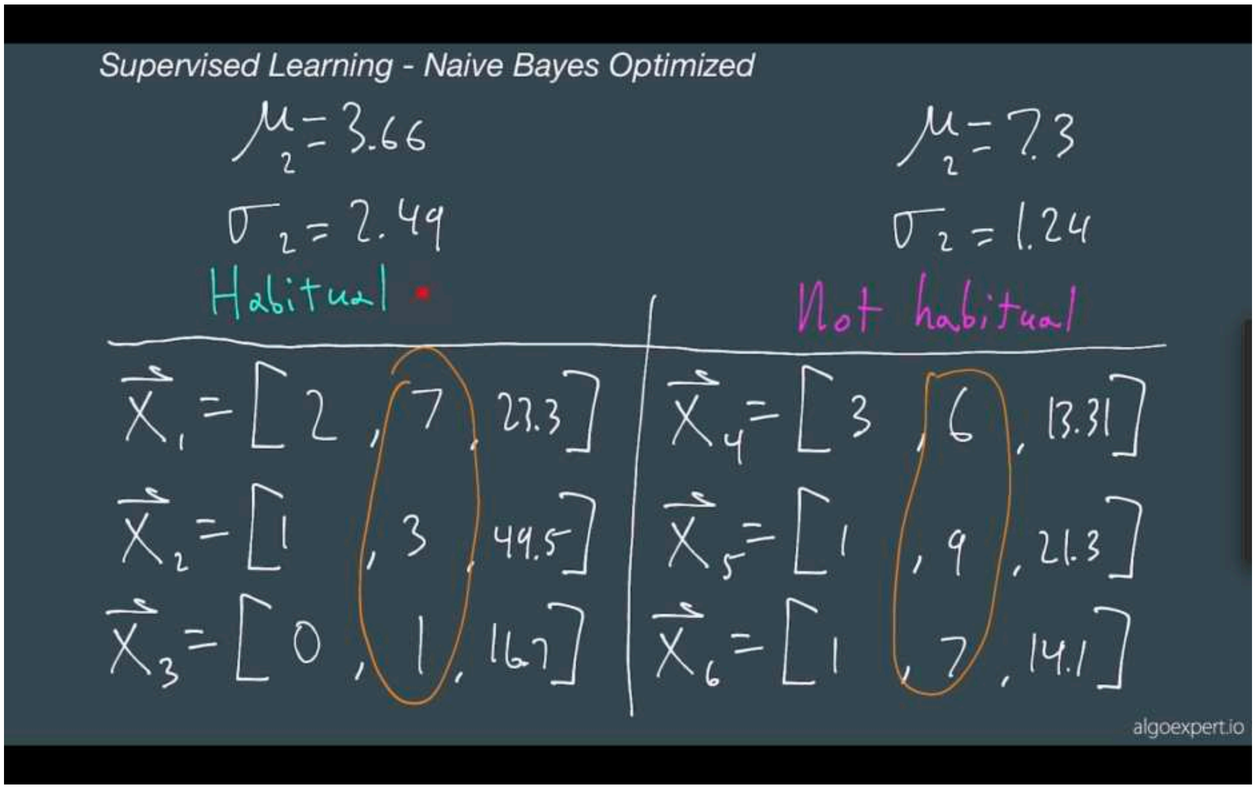
Not habitual

$\vec{X}_4 = [3, 6, 13.31]$

$\vec{X}_5 = [1, 9, 21.3]$

$\vec{X}_6 = [1, 7, 14.1]$

So on average, the habitual users had one order in the past week. Their standard deviation was 0.81. The non-habitual users on average placed 1.6 orders within the past week and their standard deviation was 0.94.

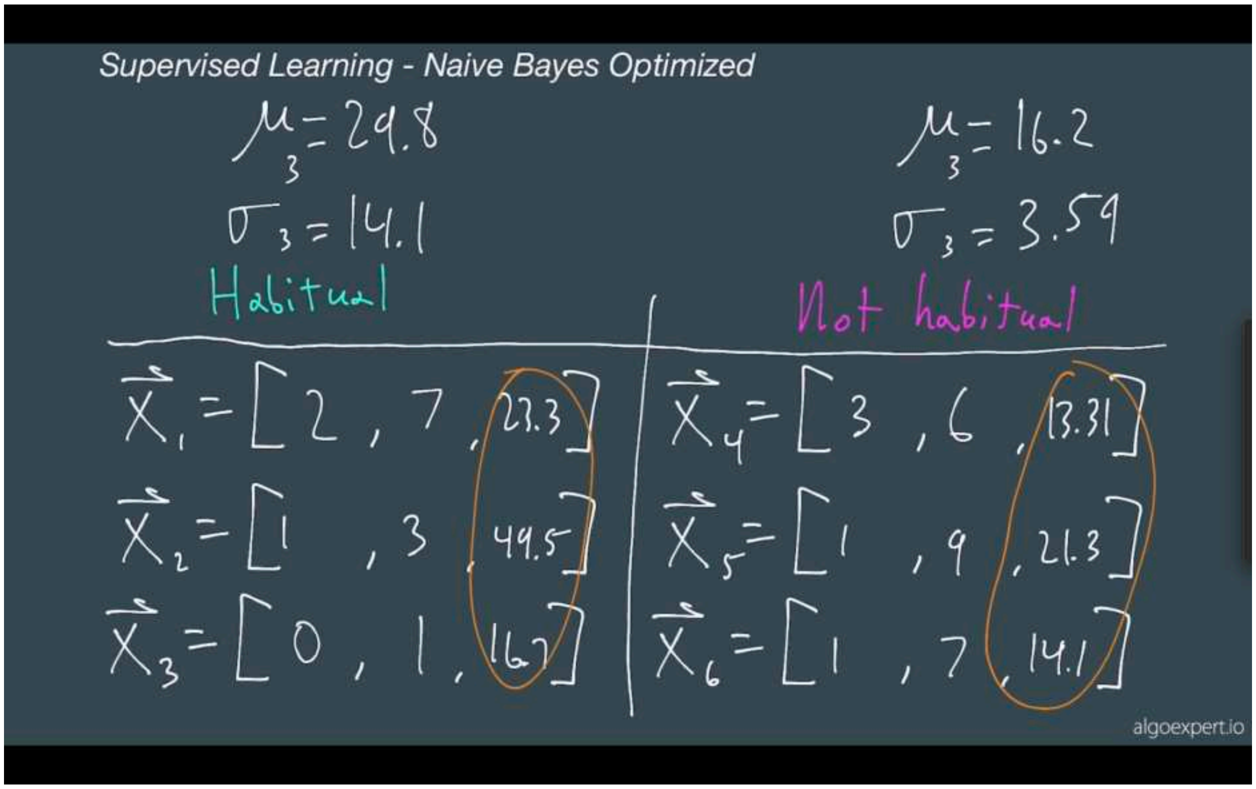


Supervised Learning - Naive Bayes Optimized

Habitual	Not habitual
$\mu_2 = 3.66$	$\mu_2 = 7.3$
$\sigma_2 = 2.49$	$\sigma_2 = 1.24$
$\vec{X}_1 = [2, 7, 23.3]$	$\vec{X}_4 = [3, 6, 13.31]$
$\vec{X}_2 = [1, 3, 49.5]$	$\vec{X}_5 = [1, 9, 21.3]$
$\vec{X}_3 = [0, 1, 16.7]$	$\vec{X}_6 = [1, 7, 14.1]$

We can continue this pattern, for instance, the habitual users, the users who did convert to habitual users after receiving a promotion, on average, they placed 3.66 orders within the past month.

For the final feature, the non-habitual users on average spent $16 and 20 cents per order. These are the means for feature one, feature two, feature three, these are the standard deviations, and we've separated them out by class.



Supervised Learning - Naive Bayes Optimized

$\mu_3 = 29.8$

$\sigma_3 = 14.1$

Habitual

$\vec{X}_1 = [2, 7, 23.3]$

$\vec{X}_2 = [1, 3, 49.5]$

$\vec{X}_3 = [0, 1, 16.7]$

$\mu_3 = 16.2$

$\sigma_3 = 3.59$

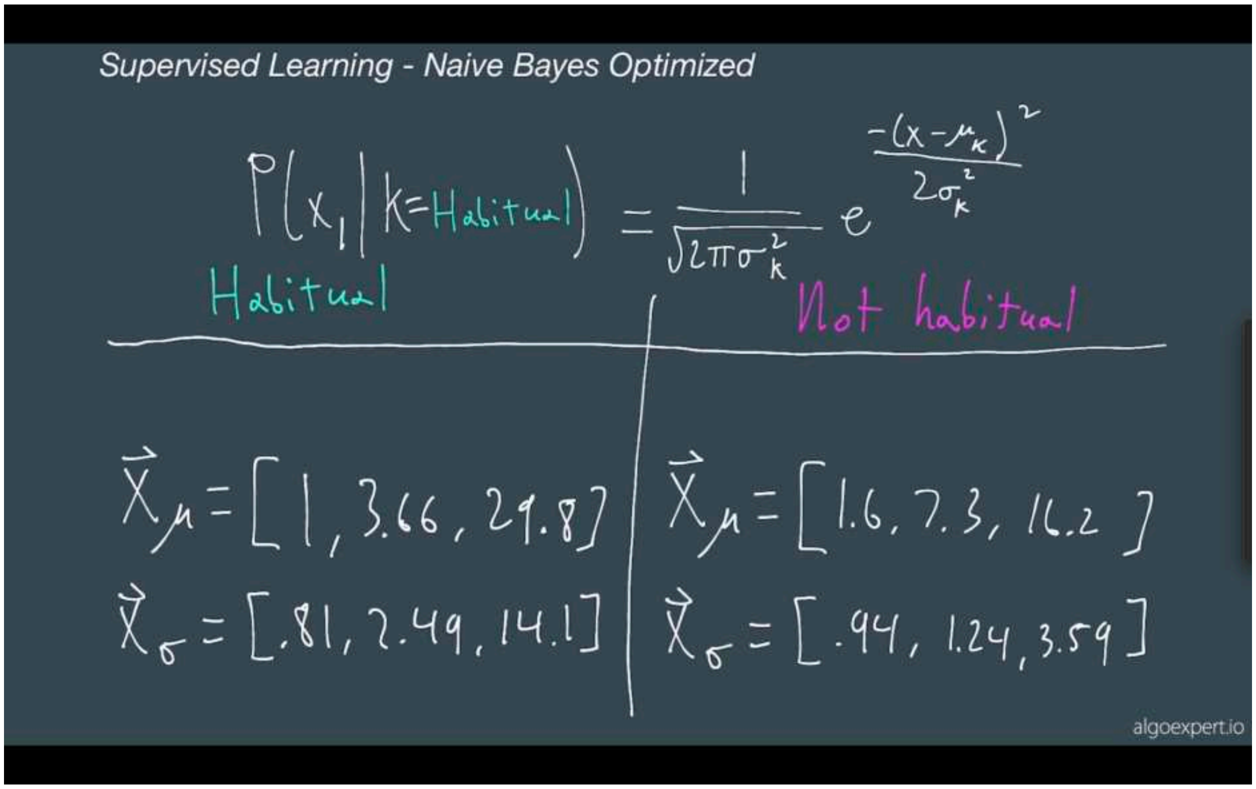
Not habitual

$\vec{X}_4 = [3, 6, 13.31]$

$\vec{X}_5 = [1, 9, 21.3]$

$\vec{X}_6 = [1, 7, 14.1]$

Okay, so now let's plug this into our model. This is a likelihood and what we're asking is what's the probability of seeing a particular feature in feature one, given that the user converted to a habitual user after receiving that promotion?



Supervised Learning - Naive Bayes Optimized

$$P(x_1 \mid k=\text{Habitual}) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x-\mu_k)^2}{2\sigma_k^2}}$$

Habitual:

$\vec{X}_\mu = [1, 3.66, 29.8]$

$\vec{X}_\sigma = [.81, 2.49, 14.1]$

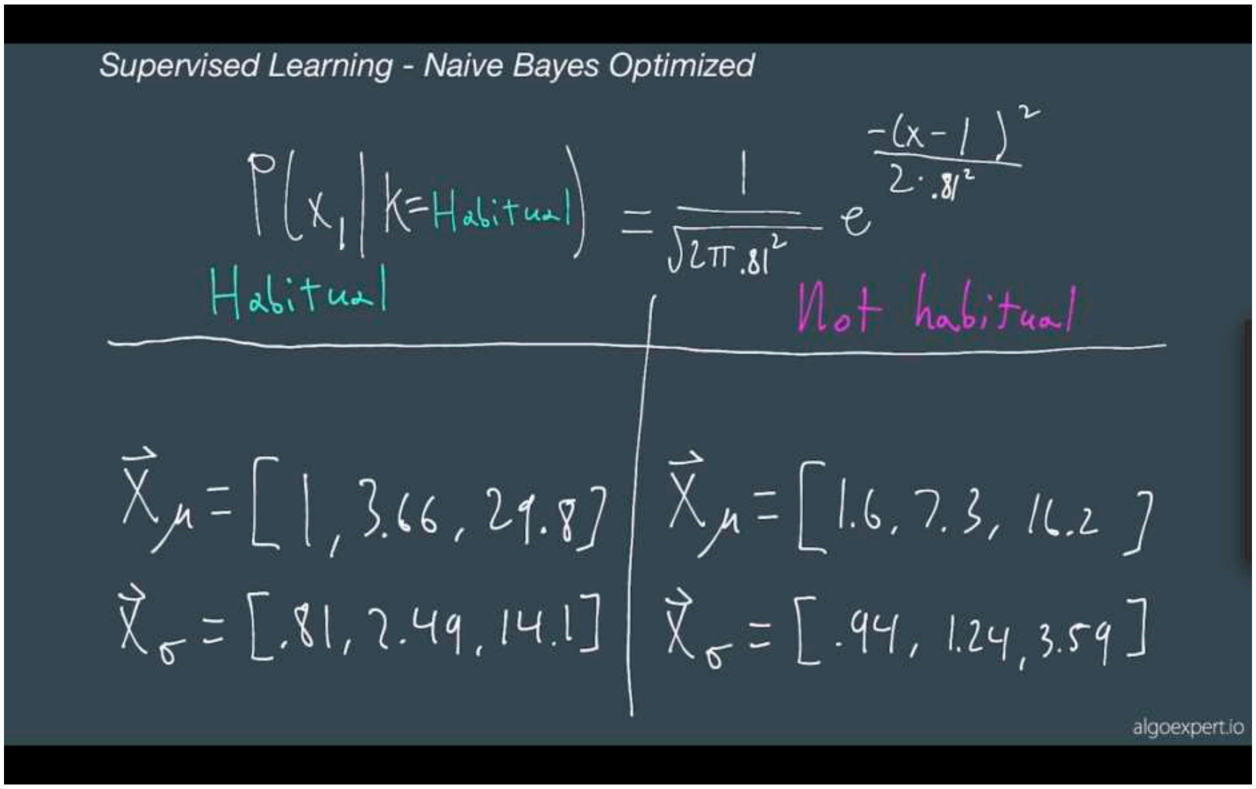
Not habitual:

$\vec{X}_\mu = [1.6, 7.3, 16.2]$

$\vec{X}_\sigma = [.94, 1.24, 3.59]$

And this is the formula that we need to solve, we'll begin by substituting the standard deviations and the means for the habitual class. So here we have 0.81, 0.81, 0.81 and of course, one.

All we did is we substituted in the standard deviations for the habitual class and the means for the habitual class for feature one. Let's plug in an example, to see how we can calculate this likelihood.



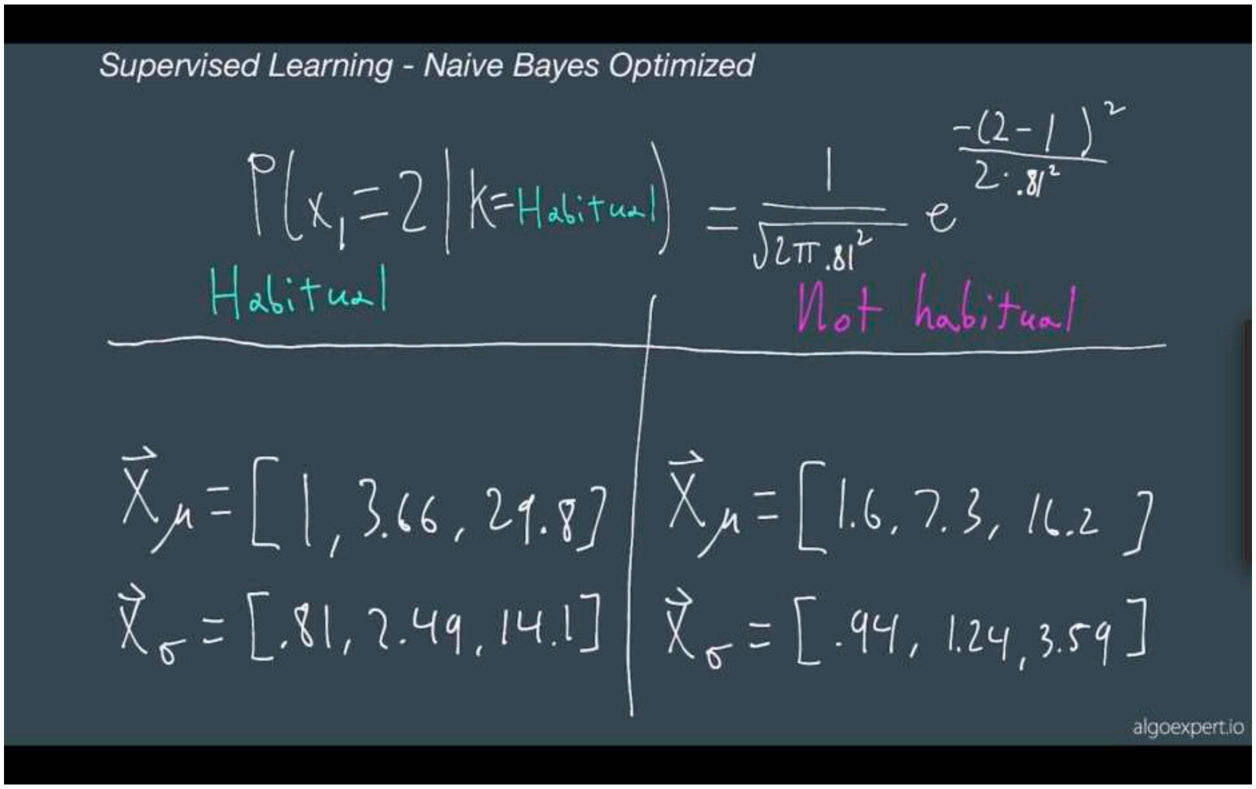
Supervised Learning - Naive Bayes Optimized

$$P(x_1 \mid k = \text{Habitual}) = \frac{1}{\sqrt{2\pi \cdot .81^2}} e^{\frac{-(x-1)^2}{2 \cdot .81^2}}$$

Habitual	Not habitual
$\vec{X}_\mu = [1, 3.66, 29.8]$	$\vec{X}_\mu = [1.6, 7.3, 16.2]$
$\vec{X}_\sigma = [.81, 2.49, 14.1]$	$\vec{X}_\sigma = [.94, 1.24, 3.59]$

algoexpert.io

We wanna know the probability that a customer has ordered twice in the last week, and since it's feature one, given that they will convert to a habitual user. And all that means is we plug into for X and then we solve.

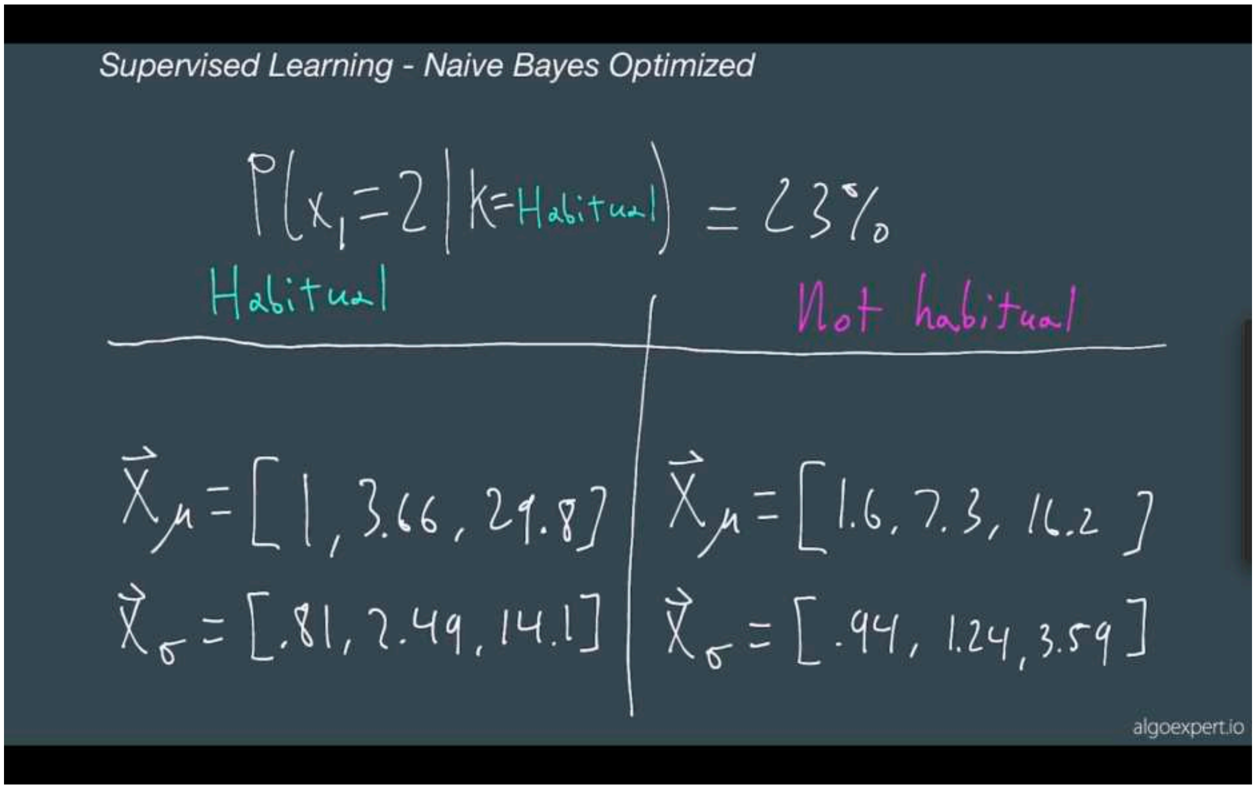


Supervised Learning - Naive Bayes Optimized

$$P(x_1 = 2 \mid k = \text{Habitual}) = \frac{1}{\sqrt{2\pi \cdot .81^2}} e^{\frac{-(2-1)^2}{2 \cdot .81^2}}$$

Habitual	Not habitual
$\vec{X}_\mu = [1, 3.66, 29.8]$	$\vec{X}_\mu = [1.6, 7.3, 16.2]$
$\vec{X}_\sigma = [.81, 2.49, 14.1]$	$\vec{X}_\sigma = [.94, 1.24, 3.59]$

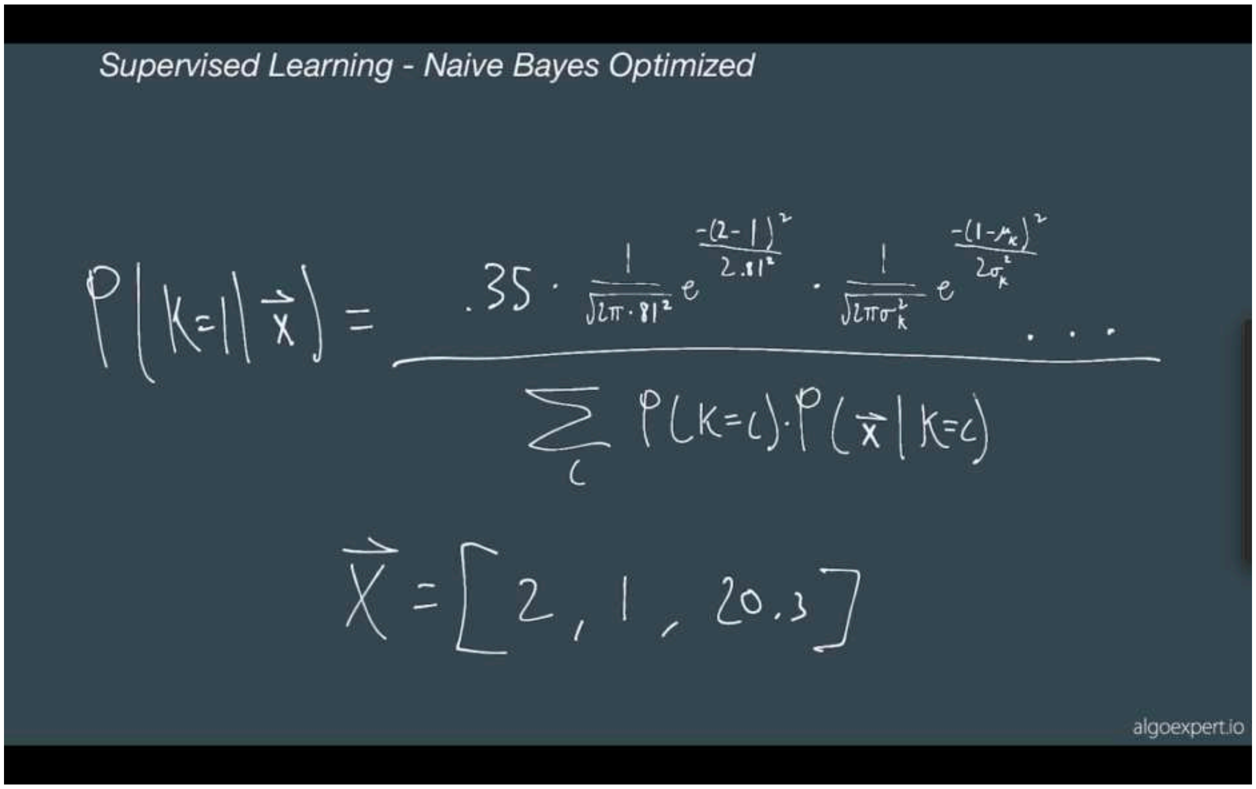
So the likelihood that a customer orders twice in the last week, given that they will convert to a habitual user is 23%.


Supervised Learning - Naive Bayes Optimized

$P(x_1 = 2 \mid k = \text{Habitual}) = 23\%$

Habitual	Not habitual
$\vec{X}_\mu = [1, 3.66, 29.8]$	$\vec{X}_\mu = [1.6, 7.3, 16.2]$
$\vec{X}_\sigma = [.81, 2.49, 14.1]$	$\vec{X}_\sigma = [.94, 1.24, 3.59]$

Okay, so now we know how to calculate the likelihoods, we just have to do that for every feature. So we'll do that for X one, we'll do that for X two, and likewise, every single standard deviation and mean will be different for those features as well.

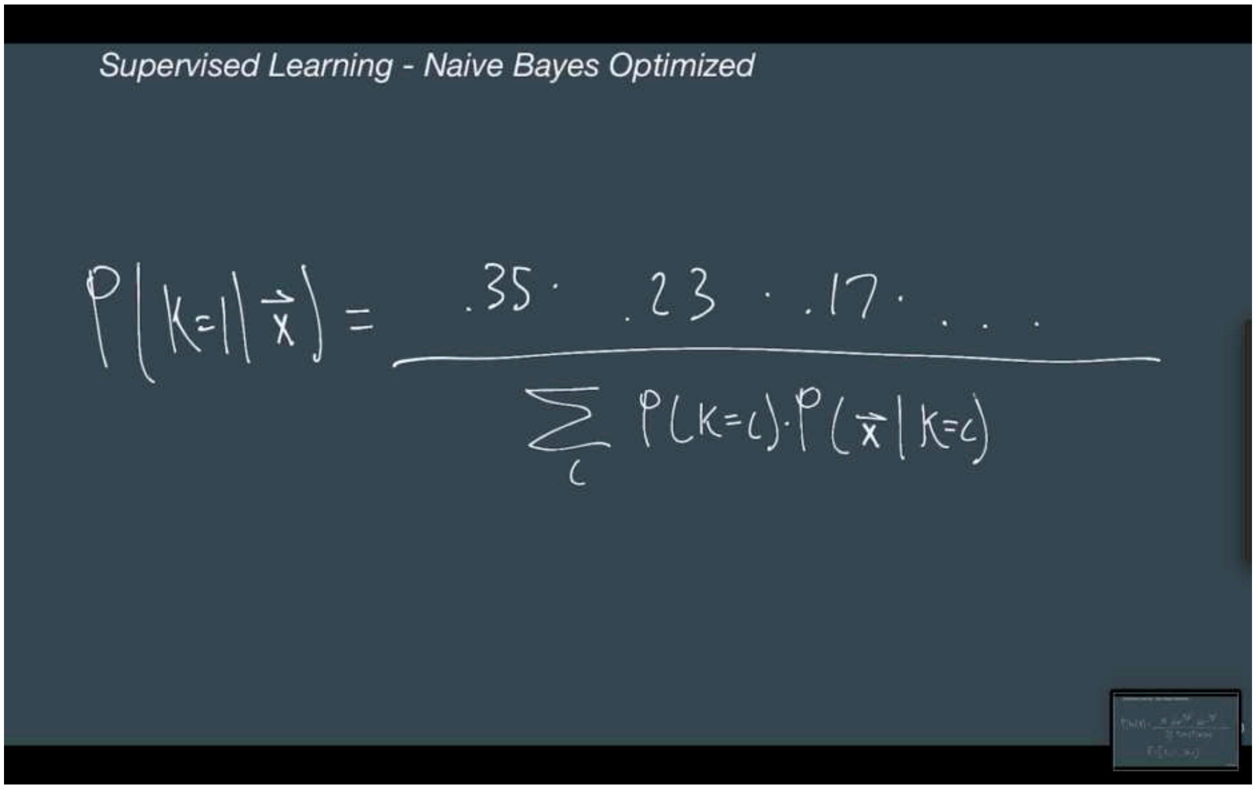


Supervised Learning - Naive Bayes Optimized

$$P(k=1 \mid \vec{x}) = \frac{.35 \cdot \frac{1}{\sqrt{2\pi \cdot .81^2}} e^{\frac{-(2-1)^2}{2 \cdot .81^2}} \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(1-\mu_k)^2}{2\sigma_k^2}} \cdots}{\sum_c P(k=c) \cdot P(\vec{x} \mid k=c)}$$

$$\vec{X} = [2, 1, 20.3]$$

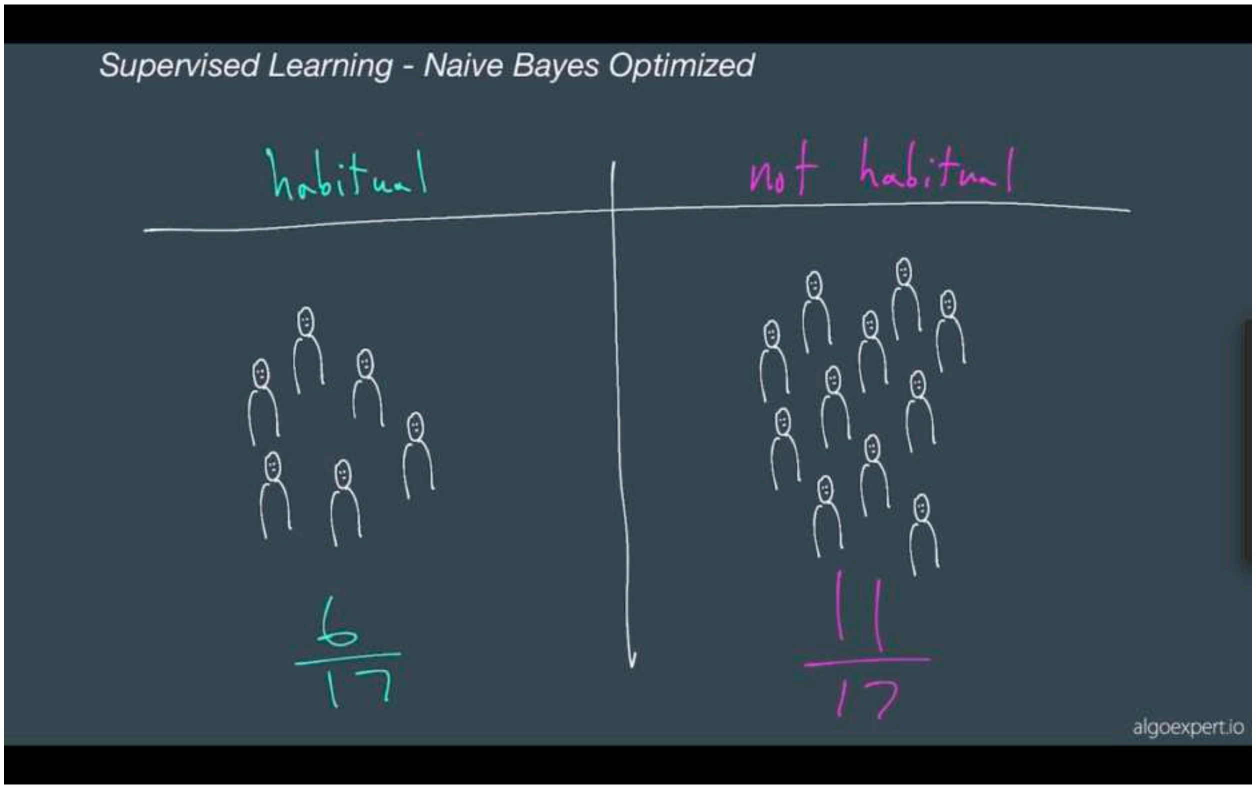
algoexpert.io



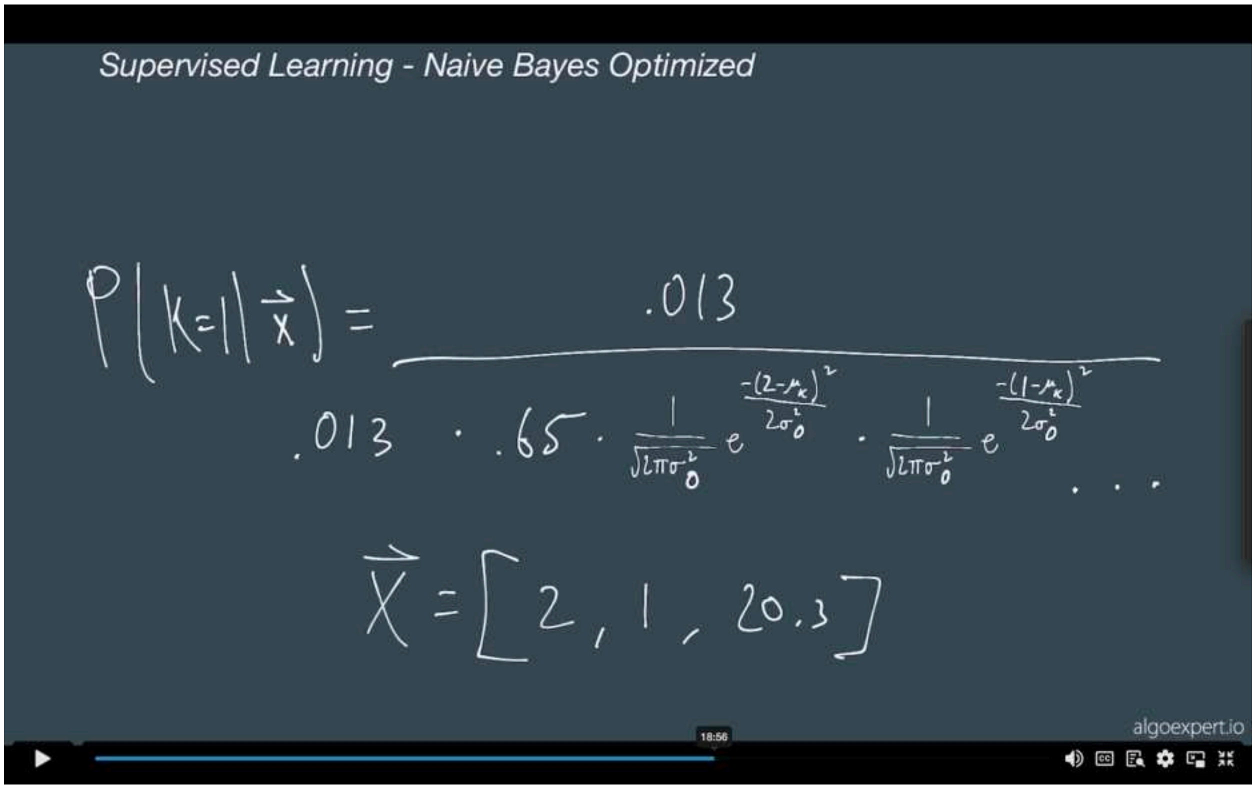
Supervised Learning - Naive Bayes Optimized

$$P(k=1 \mid \vec{x}) = \frac{.35 \cdot .23 \cdot .17 \cdot \ldots}{\sum_c P(k=c) \cdot P(\vec{x} \mid k=c)}$$

Let's go ahead quickly and see how we calculate the prior. Prior is very easy to calculate we just count the number of habitual users, and we count the number of non-habitual users, and we take the ratio of them. So six out of 17 of the total users are habitual, so that's the prior of habitual users six over 17, and the prior of a non-habitual user is 11 over 17.



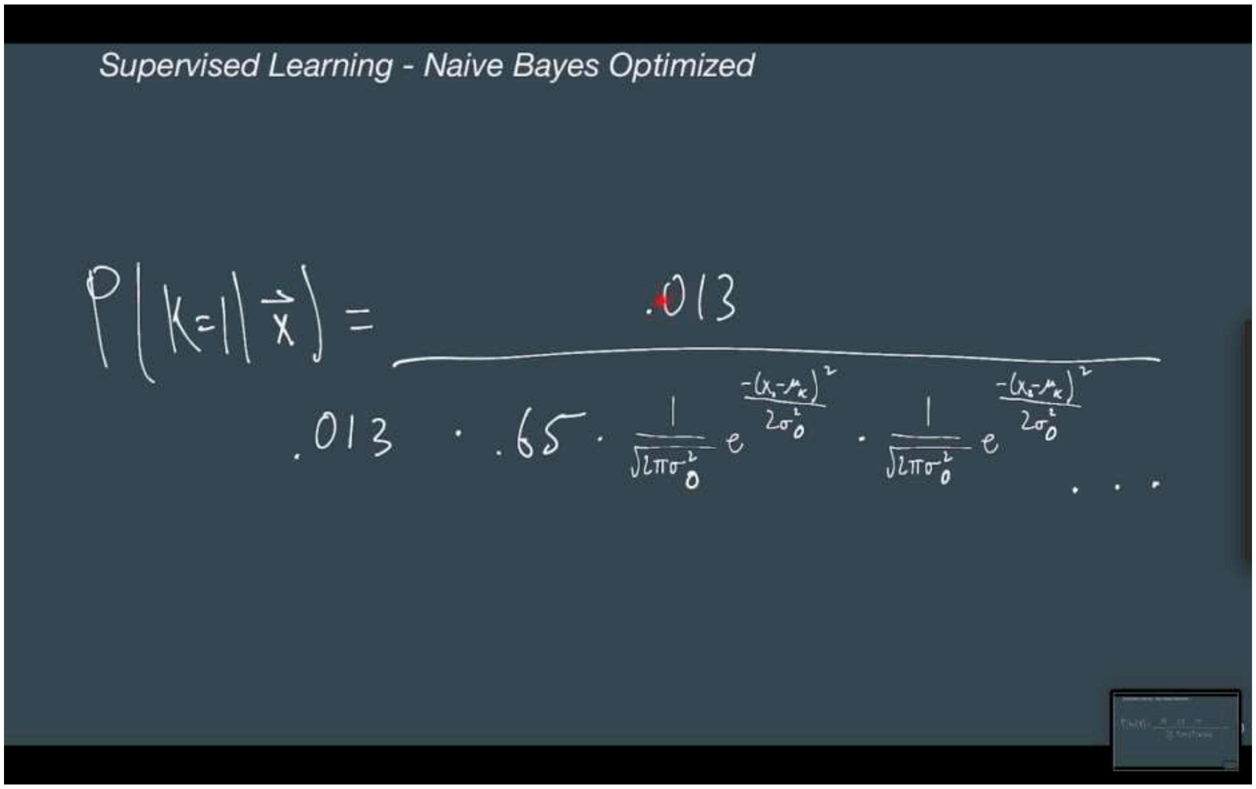
So we can plug that in because right now we wanna solve for K equals to one, here one is going to be a habitual user and zero will be a non-habitual user. And let's say that we want to keep this example, so we can plug those numbers in, solve that, in the numerator, we've already solved that 0.013, so in the denominator will also be the same.



Supervised Learning - Naive Bayes Optimized

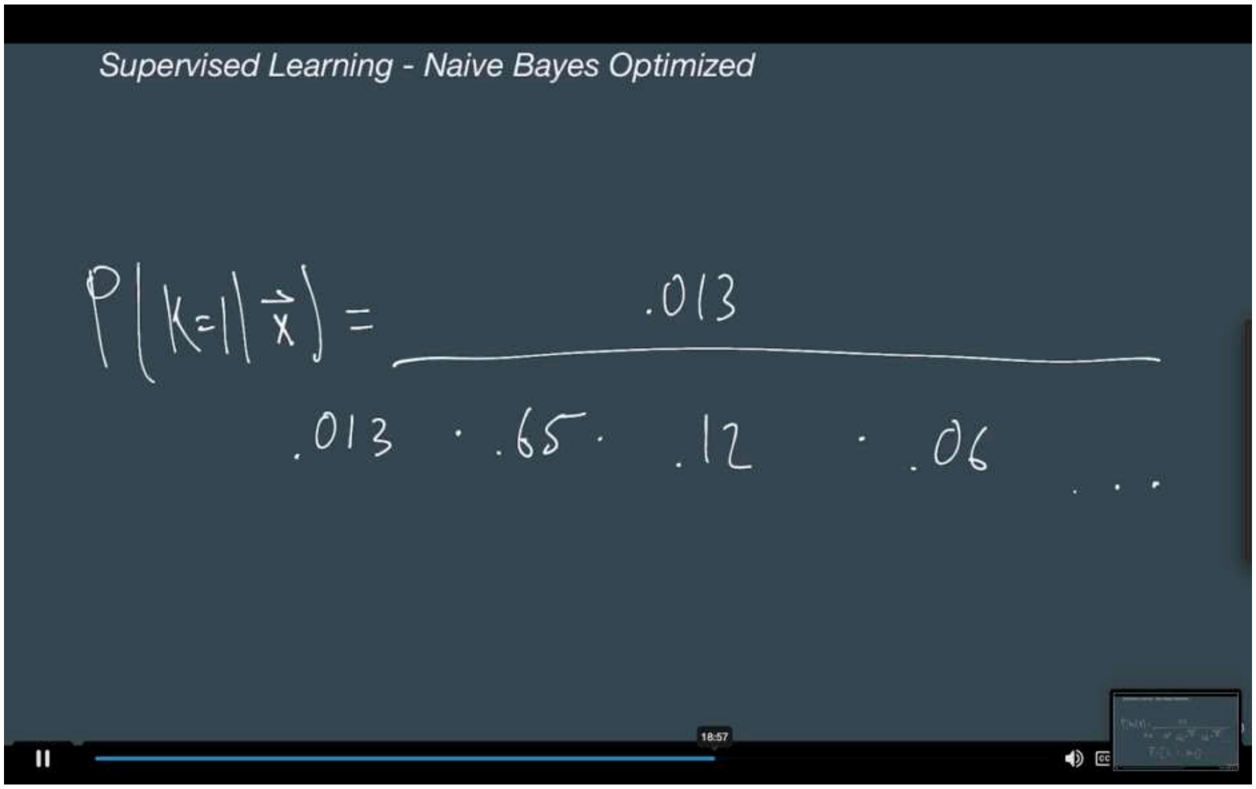
$$P(K=1|\vec{x}) = \frac{.013}{.013 \cdot .65 \cdot \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{\frac{-(2-\mu_K)^2}{2\sigma_0^2}} \cdot \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{\frac{-(1-\mu_K)^2}{2\sigma_0^2}} \cdots}$$

$$\vec{X} = [2, 1, 20.3]$$



Supervised Learning - Naive Bayes Optimized

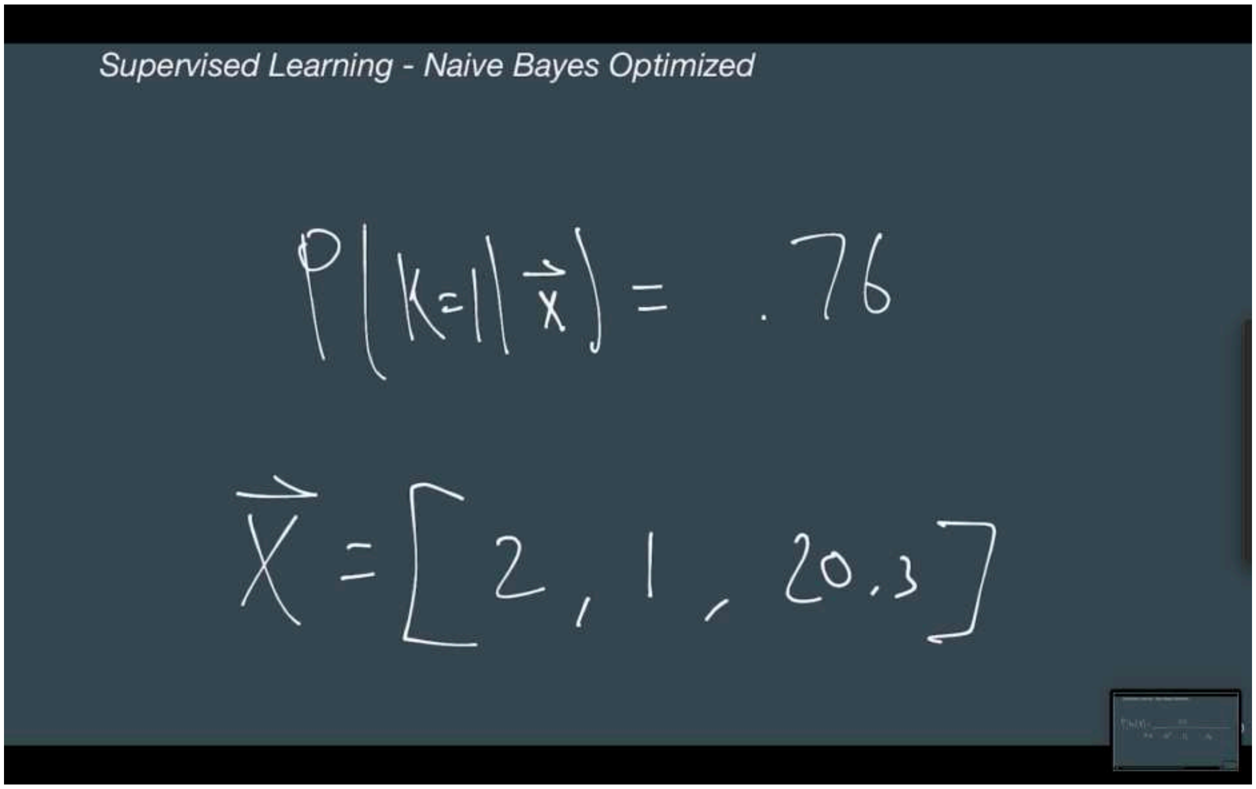
$$P(K=1|\vec{x}) = \frac{.013}{.013 \cdot .65 \cdot \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{\frac{-(x_1-\mu_K)^2}{2\sigma_0^2}} \cdot \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{\frac{-(x_2-\mu_K)^2}{2\sigma_0^2}} \cdots}$$



Supervised Learning - Naive Bayes Optimized

$$P(k=1 \mid \vec{x}) = \frac{.013}{.013 \cdot .65 \cdot .12 \cdot .06 \cdots}$$

Again, this is our example. We plug in those values, we get out a number and the probability that this user given these features will convert to a habitual user after receiving the promotion is 76%.

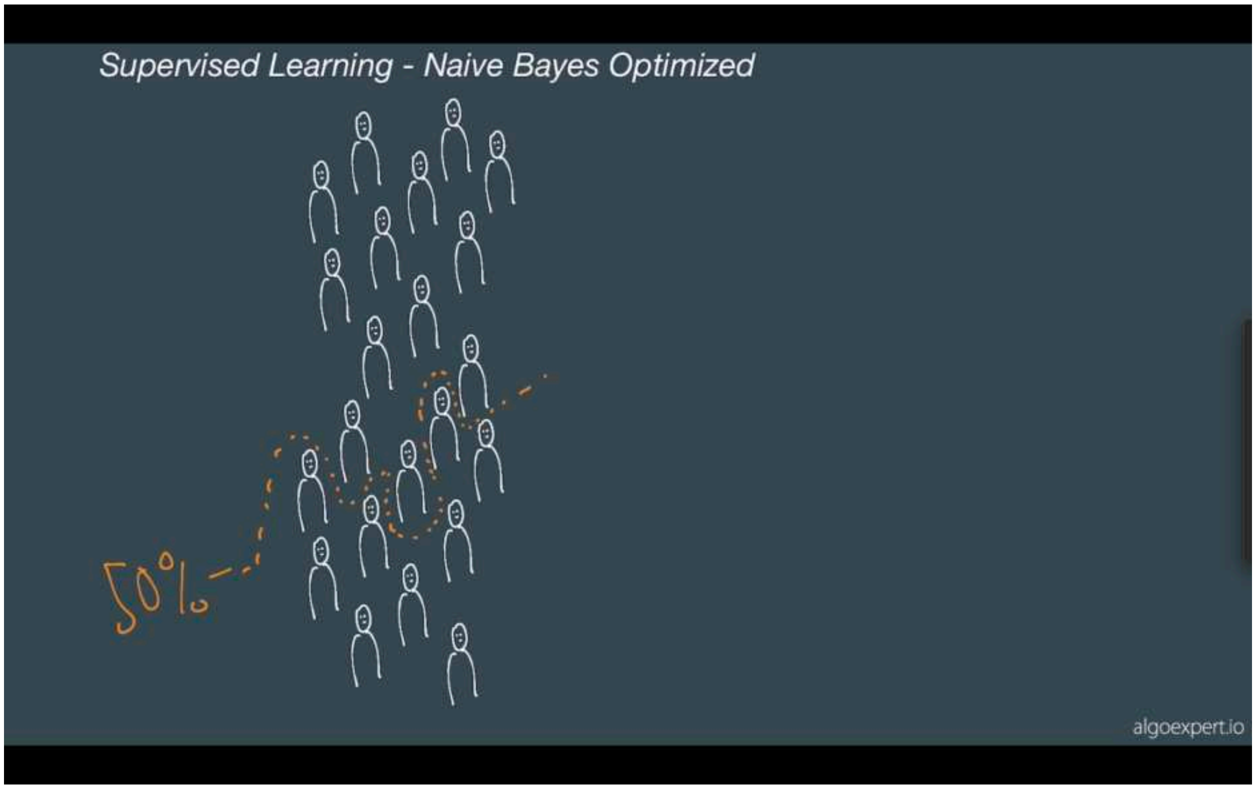


Supervised Learning - Naive Bayes Optimized

$$P(k=1 \mid \vec{x}) = .76$$

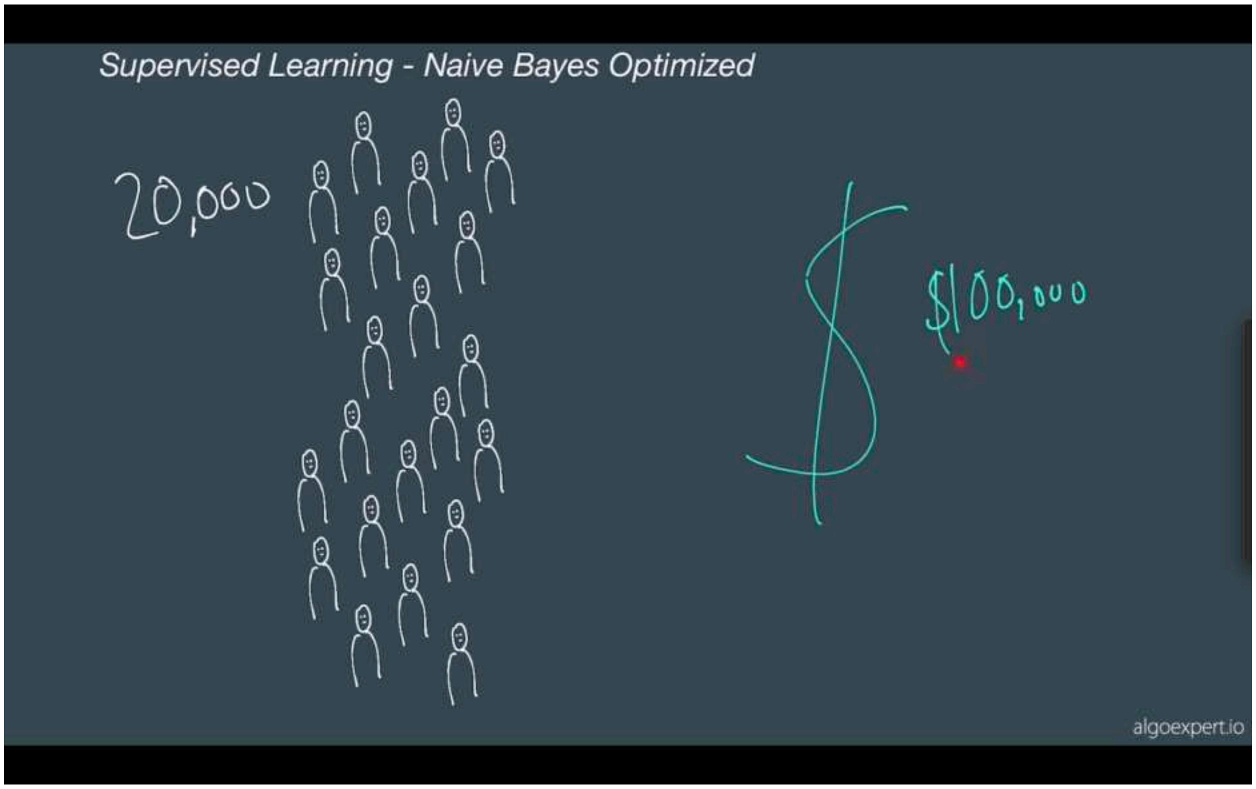
$$\vec{X} = [2, 1, 20.3]$$

I think that's a pretty good chance and I'd send them the promotion. Similar to how we did it in the spam filter we could decide on a threshold and say, if it's greater than 50%, then send them the discount. However, for marketing budgets, it could work quite differently.

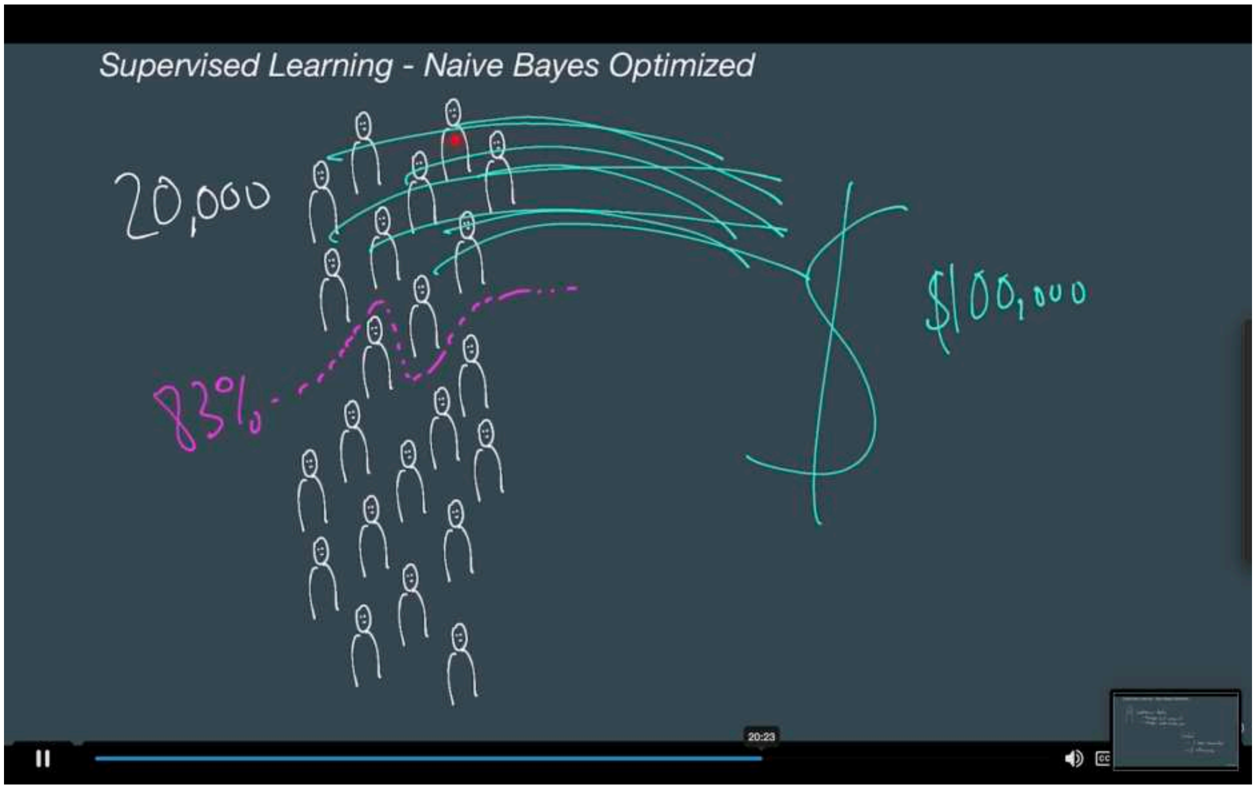


Supervised Learning - Naive Bayes Optimized

Let's say that this was our entire population, and let's say the probability of users converting to habitual users, let's say we left the threshold at 50%, that could separate the people like that. Now, typically for marketing, it will work differently, you'll usually have a budget say of $100,000 and you wanna give everyone a $5. So that means you have to find 20,000 people who are most likely to convert to habitual users, and let's say that threshold is 83%.

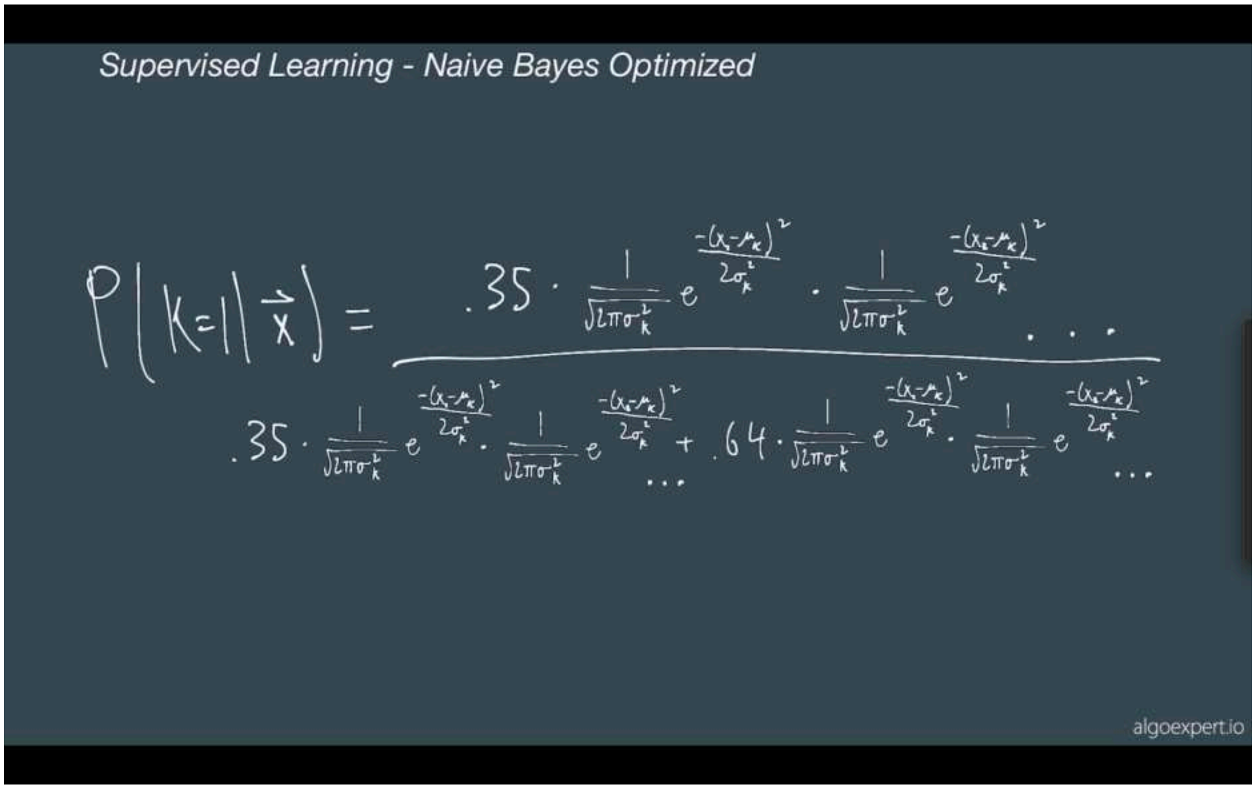


So then what would happen is we'd run every customer through our model, and if they have an 83% chance or greater of converting to a habitual user after receiving the promotion, we will send them that promotion to 20,000 people and we will have used all of our marketing budget of $100,000. What we've successfully done is maximized the return on our $100,000 by only sending to those users who are likely to convert to habitual customers.



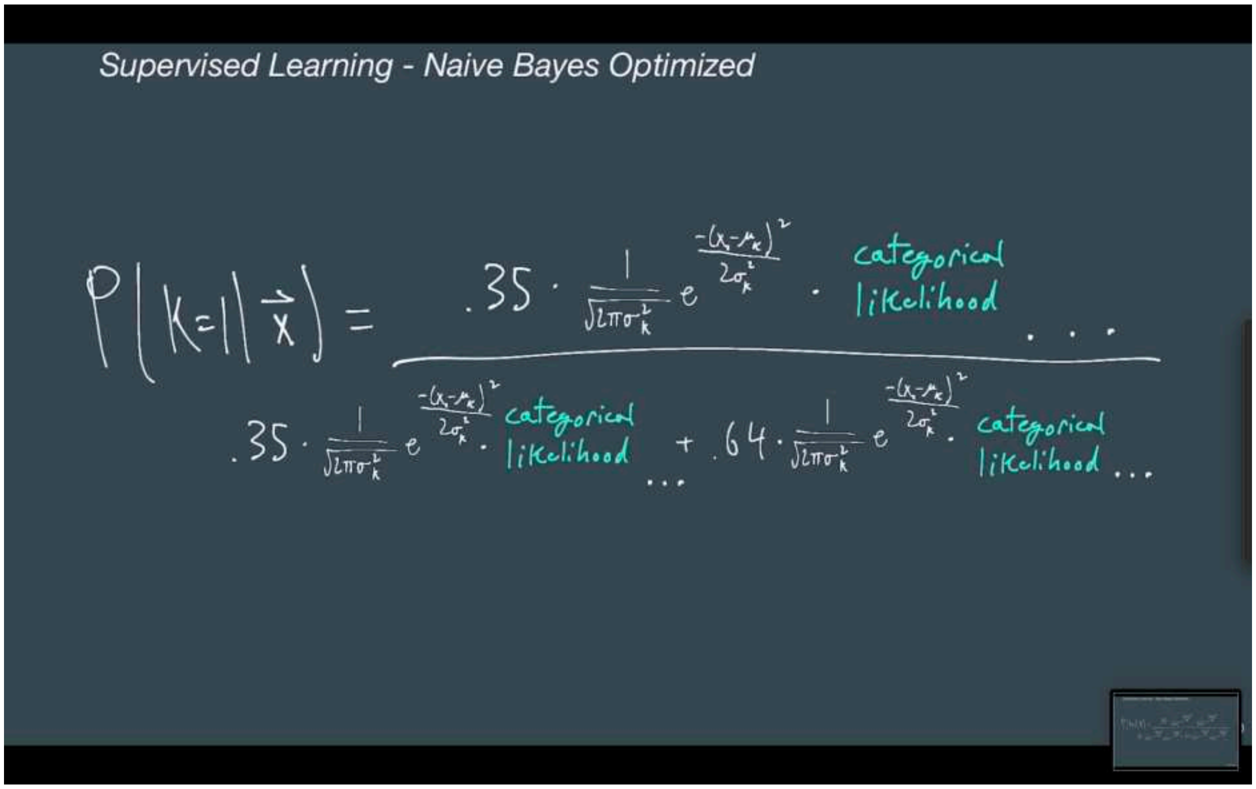
Okay, so our model is running quite well, things are going fine, our data engineer comes to us and says, hey, we have a new feature, let us know if it helps. And this new feature is a service preference. Service preference can either be pickup, take out or both.

How do we handle this categorical feature? Basically, we just have to put in a categorical likelihood alongside all of the other likelihoods that we're calculating.



Supervised Learning - Naive Bayes Optimized

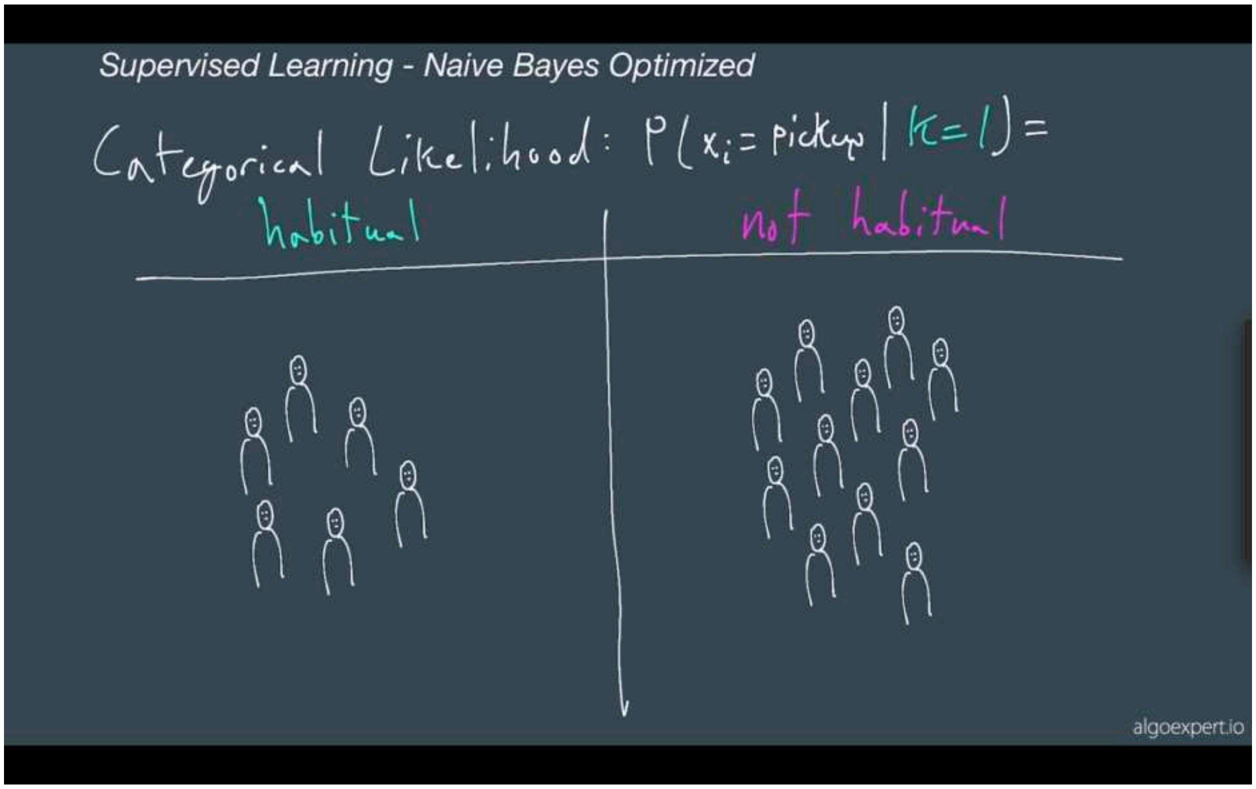
$$P(K=1|\vec{x}) = \frac{.35 \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_1-\mu_k)^2}{2\sigma_k^2}} \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_5-\mu_k)^2}{2\sigma_k^2}} \cdots}{.35 \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_1-\mu_k)^2}{2\sigma_k^2}} \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_5-\mu_k)^2}{2\sigma_k^2}} \cdots + .64 \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_1-\mu_k)^2}{2\sigma_k^2}} \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_5-\mu_k)^2}{2\sigma_k^2}} \cdots}$$



Supervised Learning - Naive Bayes Optimized

$$P(K=1|\vec{x}) = \frac{.35 \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_1-\mu_k)^2}{2\sigma_k^2}} \cdot \text{categorical likelihood} \cdots}{.35 \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_1-\mu_k)^2}{2\sigma_k^2}} \cdot \text{categorical likelihood} \cdots + .64 \cdot \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{\frac{-(x_1-\mu_k)^2}{2\sigma_k^2}} \cdot \text{categorical likelihood} \cdots}$$

We can actually mix and match here. So how do we actually calculate categorical likelihood? Well, if we're looking for the likelihood that someone's service preference is pickup, given that the label is one, then we simply count up the habitual users whose service preference is pickup and divide by the total number of habitual users. So here that's two over six, which is 33%.


Supervised Learning - Naive Bayes Optimized
Categorical Likelihood: $P(x_i = \text{pickup} \mid k=1) =$
habitual
not habitual
algoexpert.io

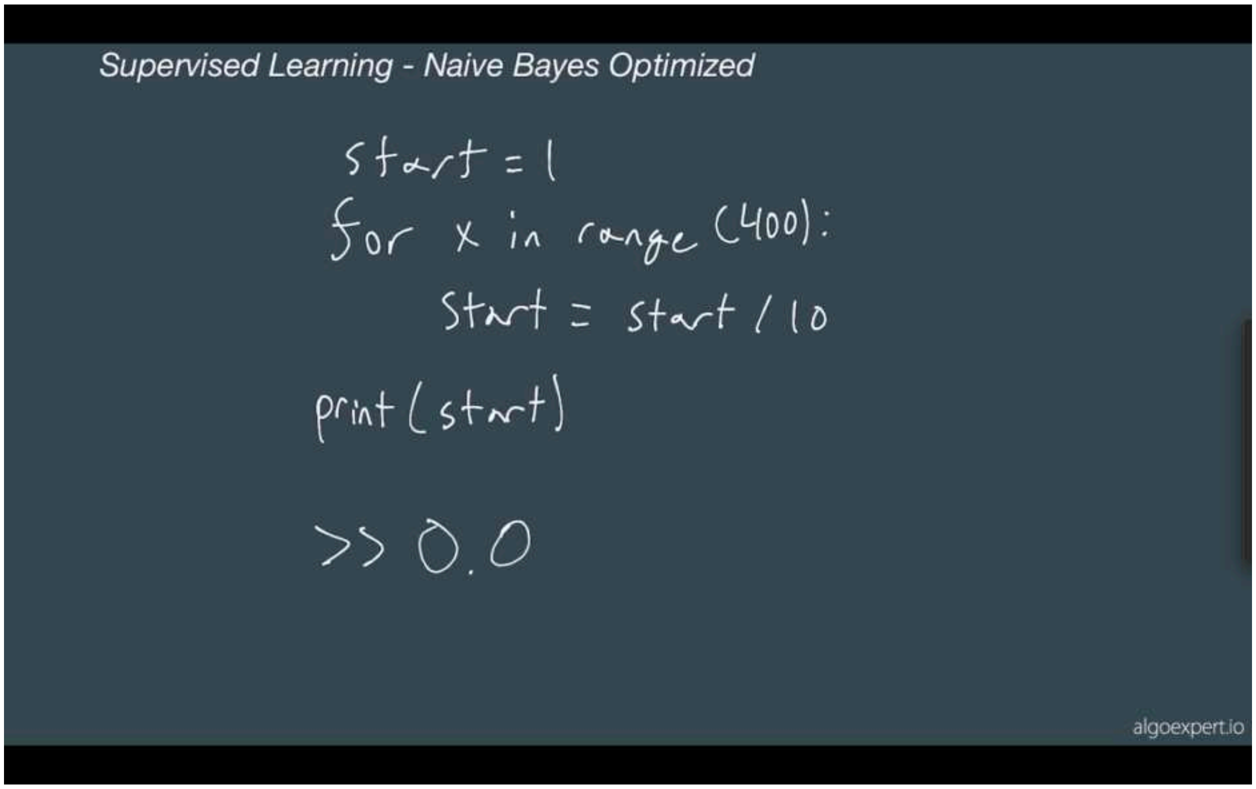
Likewise, if we wanted to calculate the likelihood of service preference being pickup, given that the person did not convert to a habitual user, we would simply count up the number of non-habitual users whose preference is pickup and divide by the total number of non-habitual users.

So that's how we can categorical likelihood and we can actually mix that in with numerical features, using the Gaussian likelihood. So what happens when our customer base begins to shift? So we launch our model out and it's working well, so more users are converting to habitual users.

Well, there's two things that we can do. One, we can retrain the model. So if our population goes from here, you can retrain the model, and then if it moves like this, we retrain the model, say once a month or once a week. Another way that we can do it is more of an online learning it's called where for every single user that ends up converting, so for instance, if this person, all of a sudden converts after receiving the promotion, we now have another piece of labeled data so we can go into our model and update those parameters just as that one person converted.

We would stop the need for massive updates and retraining from scratch, and we would incrementally train the model as users moved over from non-habitual to habitual. Or likewise, if someone moved from habitual to non-habitual. But one thing to note is that there will be problems when it comes to implementing this. One of those problems is going to have to deal with these likelihood term multiplications in the numerators and the denominators.

Let's say that we have 100 features, so that means we're going to be multiplying 100 things, if those values are between zero and one, and they are because of probabilities, then that number will gradually go towards zero.



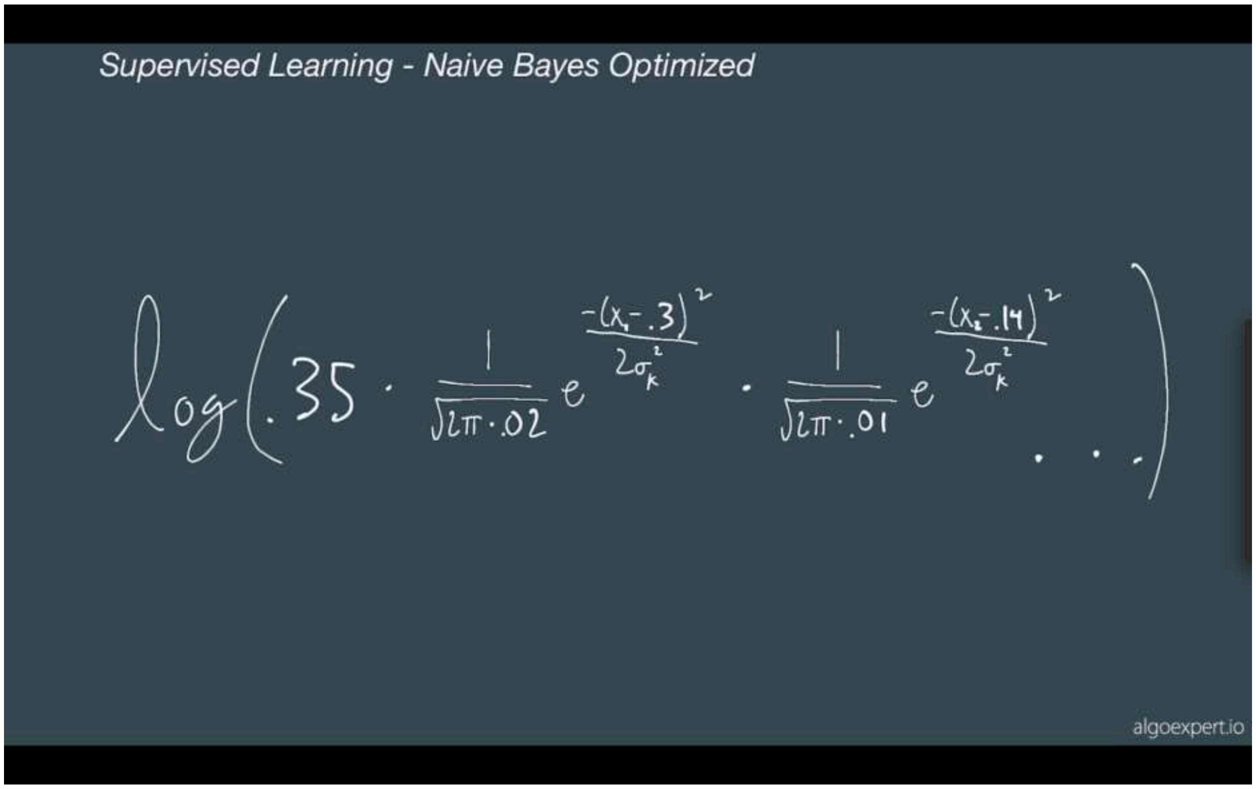
Supervised Learning - Naive Bayes Optimized

```
start = 1
for x in range(400):
    start = start / 10
print(start)

>> 0.0
```

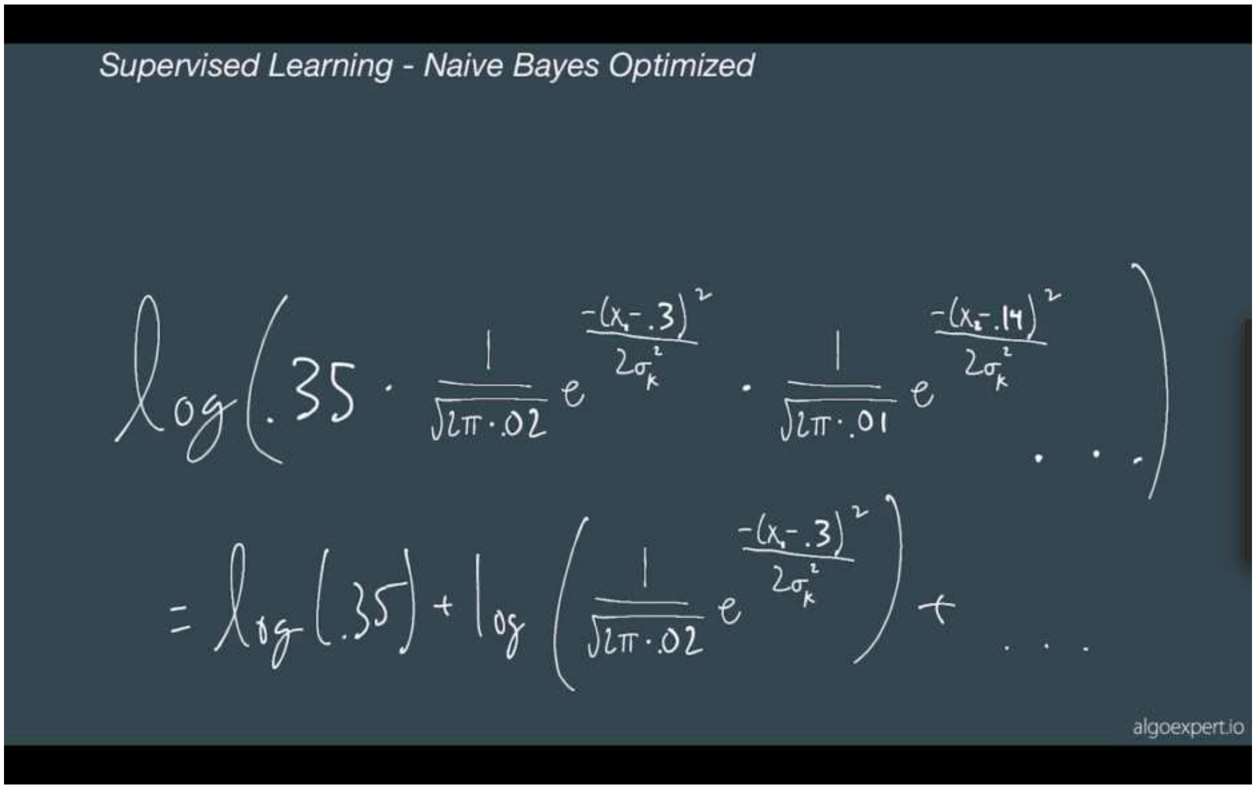
So for instance, as an example, you can open up a Python interpreter or anything like that, and just start with the number one and then divide that number by 10, 400 times, you're gonna get at zero. Even though the answer isn't technically zero, it's a very small number, software typically has problems representing such infinitesimally small digits.

So we need a solution to figure this out, and we can just take the log of the numerator.


Supervised Learning - Naive Bayes Optimized

$$\log\left(.35 \cdot \frac{1}{\sqrt{2\pi \cdot .02}} e^{\frac{-(x_1 - .3)^2}{2\sigma_k^2}} \cdot \frac{1}{\sqrt{2\pi \cdot .01}} e^{\frac{-(x_2 - .14)^2}{2\sigma_k^2}} \cdots\right)$$

And you might be thinking, well, we still have these multiplications what did we solve? Well, an interesting property of the log is that can separate out the terms in the log and separate them out with an addition. So now what this does is it allows us to represent our same model without all of the repeated multiplications that can erase the number to zero.

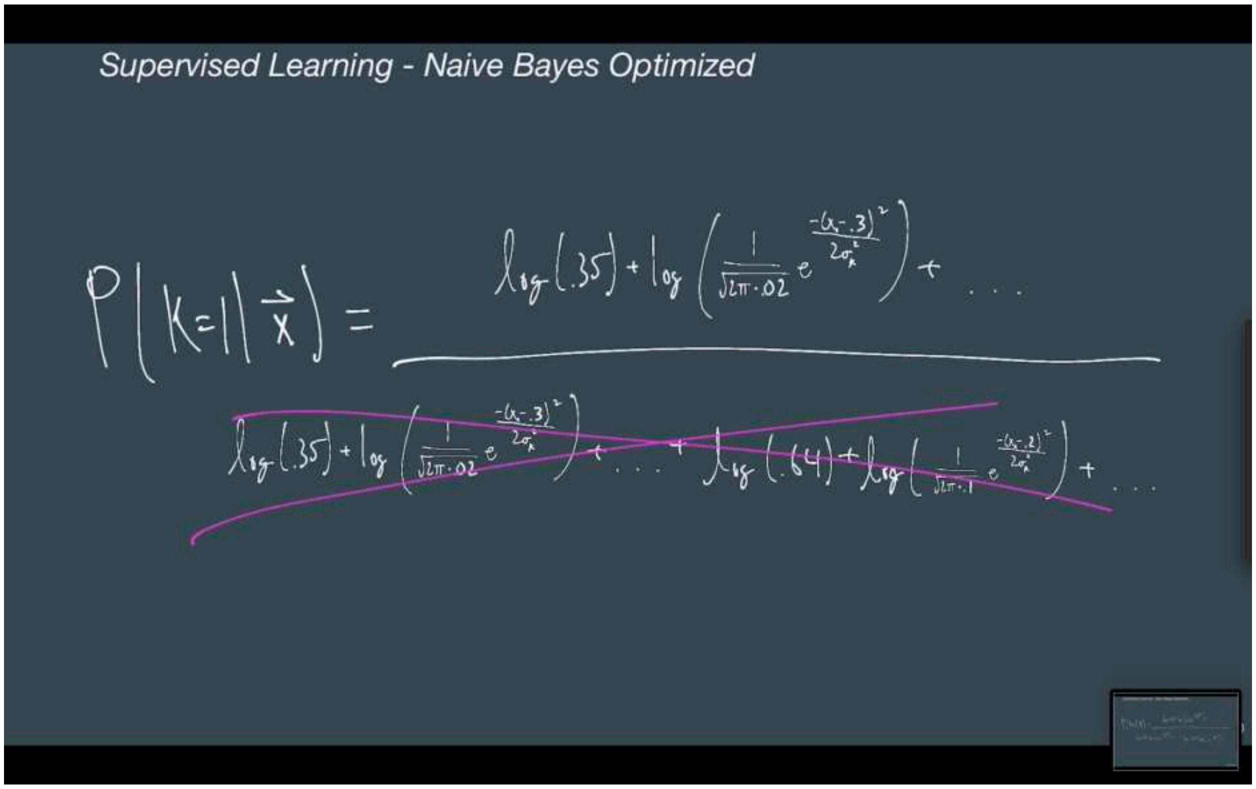


Supervised Learning - Naive Bayes Optimized

$$\log\left(.35 \cdot \frac{1}{\sqrt{2\pi \cdot .02}} e^{\frac{-(x_1-.3)^2}{2\sigma_k^2}} \cdot \frac{1}{\sqrt{2\pi \cdot .01}} e^{\frac{-(x_2-.14)^2}{2\sigma_k^2}} \cdots\right)$$

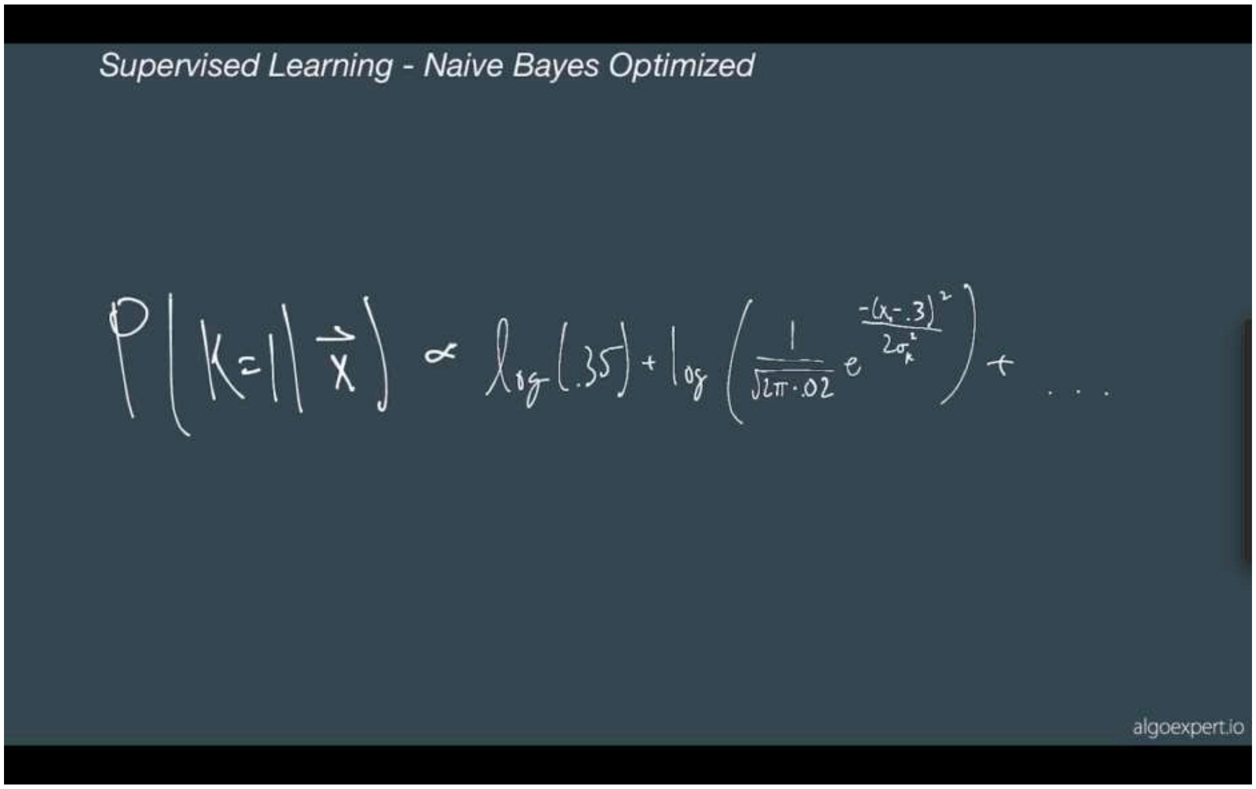
$$= \log(.35) + \log\left(\frac{1}{\sqrt{2\pi \cdot .02}} e^{\frac{-(x_1-.3)^2}{2\sigma_k^2}}\right) + \cdots$$

One final thing to note, we technically don't need to calculate the denominator because the denominator is going to be constant for every single class that we evaluate.


Supervised Learning - Naive Bayes Optimized

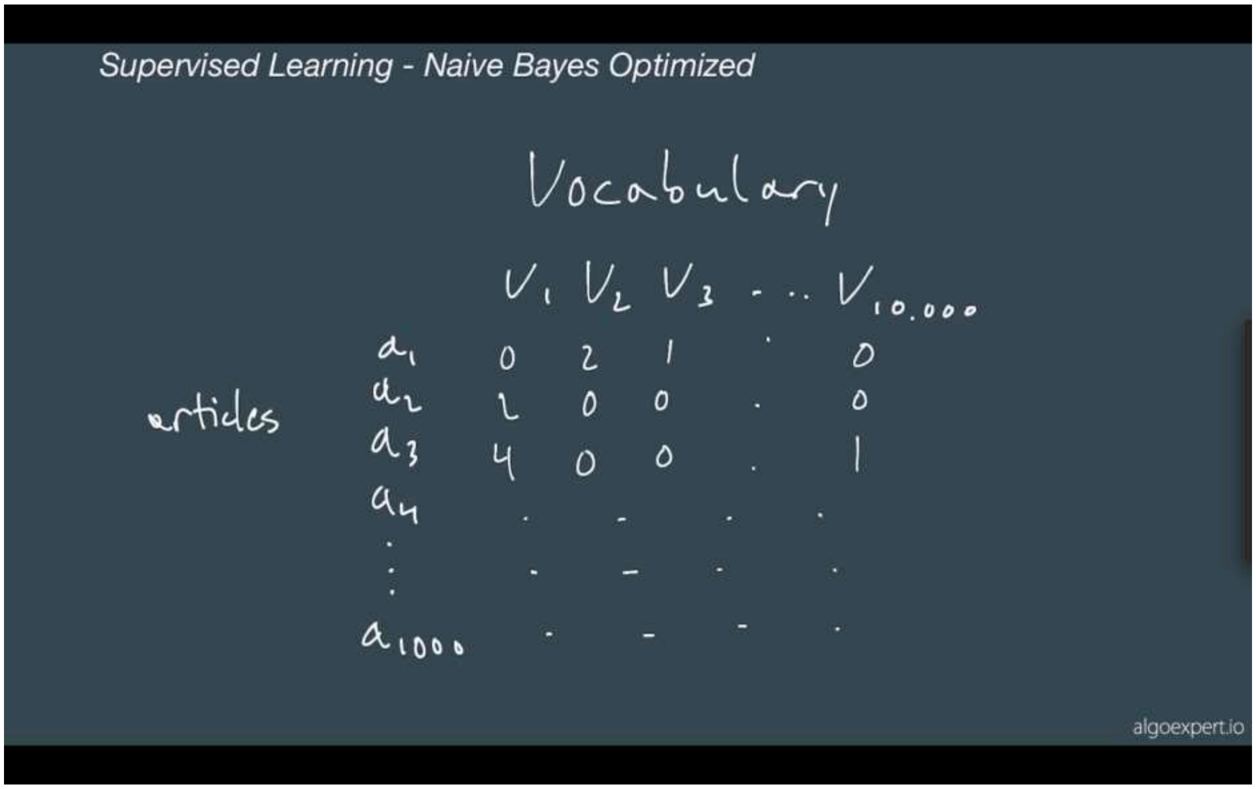
$$P(K=1 \mid \vec{x}) = \frac{\log(.35) + \log\left(\frac{1}{\sqrt{2\pi \cdot .02}} e^{\frac{-(x_1-.3)^2}{2\sigma_x^2}}\right) + \dots}{\cancel{\log(.35) + \log\left(\frac{1}{\sqrt{2\pi \cdot .02}} e^{\frac{-(x_1-.3)^2}{2\sigma_x^2}}\right) + \dots + \log(.64) + \log\left(\frac{1}{\sqrt{2\pi \cdot .1}} e^{\frac{-(x_1-.2)^2}{2\sigma_x^2}}\right) + \dots}}$$

So what we can actually do is just evaluate the numerator and then take the maximum value of the numerator. So our formula actually turns into this, where this is actually a proportion to sign, not equal to, since we're not dividing by the numerator anymore, we can't technically say this is equal, but it will definitely be proportional since the only thing that's changing is the numerator.


Supervised Learning - Naive Bayes Optimized

$$P(K=1 \mid \vec{X}) \propto \log(.35) + \log\left(\frac{1}{\sqrt{2\pi \cdot .02}} e^{\frac{-(x_1-.3)^2}{2\sigma_k^2}}\right) + \ldots$$

So next up I wanna cover **memory optimizations.** Imagine we have 10,000 words in our vocabulary that we've discovered across 1,000 articles. If we assume a 32 bit to represent each of these elements, then we're going to get a 40 gigabyte matrix.



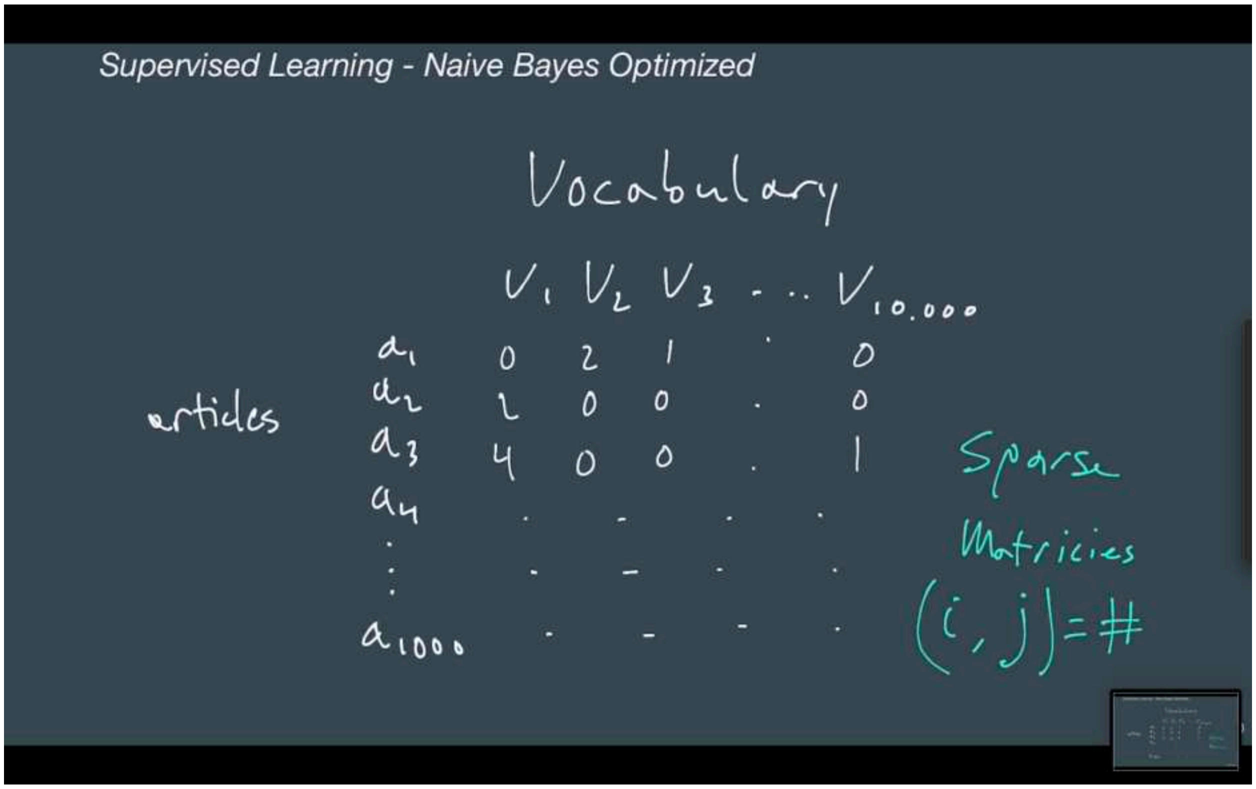
Supervised Learning - Naive Bayes Optimized

Vocabulary

$$\text{articles}\quad\begin{array}{c|ccccc} & V_1 & V_2 & V_3 & \cdots & V_{10.000} \\ a_1 & 0 & 2 & 1 & \cdot & 0 \\ a_2 & 2 & 0 & 0 & \cdot & 0 \\ a_3 & 4 & 0 & 0 & \cdot & 1 \\ a_4 & \cdot & - & \cdot & \cdot & \\ \vdots & - & - & \cdot & \cdot & \\ a_{1000} & \cdot & - & - & \cdot & \end{array}$$

algoexpert.io

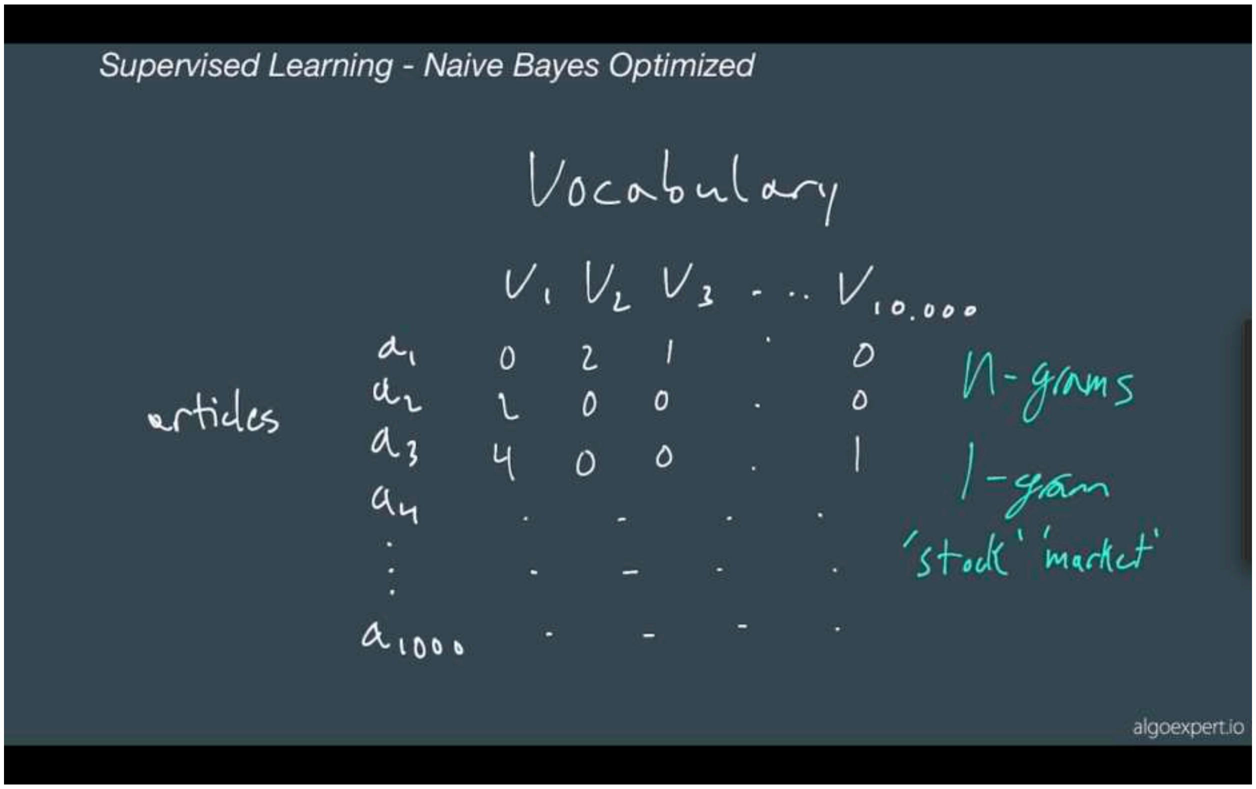
Fortunately for us, most of these elements are zero because a lot of these articles will only contain some of the words. How we represent **sparse matrices** well, one way is to take the, i, j as the key. So for instance, this would be three, one for this element and the value would just be the value in the original matrix.



$$\begin{array}{c|ccccc} & V_1 & V_2 & V_3 & \cdots & V_{10,000} \\ \hline a_1 & 0 & 2 & 1 & \cdot & 0 \\ a_2 & 2 & 0 & 0 & \cdot & 0 \\ a_3 & 4 & 0 & 0 & \cdot & 1 \\ a_4 & \cdot & \cdot & \cdot & \cdot & \\ \vdots & \cdot & \cdot & \cdot & \cdot & \\ a_{1000} & \cdot & \cdot & \cdot & \cdot & \end{array}$$

And if the value and the original matrix is zero, we just don't put that element in the hash map.

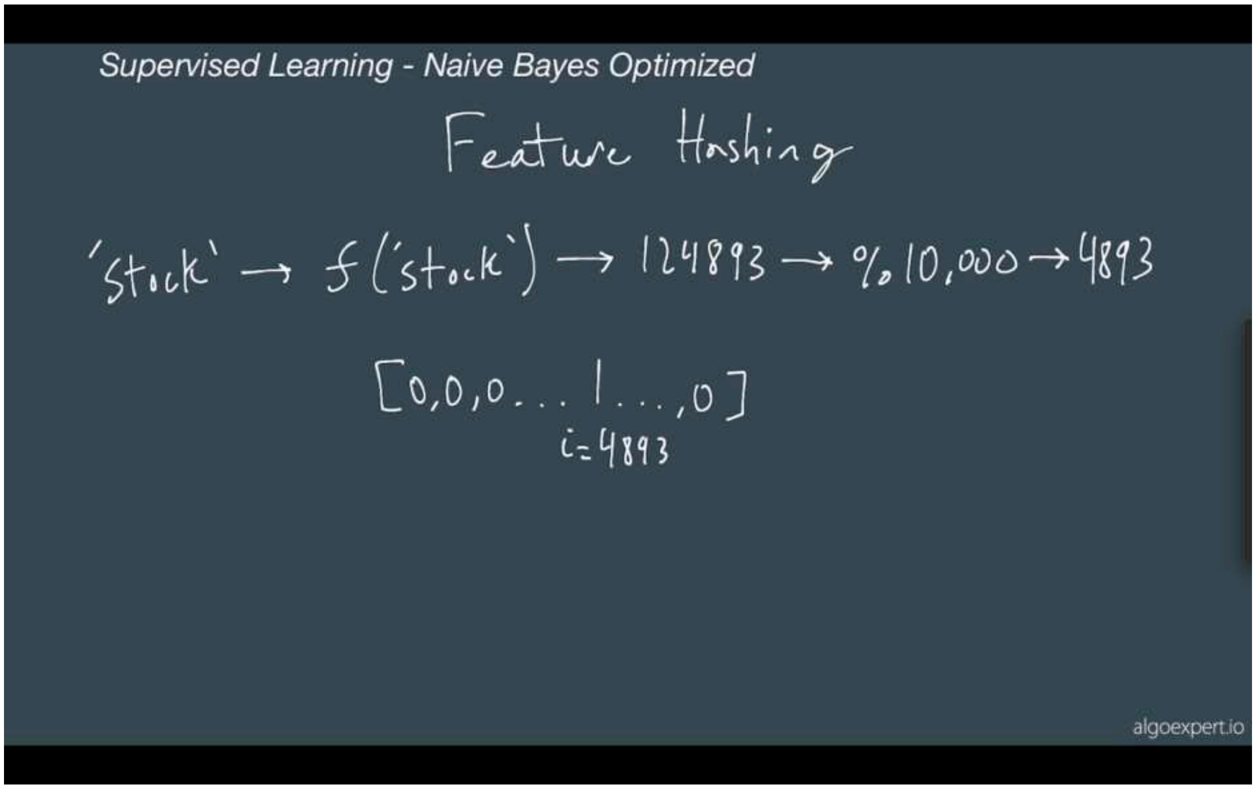
Another way we can reduce the size of this vocabulary is to take advantage of **n-grams.** So right now, every single word gets its own entry in this vocabulary. These are called one-grams, for instance, stock and market. An example of a two-gram would be stock market. So here instead of stocks and market taking up V1 and V2, V1, would just take up take up a stock market.



$$\begin{array}{c|ccccc} & V_1 & V_2 & V_3 & \dots & V_{10.000} \\ \hline a_1 & 0 & 2 & 1 & \cdot & 0 \\ a_2 & 2 & 0 & 0 & \cdot & 0 \\ a_3 & 4 & 0 & 0 & \cdot & 1 \\ a_4 & \cdot & - & \cdot & \cdot & \\ \vdots & - & - & \cdot & \cdot & \\ a_{1000} & \cdot & - & - & \cdot & \end{array}$$

So finally we can just ditch the entire vocabulary in itself and take advantage of something called **feature hashing**. This is actually a really cool concept.

We take a word, we put it inside of a hash function, we get the value of the hash function, we mode it by how much ever space we want to use.

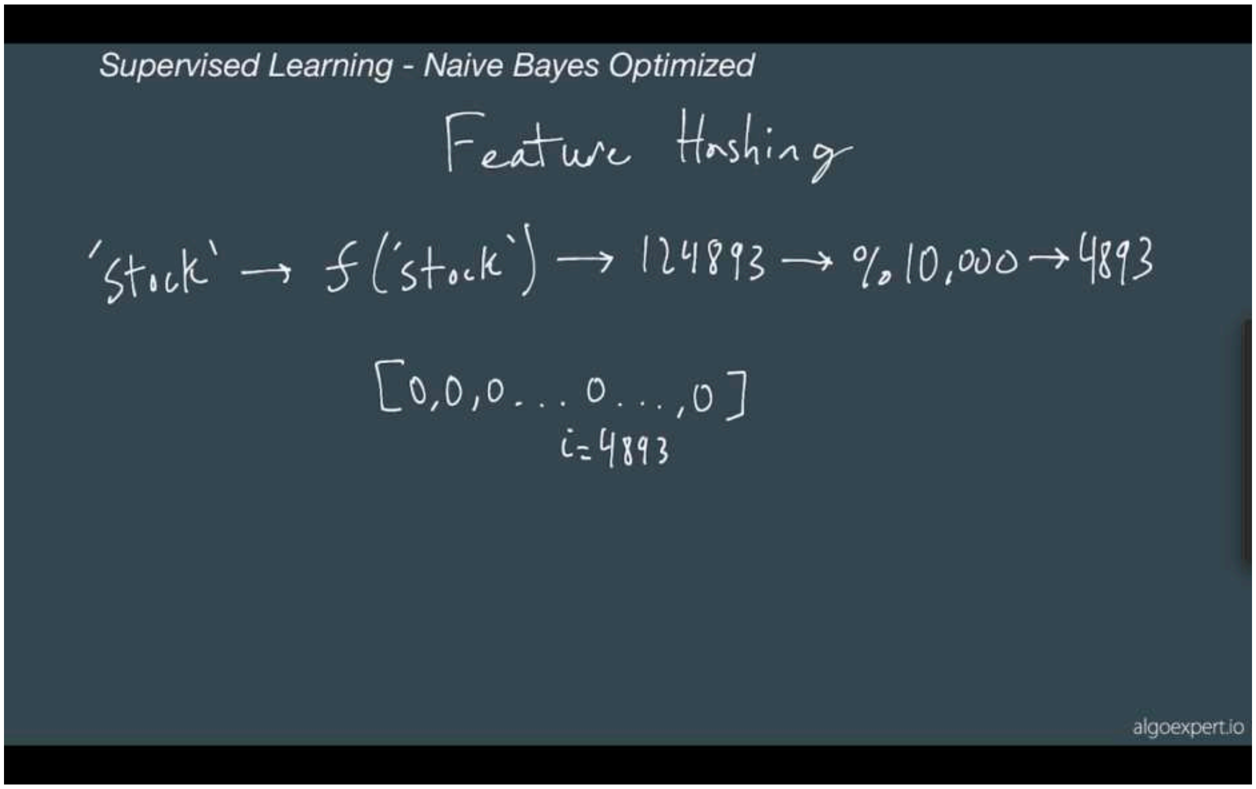

Supervised Learning - Naive Bayes Optimized

Feature Hashing

$$\text{'stock'} \rightarrow f(\text{'stock'}) \rightarrow 124893 \rightarrow \%\,10{,}000 \rightarrow 4893$$

$$[0,0,0 \dots \underset{i=4893}{1} \dots, 0]$$

So let's say we wanna use 10,000 elements worth of space to represent a vocabulary, and we get an index I, we map this index I to an array and we simply increment that element. So that would now go from a zero to a one.



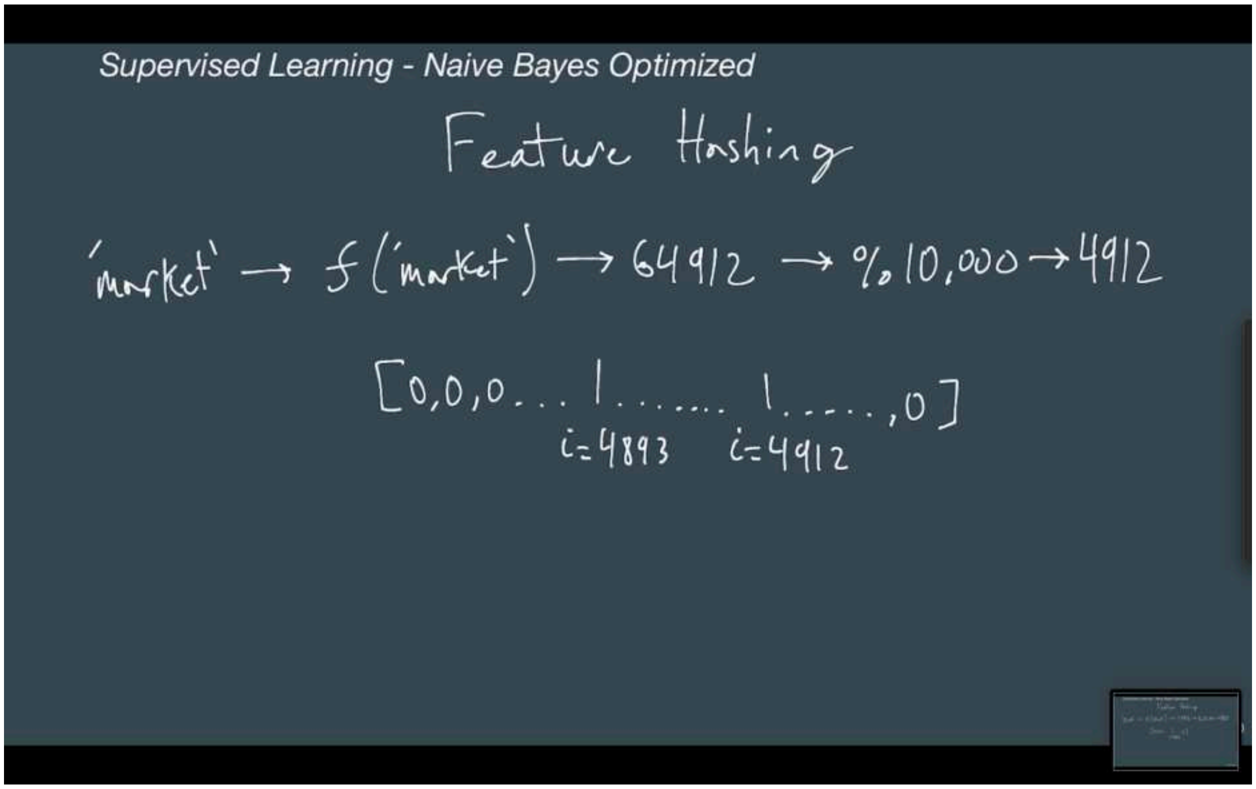
Supervised Learning - Naive Bayes Optimized

Feature Hashing

$$'stock' \rightarrow f('stock') \rightarrow 124893 \rightarrow \% 10{,}000 \rightarrow 4893$$

$$[0,0,0 \ldots \underset{i=4893}{0} \ldots, 0]$$

Let's say we got the word market in here, we'd hash function in the market, we'd get that. We take the modulates of 10,000, we'd get another index that would be here and we'd increment that index from zero to one.

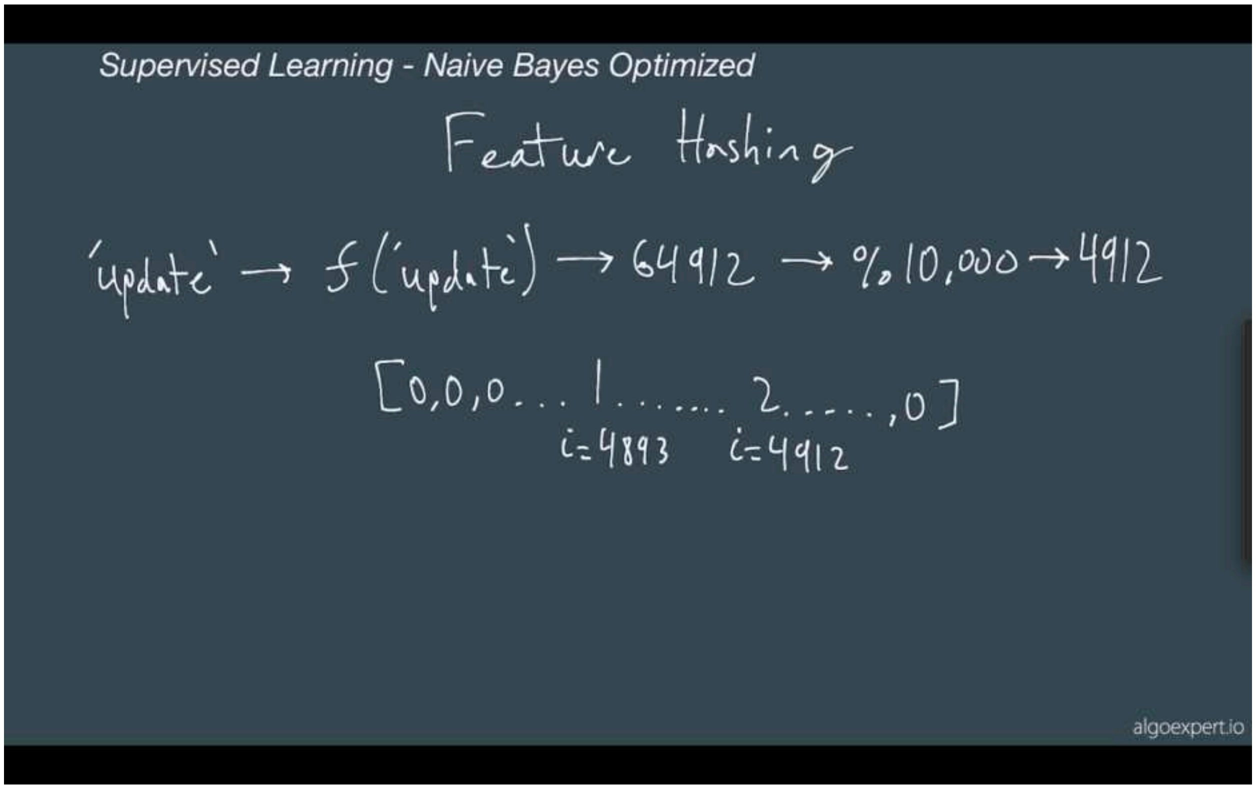

Supervised Learning - Naive Bayes Optimized

Feature Hashing

$$\text{'market'} \rightarrow f(\text{'market'}) \rightarrow 64912 \rightarrow \%10{,}000 \rightarrow 4912$$

$$[0,0,0 \ldots \underset{i=4893}{1} \ldots\ldots \underset{i=4912}{1} \ldots\ldots, 0]$$

Let's say we got the word update, let's say it hash to the same value, which means they would mode to the same value, we would simply update that index again.


Supervised Learning - Naive Bayes Optimized

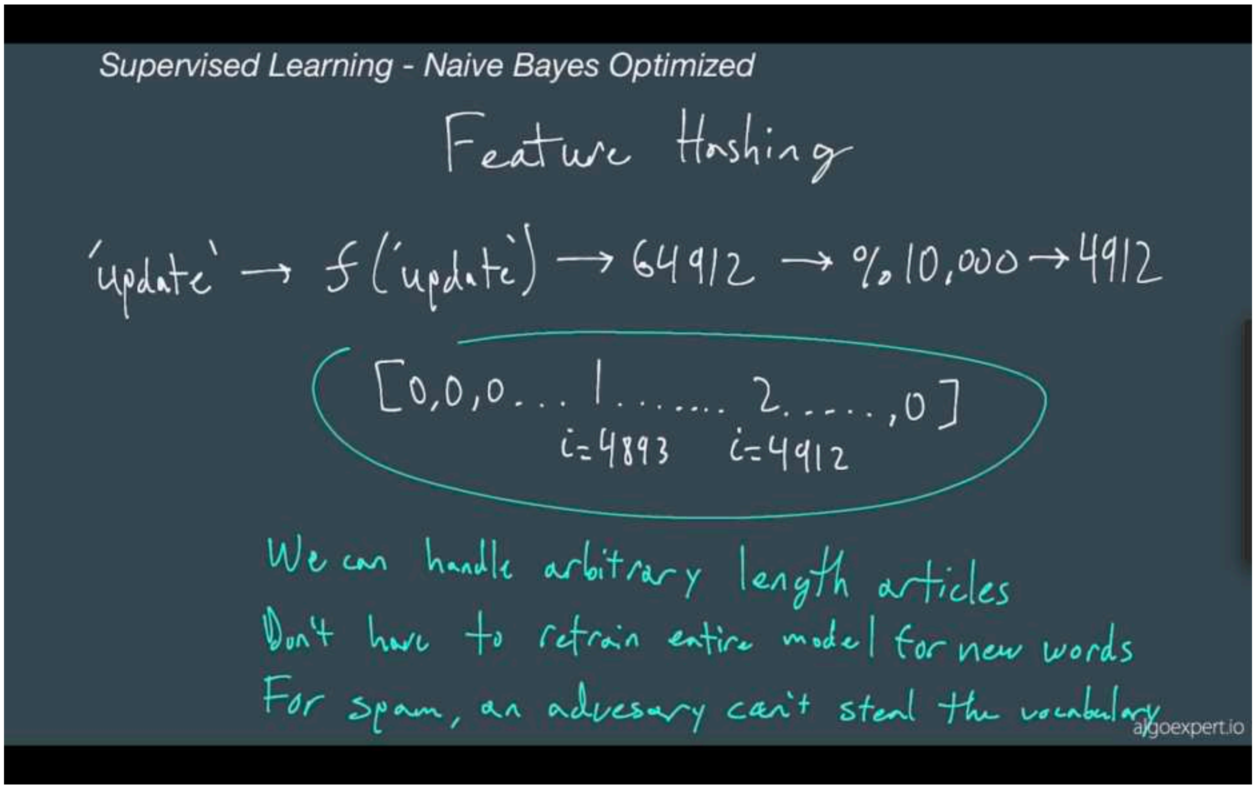
Feature Hashing

$$\text{'update'} \rightarrow f(\text{'update'}) \rightarrow 64912 \rightarrow \%10{,}000 \rightarrow 4912$$

$$[0,0,0\ldots \underset{i=4893}{1} \ldots\ldots \underset{i=4912}{2} \ldots\ldots, 0]$$

So for collisions, we just keep updating. Now what's really interesting about this, is this means we can handle arbitrary length documents, right?

Because as long as the words keep coming in, we can represent them by just incrementing specific indices, given a particular word. Another interesting thing is that we don't have to retrain models so often for every single new word that comes up because these new words can just be hashed to a specific index and then we can continue on.



Supervised Learning - Naive Bayes Optimized

Feature Hashing

$'update' \rightarrow f('update') \rightarrow 64912 \rightarrow \%10{,}000 \rightarrow 4912$

$[0,0,0 \ldots 1 \ldots\ldots 2 \ldots\ldots, 0]$

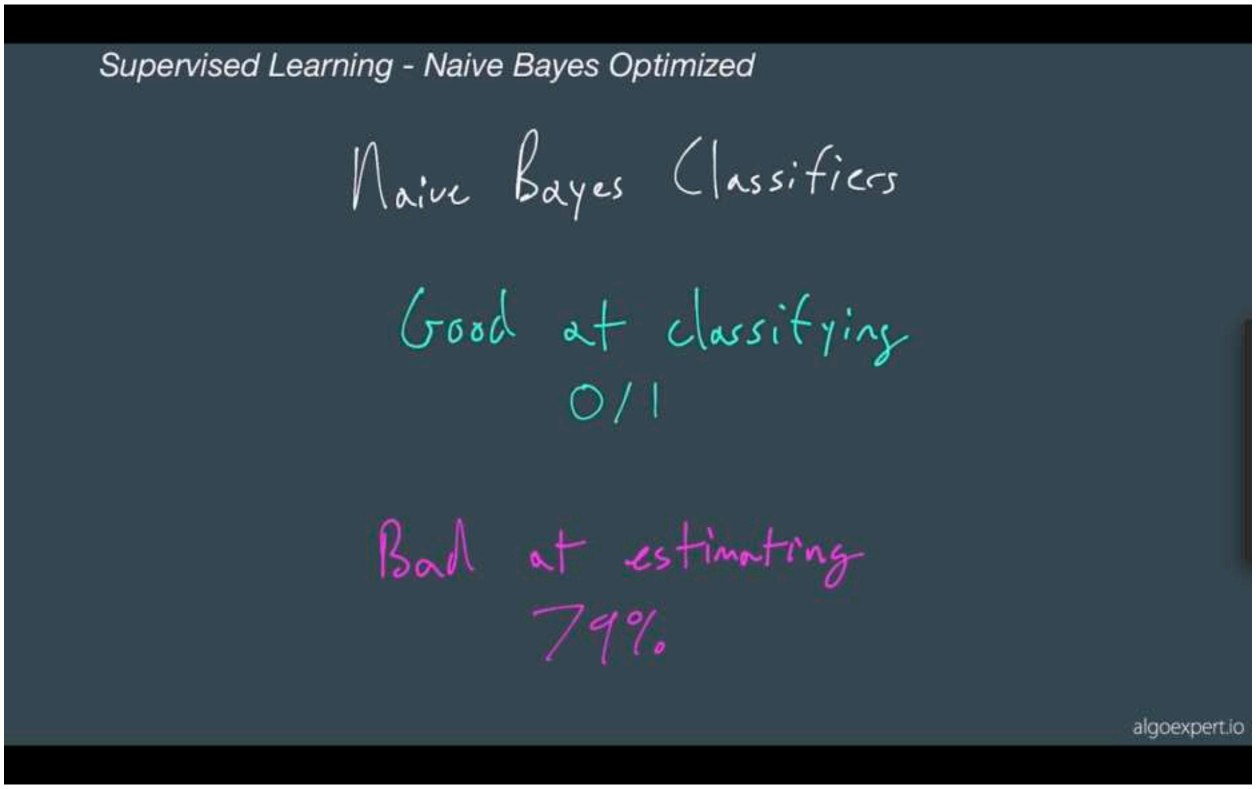
$i = 4893 \quad i = 4912$

We can handle arbitrary length articles

Don't have to retrain entire model for new words

For spam, an advesary can't steal the vocabulary

Finally for spam, let's say an adversary gets a hold of whatever vocabulary that you use to detect spam, here they wouldn't have a vocabulary to obtain, so it can be beneficial that way.



Supervised Learning - Naive Bayes Optimized

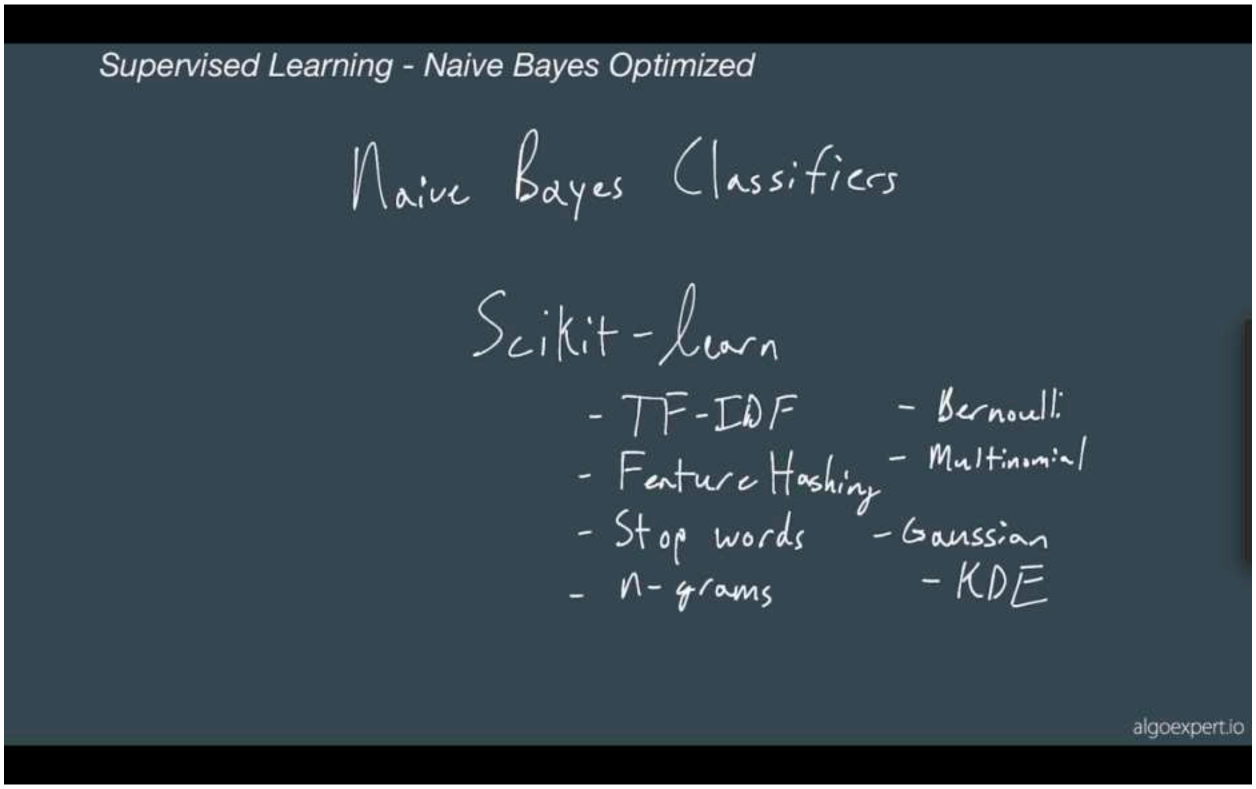
Naive Bayes Classifiers

Good at classifying
0/1

Bad at estimating
79%

algoexpert.io

One thing to keep in mind as well, so Naive Bayes classifiers are good at **classifying**. Okay, between zero and one, however, they're bad at **estimating**. So take the probabilities with a grain of salt, but thanks to the threshold, they actually end up being good classifiers.


Supervised Learning - Naive Bayes Optimized

Naive Bayes Classifiers

Scikit-learn

- TF-IDF
- Feature Hashing
- Stop words
- n-grams

- Bernoulli
- Multinomial
- Gaussian
- KDE

In general, a good library for Naive Bayes is **scikit-learn**, they allow you to use TF-IDF, they support feature hashing, stop words, n-grams. They support Bernoulli, multinomial, Gaussian.

And one thing in case it wasn't clear, you can only use the Gaussian likelihood if the values that you're trying to estimate are Gaussian in themselves. So if your data follows a power distribution, it's probably not going to work very well using the Gaussian likelihood. So they actually support something called KDE, we're not gonna get too far into it, it's called **kernel density estimation**, all it does is it tries to estimate the distribution of your data through a series of typically Gaussian distribution. So even if you're unsure about what distribution your data actually follows, you can use KDE to help you figure that out.

So just to wrap this up, what we did is we figured out how to use larger documents with Naive Bayes classifier. With that we needed a multi-classification, we needed to use the multinomial likelihood, when we don't have words and we have continuous data, we can use the Gaussian likelihood. We figured out how to get rid of the vocabulary completely and instead we can use feature hashing, and we also figured out some cool libraries that we can use to implement the stuff on our own. Thanks for joining us in this video, join us next video as we continue our machine learning journey.