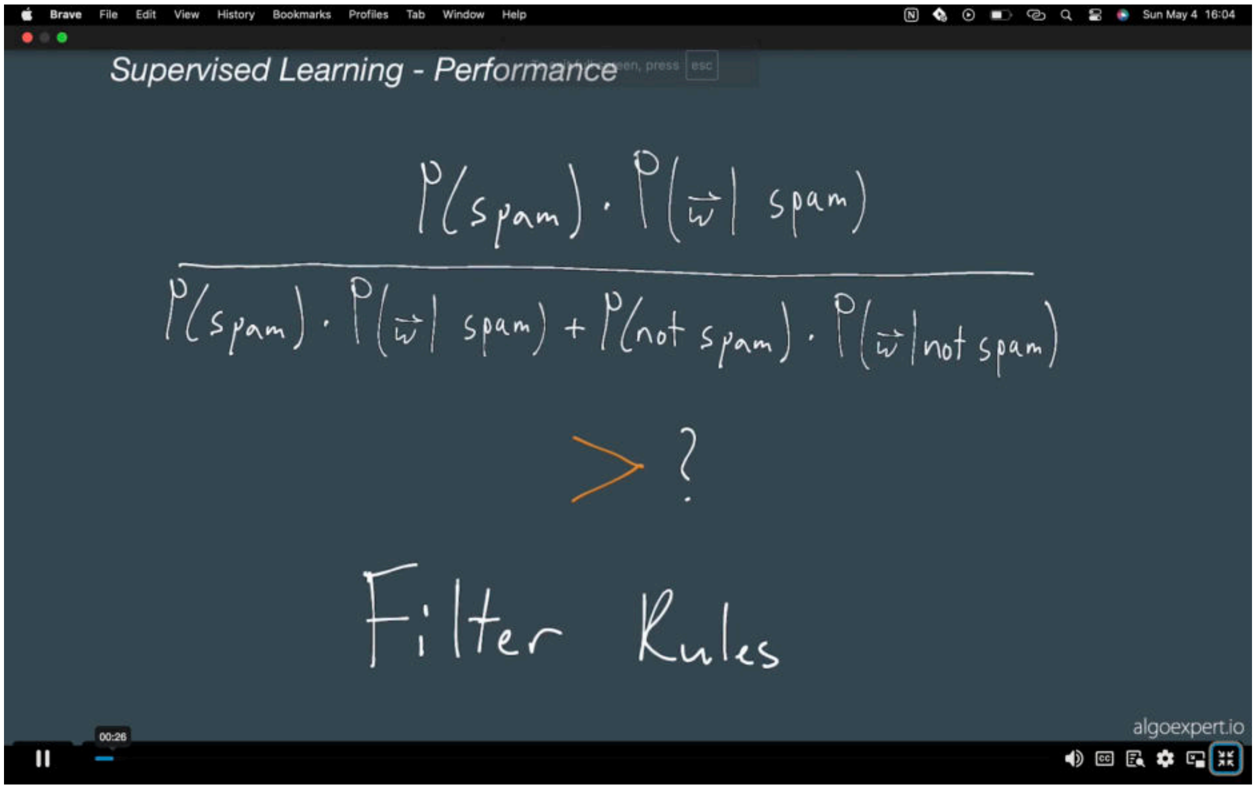
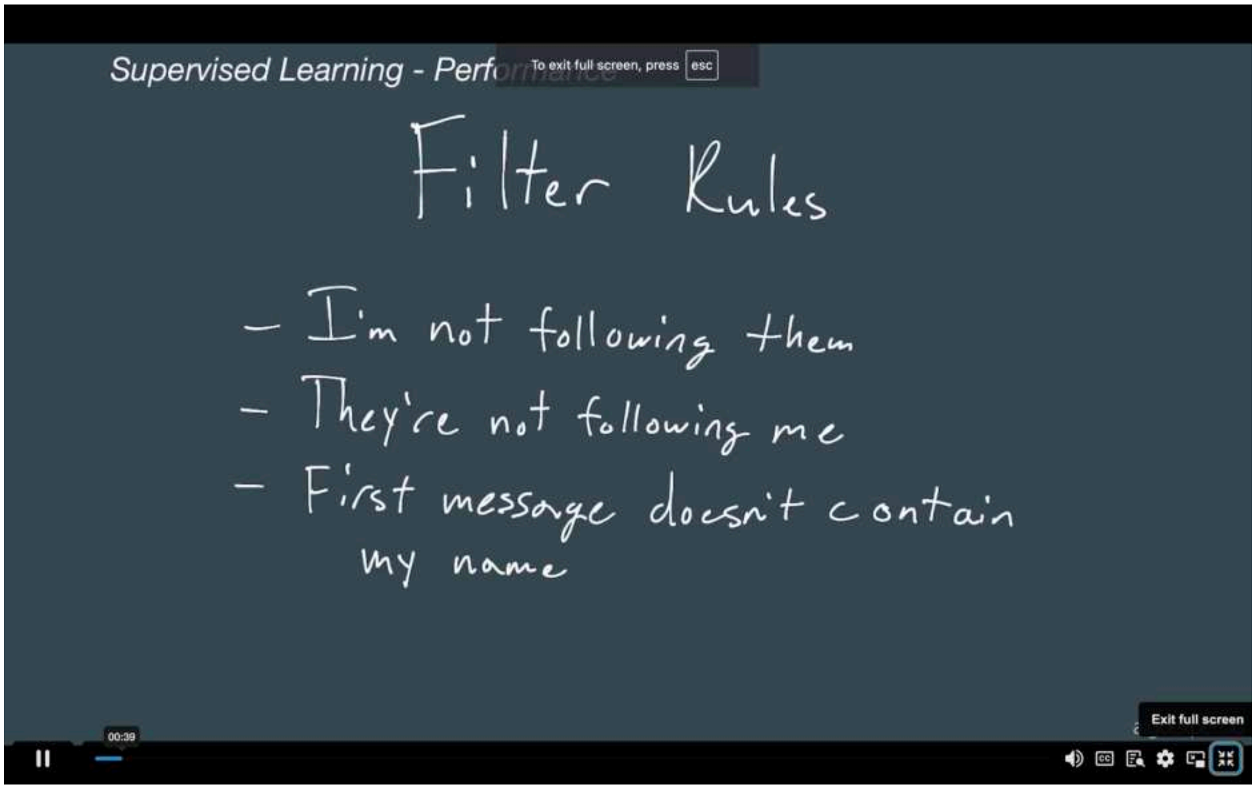
Performance

Welcome back everybody. This is ML experts, Machine Learning Crash Course. In this video we're going to be talking about "Model Performance". And our last session we designed our first model. In this video, we're going to be going over how to decide if the model we built is an improvement on our filter rules or heuristic.


Supervised Learning - Performance
$$\frac{P(\text{spam}) \cdot P(\vec{w} \mid \text{spam})}{P(\text{spam}) \cdot P(\vec{w} \mid \text{spam}) + P(\text{not spam}) \cdot P(\vec{w} \mid \text{not spam})}$$
> ?
Filter Rules
algoexpert.io
00:26

The point of this video, is to figure out what this greater sign actually means. But first let's solidify the decision points for when we declare something as spam.

For the filter rules it was quite simple. If any of these conditions were met, then we would determine that the message was spam. For the model though, it would output a particular probability.

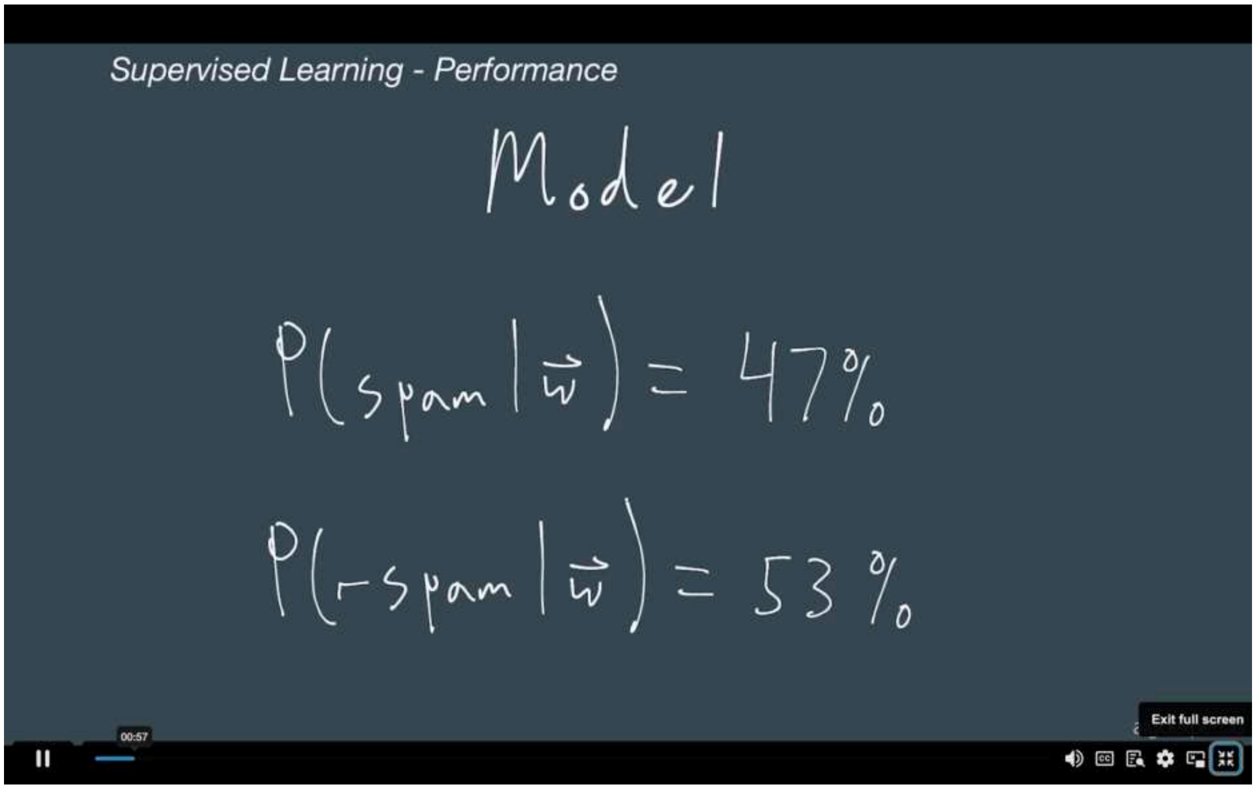


Supervised Learning - Perf

Filter Rules

- I'm not following them
- They're not following me
- First message doesn't contain my name

In this case 47% probability of being spam, given some message. Well, how do we relate that to actually deciding if the message is spam or not? Since the probability of spam is 47%, that means the probability of it not being spam is 53%. The common decision point is to select the highest probability among the two possibilities.



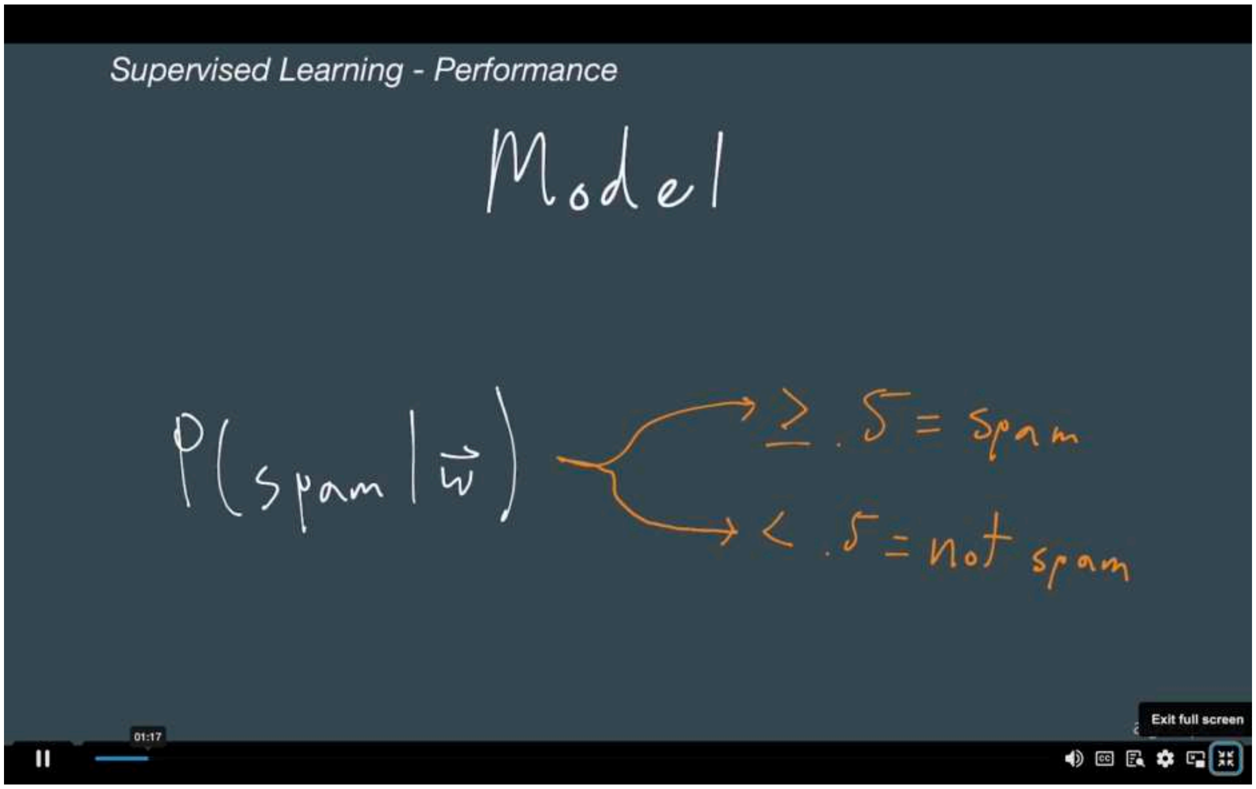
Supervised Learning - Performance

Model

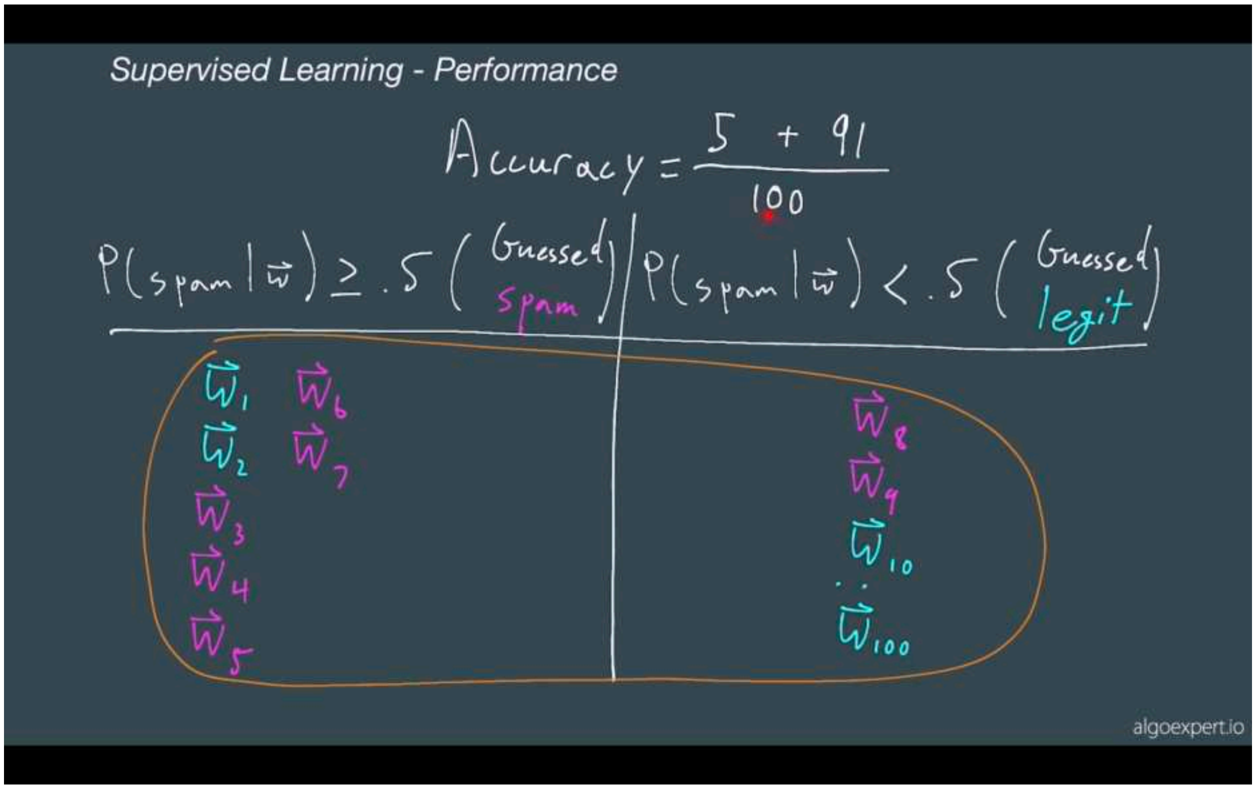
$$P(spam \mid \vec{w}) = 47\%$$

$$P(\neg spam \mid \vec{w}) = 53\%$$

So here, since the probability of the message not being spam is greater than the probability of the message being spam, we would deem the message legitimate. So, we'll define our model's decision pool, such that if the probability of spam is greater than or equal to 50%, then we will deem the message spam. If it's less than 50%, we'll declare the message as not spam.


Supervised Learning - Performance
Model
$P(\text{spam} \mid \vec{w})$
$\geq .5 =$ spam
$< .5 =$ not spam
01:17
Exit full screen

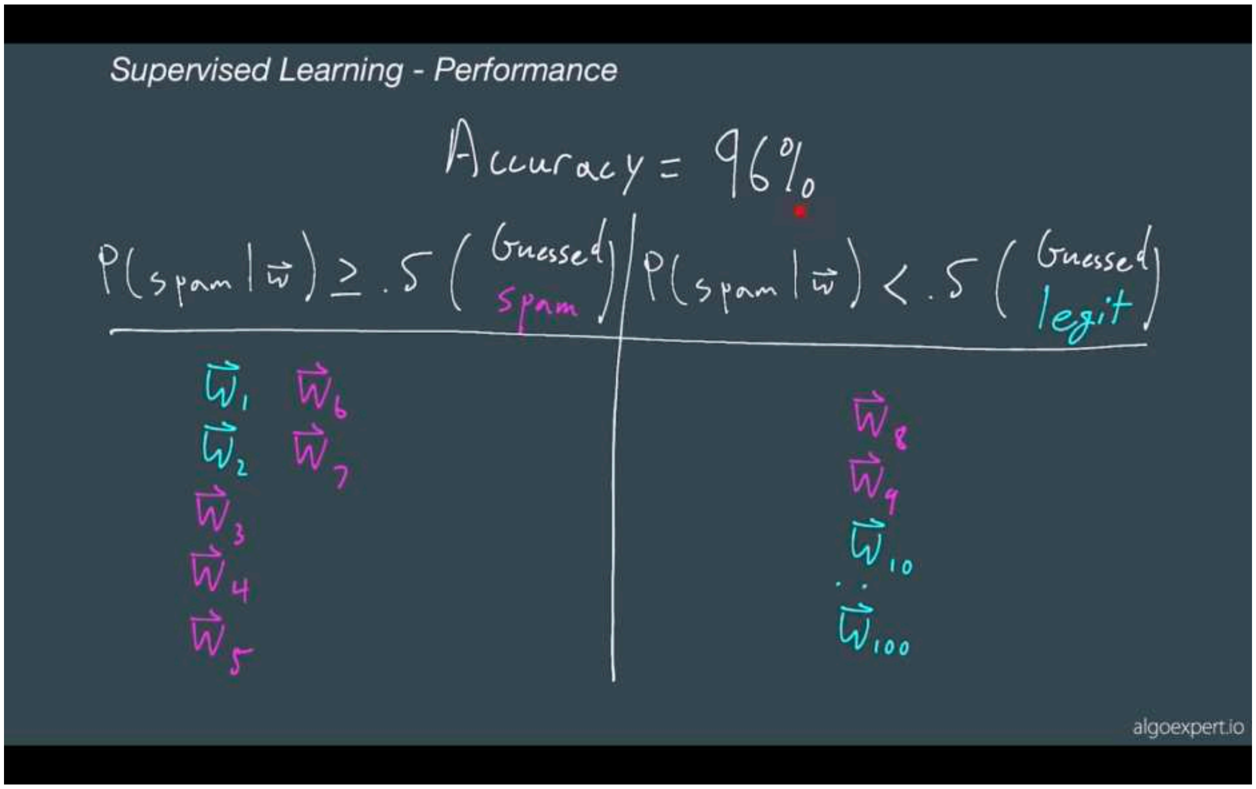
Now that we have the decision points figured out, let's start evaluating the performance of the filter rules and the model, by starting out with a very easy calculation called **accuracy**. For accuracy, all that we have to do is count up the number of spam messages that we guessed correctly. And then we count up the number of legitimate messages that we guessed correctly.


Supervised Learning - Performance

$$\text{Accuracy} = \frac{5 + 91}{100}$$

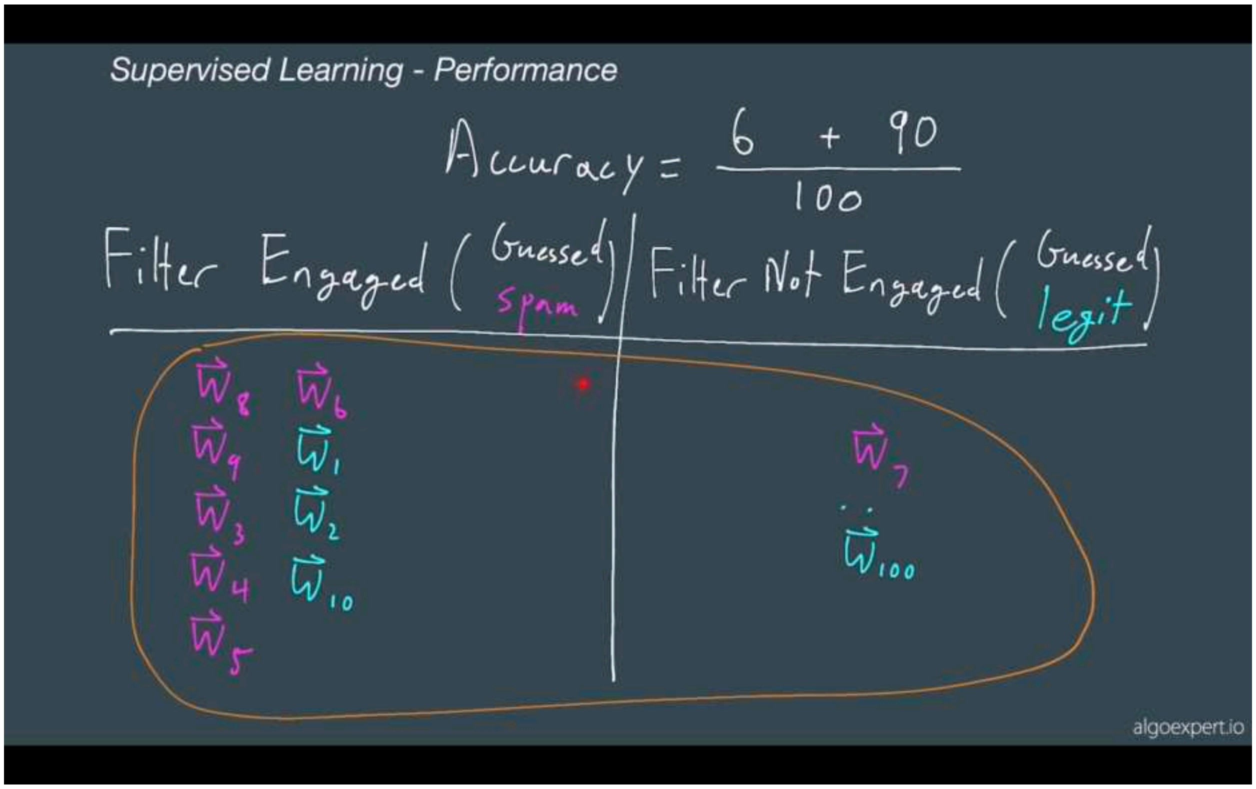
$P(\text{spam} \mid \vec{w}) \geq .5$ (Guessed spam)
$P(\text{spam} \mid \vec{w}) < .5$ (Guessed legit)
$\vec{w}_1$ $\vec{w}_2$ $\vec{w}_3$ $\vec{w}_4$ $\vec{w}_5$ $\vec{w}_6$ $\vec{w}_7$
$\vec{w}_8$ $\vec{w}_9$ $\vec{w}_{10}$ $\vec{w}_{100}$
algoexpert.io

We add those up and divide by the total number of messages. So in this case, the accuracy of the model across all of our examples is 96%.

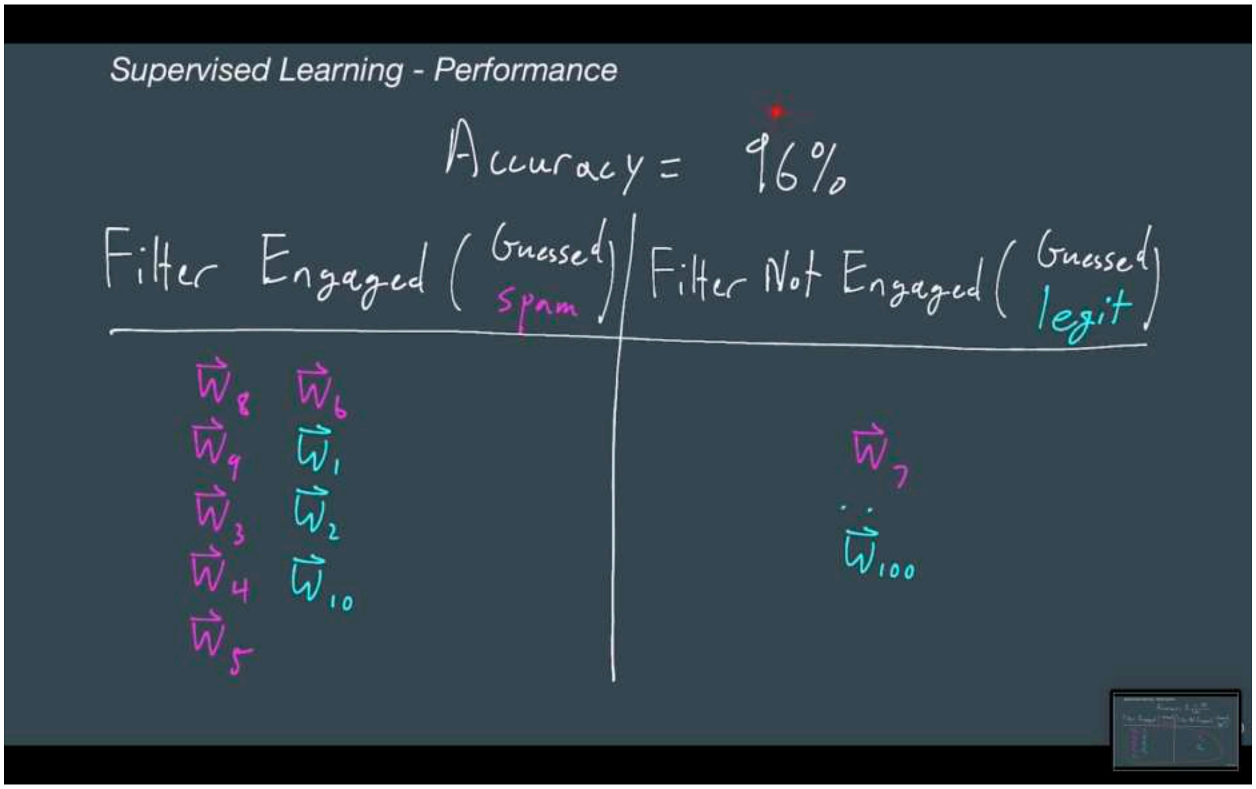


It's important to note here that these 100 messages are different from the messages that we trained on. So, these new 100 messages have never been seen by the model before. I think 96% accuracy is pretty good for our first model.

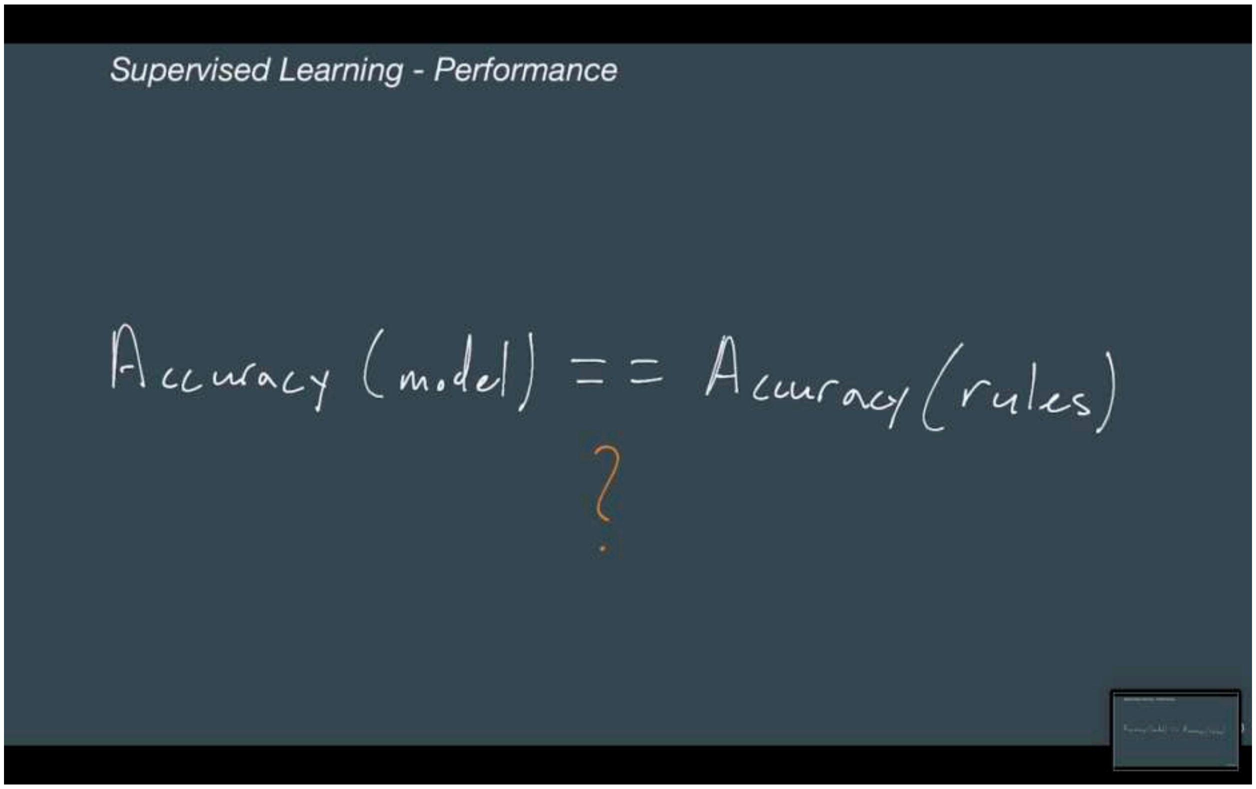
Let's see how well the **filter rules** do. Here we'll repeat the same process. We'll count the number of spam messages, that the filter rules correctly filtered as spam.


Supervised Learning - Performance
$\text{Accuracy} = \frac{6 + 90}{100}$
Filter Engaged (Guessed spam)
Filter Not Engaged (Guessed legit)
$\vec{W}_8$ $\vec{W}_6$
$\vec{W}_9$ $\vec{W}_1$
$\vec{W}_3$ $\vec{W}_2$
$\vec{W}_4$ $\vec{W}_{10}$
$\vec{W}_5$
$\vec{W}_7$
$\vec{W}_{100}$
algoexpert.io

We'll add that to the number of messages that the filter correctly let through as legitimate messages, and then we'll divide by the total number of our messages.


Supervised Learning - Performance
Accuracy = 96%
Filter Engaged (Guessed spam)
Filter Not Engaged (Guessed legit)
$\vec{W}_8$ $\vec{W}_6$
$\vec{W}_9$ $\vec{W}_1$
$\vec{W}_3$ $\vec{W}_2$
$\vec{W}_4$ $\vec{W}_{10}$
$\vec{W}_5$
$\vec{W}_7$
$\vec{W}_{100}$

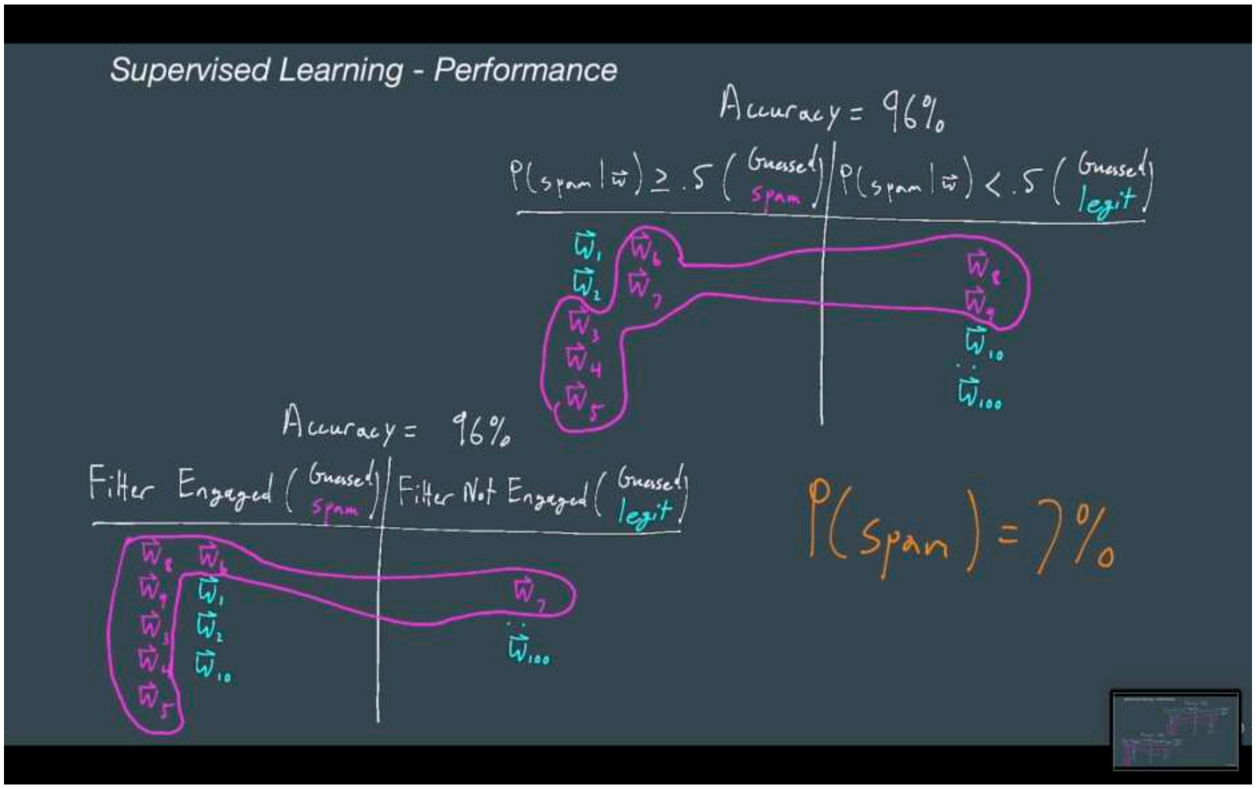
Here, we get a 90 wait, the accuracy of the filter rules and the accuracy of the model are the exact same. We went through all that trouble of probabilities and conditional probabilities, just to end up with the same exact accuracy as our very basic rules.


Supervised Learning - Performance

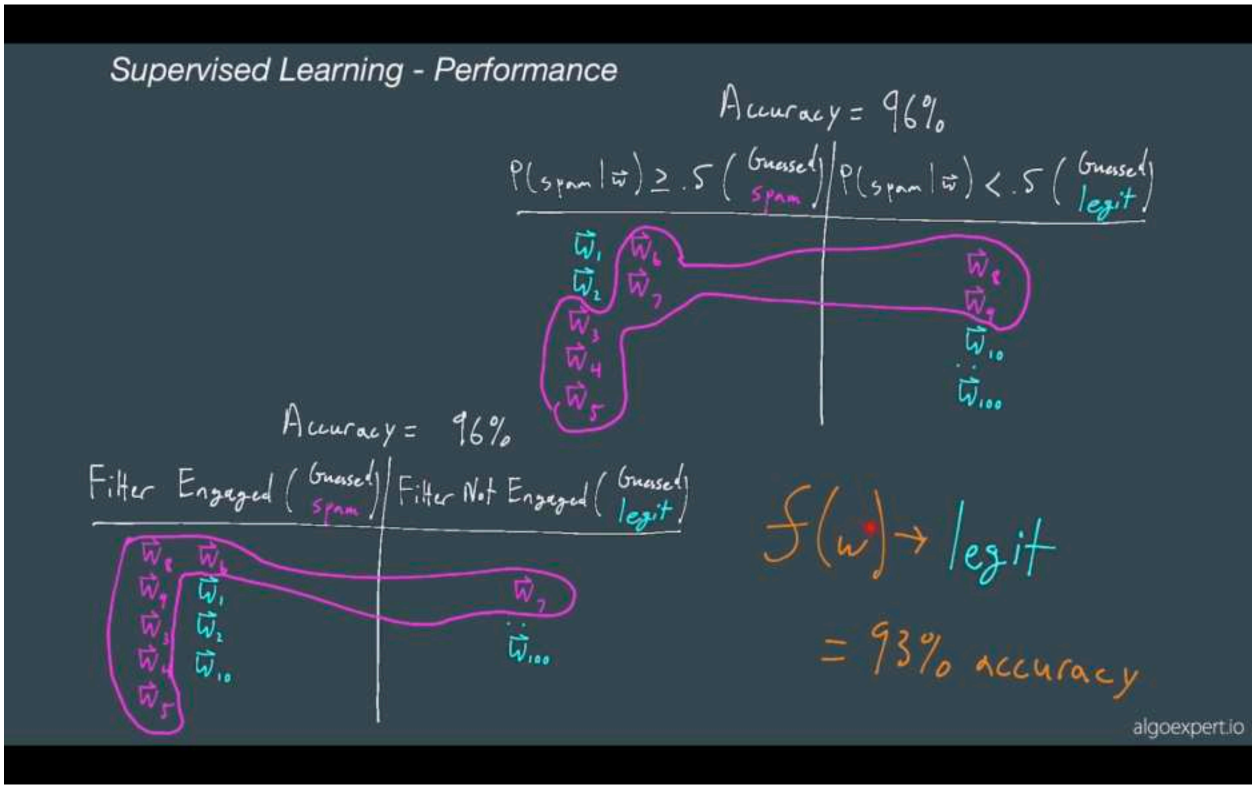
$$\text{Accuracy}(\text{model}) == \text{Accuracy}(\text{rules})$$

?

This is a really good question. Let's look at **why accuracy may not be such a great metric** to use in terms of measuring the performance of our model. One glaring issue that I see, is that we can abuse this accuracy metric, simply from the fact that the probability of any message being spam is just 7%.


Supervised Learning - Performance
Accuracy = 96%
P(spam | w) ≥ .5 (Guessed spam)
P(spam | w) < .5 (Guessed legit)
w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w100
Accuracy = 96%
Filter Engaged (Guessed spam)
Filter Not Engaged (Guessed legit)
P(spam) = 7%

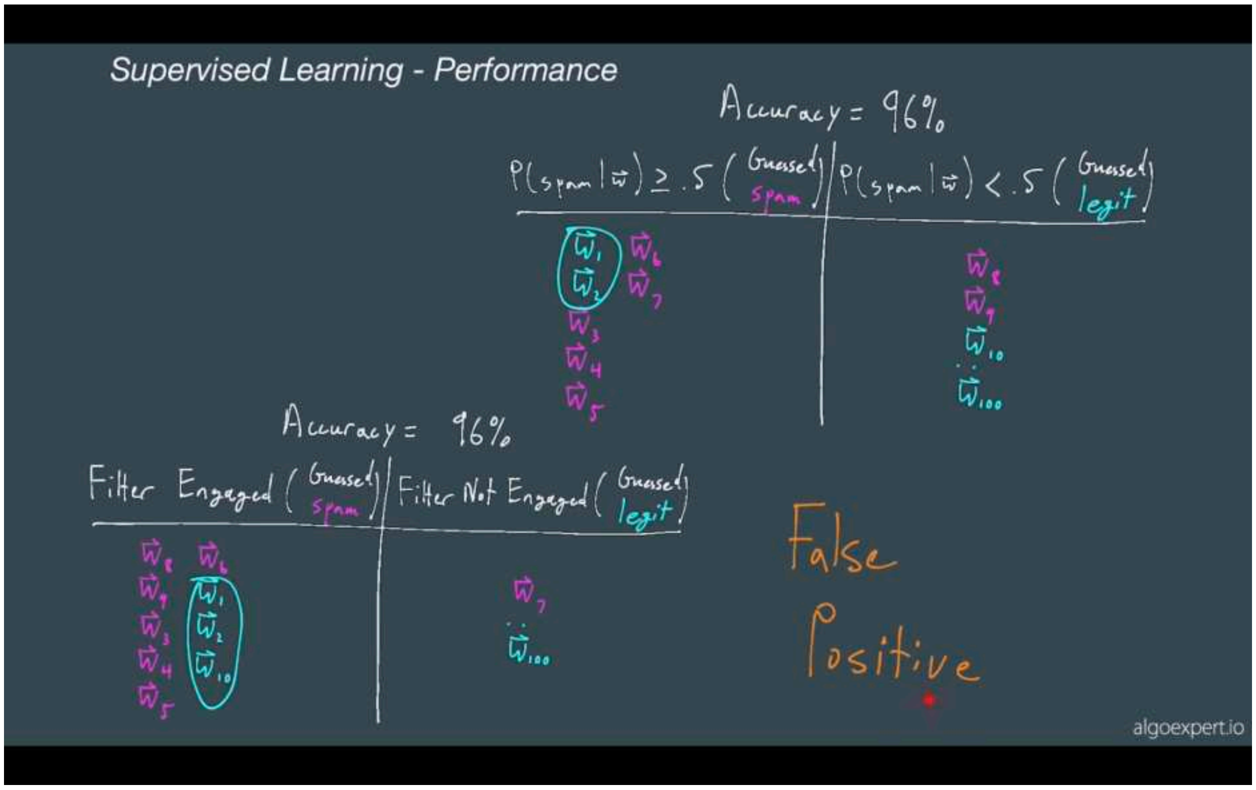
All the other messages are legitimate. This means that we could create a model which just takes in particular words, always guesses that the message is legitimate and it would come out with a 93% accuracy.


Supervised Learning - Performance
Accuracy = 96%
P(spam | w) ≥ .5 (Guessed spam)
P(spam | w) < .5 (Guessed legit)
Accuracy = 96%
Filter Engaged (Guessed spam)
Filter Not Engaged (Guessed legit)
f(w) → legit
= 93% accuracy
algoexpert.io

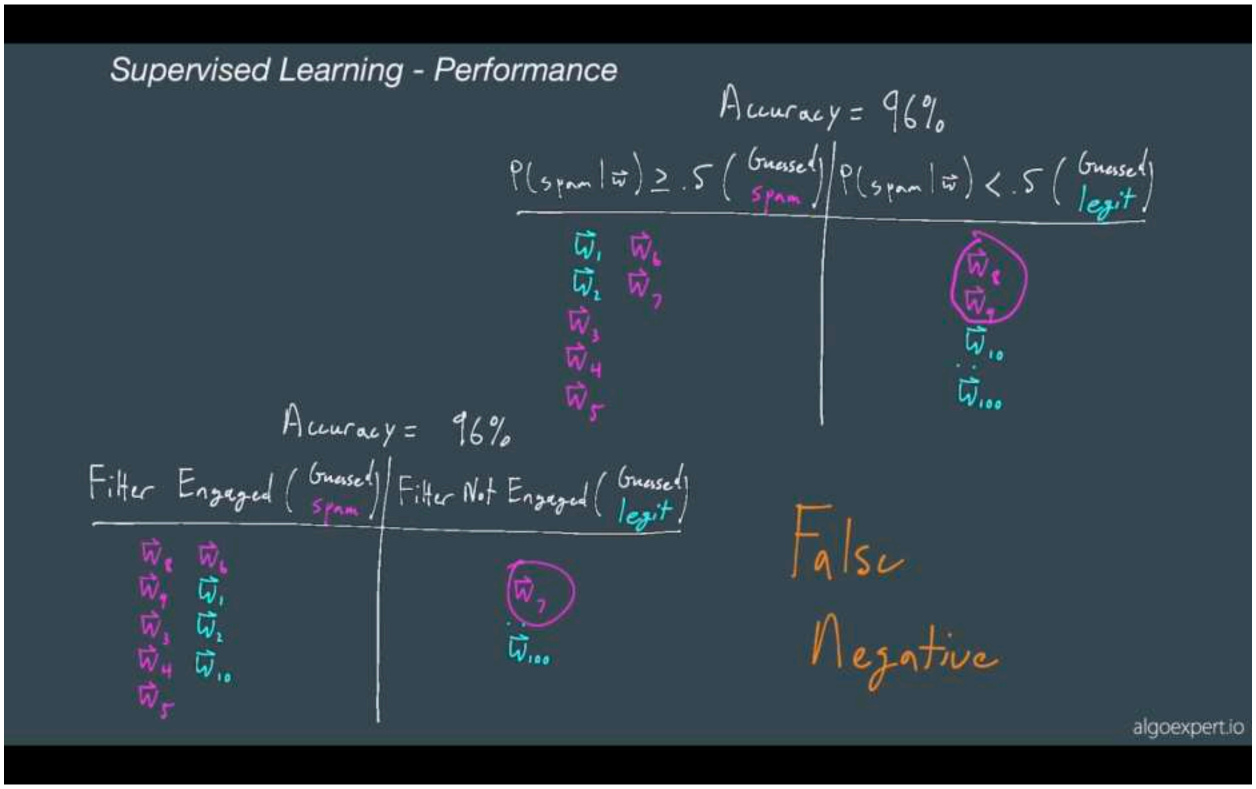
Even though this hypothetical model avoids the entire point of being a spam filter, it looks like it does pretty well in terms of accuracy. Later we'll talk about how **imbalanced classes** can actually affect the way that we need to train certain models.

But in this video, we'll just go over the performance implications when working with imbalanced classes. One problem with accuracy is that it doesn't penalize classifying non-spam as spam.

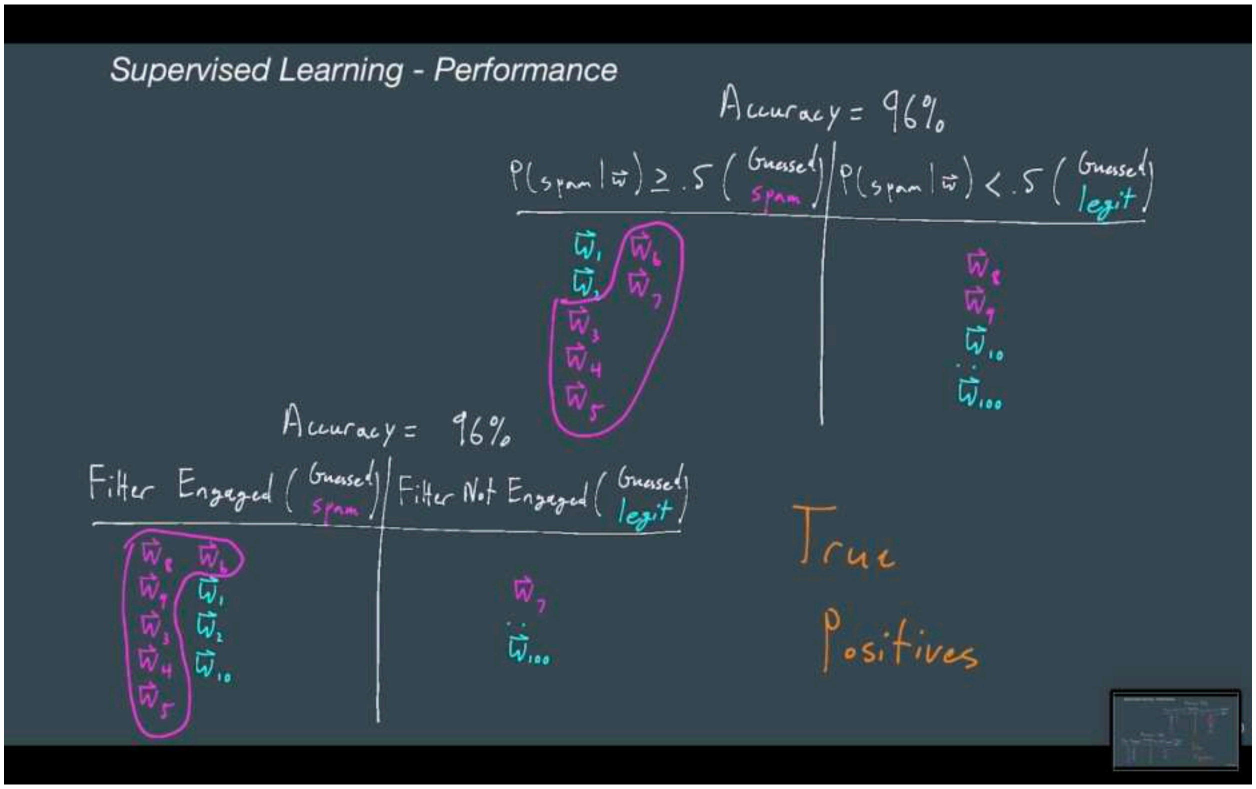
For instance, here w1 and w2 were legitimate messages but they got guesed to be spam for the model. Here the filter rules guesed three messages to be spam. When in fact they were not spam. This is technically called a false positive.


Supervised Learning - Performance
Accuracy = 96%
$P(\text{spam} \mid \vec{w}) \geq .5$ (Guessed spam)
$P(\text{spam} \mid \vec{w}) < .5$ (Guessed legit)
$\vec{w}_1$ $\vec{w}_2$ $\vec{w}_3$ $\vec{w}_4$ $\vec{w}_5$ $\vec{w}_6$ $\vec{w}_7$
$\vec{w}_8$ $\vec{w}_9$ $\vec{w}_{10}$ $\vec{w}_{100}$
Accuracy = 96%
Filter Engaged (Guessed spam)
Filter Not Engaged (Guessed legit)
$\vec{w}_8$ $\vec{w}_9$ $\vec{w}_3$ $\vec{w}_4$ $\vec{w}_5$ $\vec{w}_6$ $\vec{w}_1$ $\vec{w}_2$ $\vec{w}_{10}$
$\vec{w}_7$ $\vec{w}_{100}$
False Positive
algoexpert.io

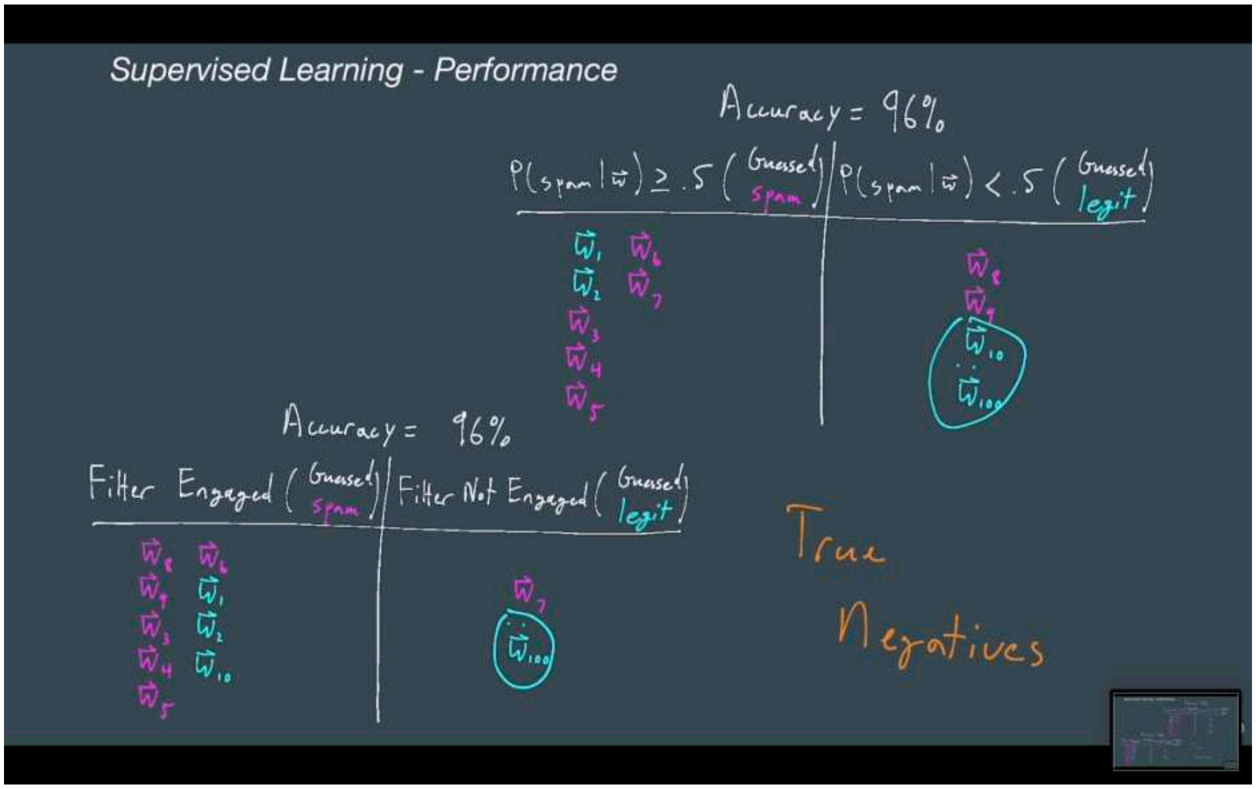
Now messages that are spam but are classified to not be spam, result in false negatives.


Supervised Learning - Performance
Accuracy = 96%
P(spam | w) ≥ .5 (Guessed spam)
P(spam | w) < .5 (Guessed legit)
Accuracy = 96%
Filter Engaged (Guessed spam)
Filter Not Engaged (Guessed legit)
False Negative
algoexpert.io

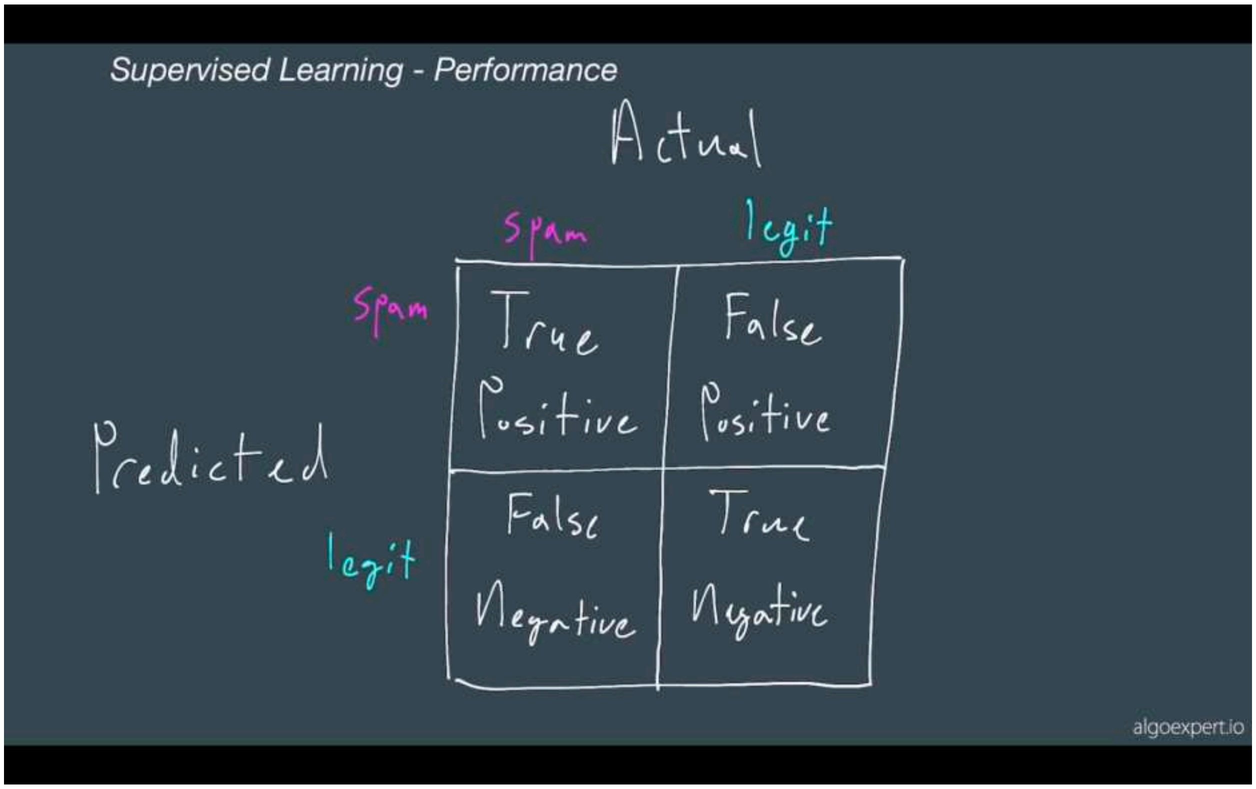
Now true positives are when we actually guess messages to be spam when they are,


Supervised Learning - Performance
Accuracy = 96%
P(spam | w) ≥ .5 (Guessed spam)
P(spam | w) < .5 (Guessed legit)
Accuracy = 96%
Filter Engaged (Guessed spam)
Filter Not Engaged (Guessed legit)
True Positives

and true negatives are when we guess messages to be legitimate when they actually are.


Supervised Learning - Performance
Accuracy = 96%
P(spam | w) ≥ .5 (Guessed spam)
P(spam | w) < .5 (Guessed legit)
Accuracy = 96%
Filter Engaged (Guessed spam)
Filter Not Engaged (Guessed legit)
True Negatives

Accuracy only takes into account true negatives and true positives. All of these terms can be grouped into what's called a confusion matrix.

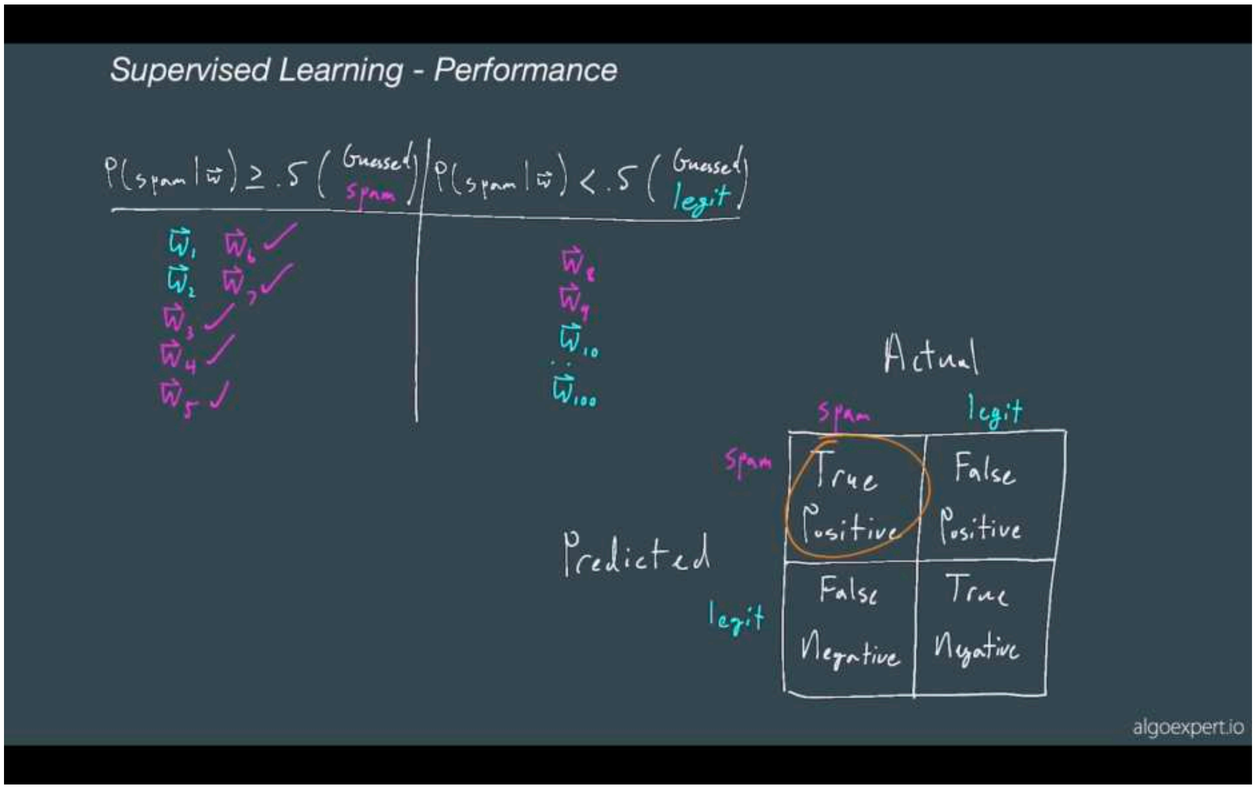


Supervised Learning - Performance

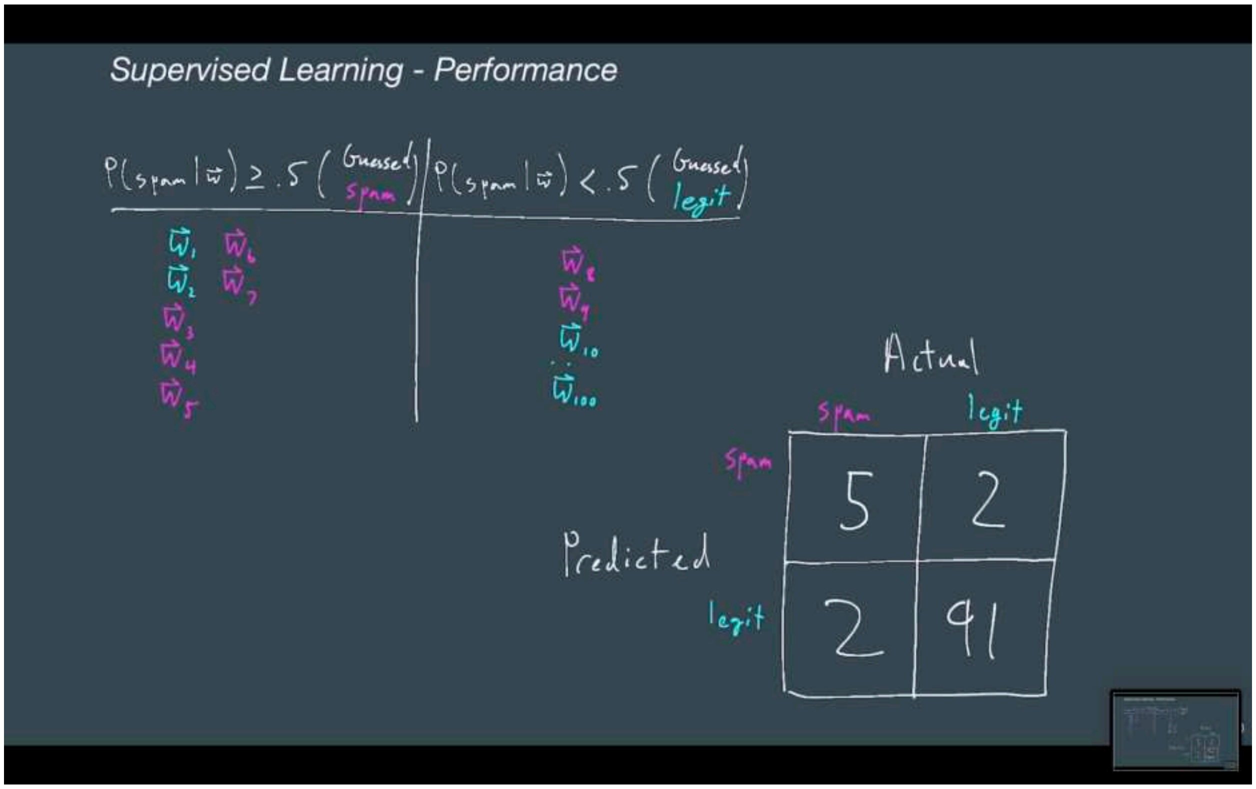
		Actual	
		spam	legit
Predicted	Spam	True Positive	False Positive
	legit	False Negative	True Negative

algoexpert.io

So here, starting from the top left quadrant, we have true positive. That means that we predicted the message to be spam when it actually was spam. A false positive is when we predicted a message to be spam when it was actually a legitimate message. I think we get the idea.



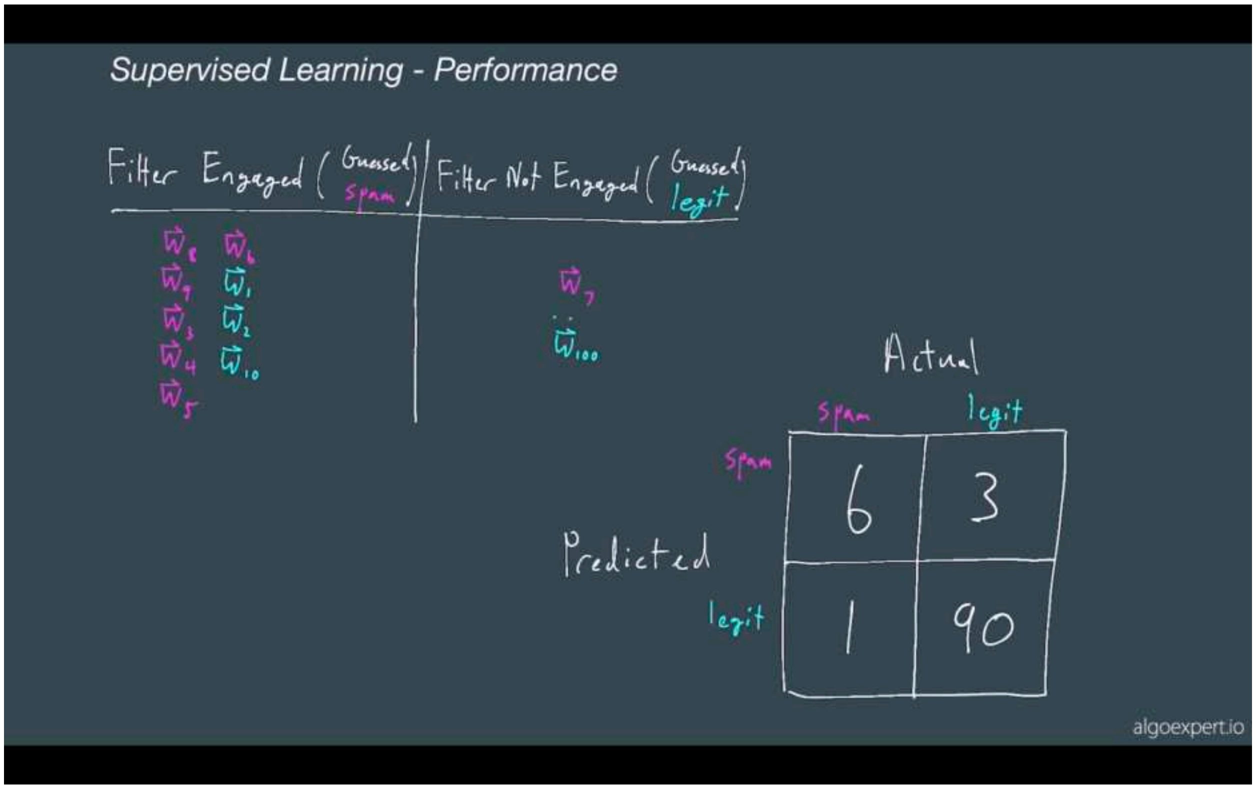
So, let's apply this to our model and filter rules. So, here we're going to solve for true positives first. This means we're just going to count the number of spam messages that were correctly categorized here, that's five. Then we'll move to the number of false positives here, that's two. Next up is false negatives here, that's two as well. And finally our true negatives is 91.



Supervised Learning - Performance

		Actual	
		spam	legit
Predicted	spam	5	2
	legit	2	91

So, this is the confusion in matrix for our model. Let's repeat the same process before our filter rules. Here we had six true positives. We had 92 negatives. We had three false positives and one false negative.



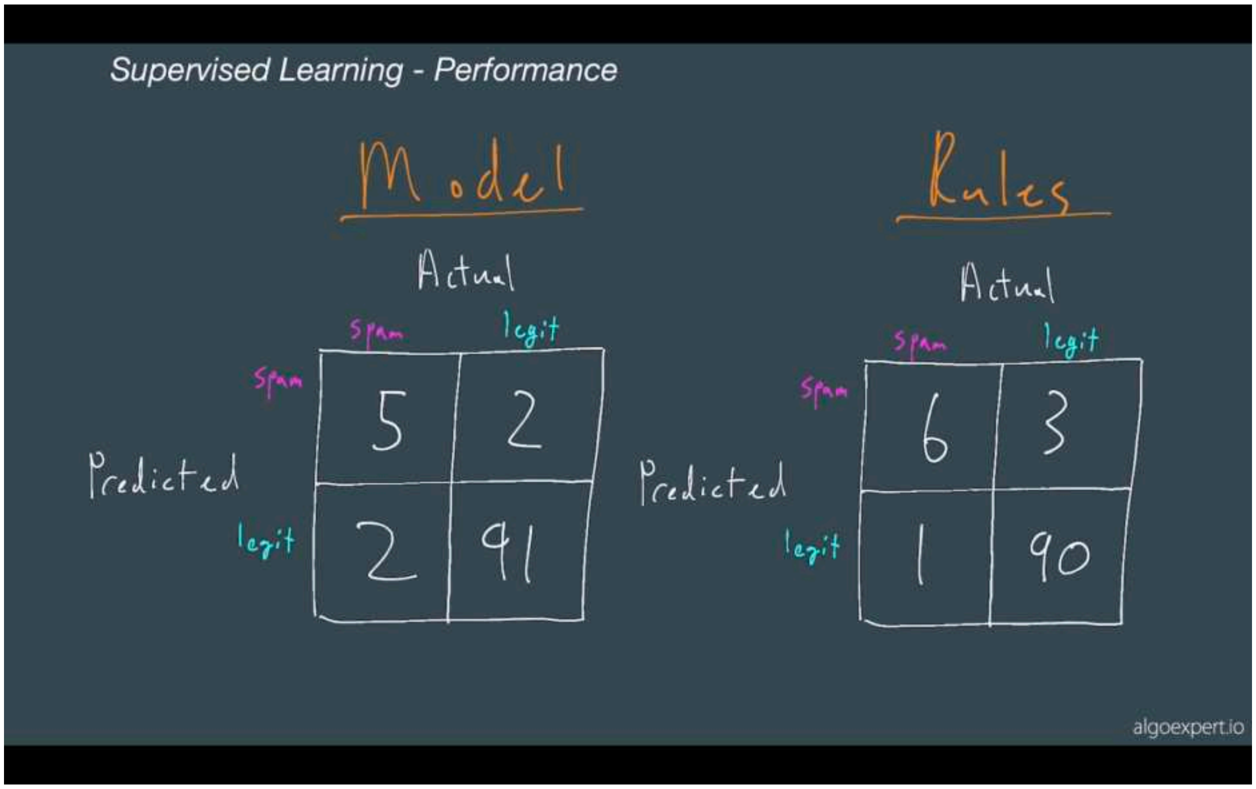
Supervised Learning - Performance

Filter Engaged (Guessed spam)	Filter Not Engaged (Guessed legit)
$\vec{w}_8$ $\vec{w}_6$	$\vec{w}_7$
$\vec{w}_9$ $\vec{w}_1$	$\vdots$
$\vec{w}_3$ $\vec{w}_2$	$\vec{w}_{100}$
$\vec{w}_4$ $\vec{w}_{10}$	
$\vec{w}_5$	

		Actual	
		spam	legit
Predicted	spam	6	3
	legit	1	90

algoexpert.io

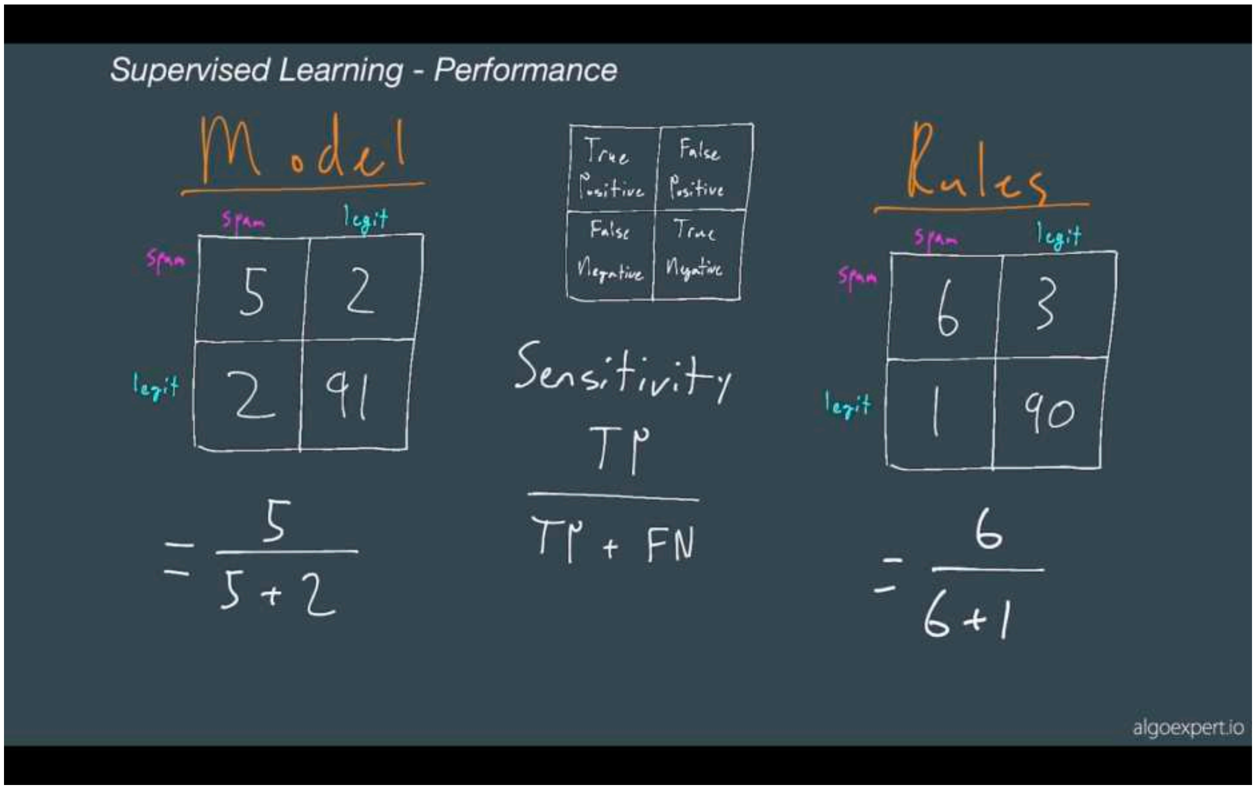
So, now we have two confusion matrices. We have one for the model and one for the rules. This should be all the information that we need in determining the performance of each. But I still feel like, I don't know which one is better.



If we had something like this, for instance, the decision would be pretty clear. Here the rules only correctly classified 61 out of the 100 messages. Well here, the model would have correctly classified 99 messages. But in our case, we're not so lucky.

Both these models correctly classify the same number of messages. They also misclassify the same number of messages. Luckily, there are different metrics we can look at to help us out. Let's make room and first go over something called sensitivity.

Sensitivity is the true positives over the true positives plus false negatives.


Supervised Learning - Performance

	True Positive	False Positive
	False Negative	True Negative

Model:

	spam	legit
spam	5	2
legit	2	91

Rules:

	spam	legit
spam	6	3
legit	1	90

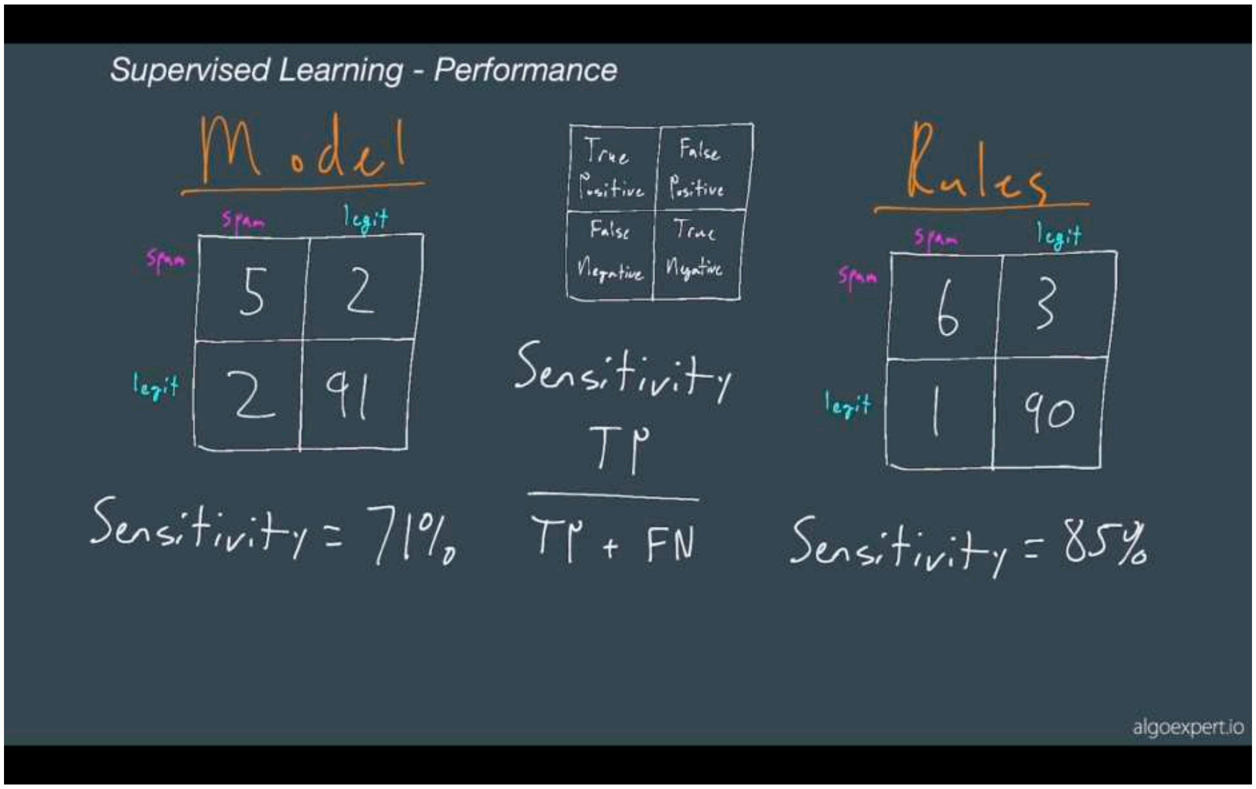
Sensitivity

$$\frac{TP}{TP + FN}$$

$$= \frac{5}{5+2}$$

$$= \frac{6}{6+1}$$

So, in each of these cases, the sensitivity is 71% and 85% respectively. Sensitivity for our case means the model's ability to correctly classify spam messages.



Supervised Learning - Performance

Model

	spam	legit
spam	5	2
legit	2	91

Sensitivity = 71%

True Positive	False Positive
False Negative	True Negative

Sensitivity

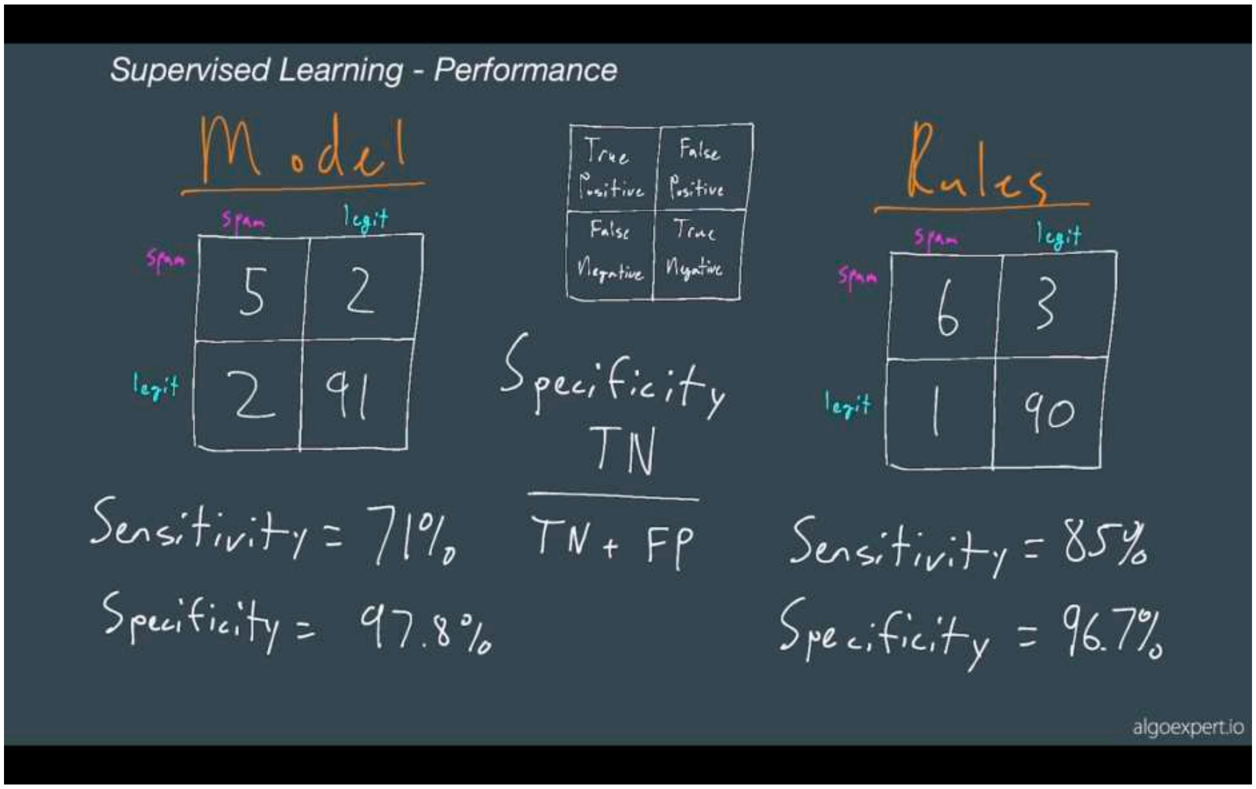
$$\frac{TP}{TP + FN}$$

Rules

	spam	legit
spam	6	3
legit	1	90

Sensitivity = 85%

Now it sounds bad because our model is actually worse than the rules at correctly classifying spam messages. It's not so bad because the specificity which is the true negatives over the true negatives plus false positives for the model case, is actually higher than the rules.



Supervised Learning - Performance

Model

	spam	legit
spam	5	2
legit	2	91

Sensitivity = 71%

Specificity = 97.8%

True Positive	False Positive
False Negative	True Negative

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Rules

	spam	legit
spam	6	3
legit	1	90

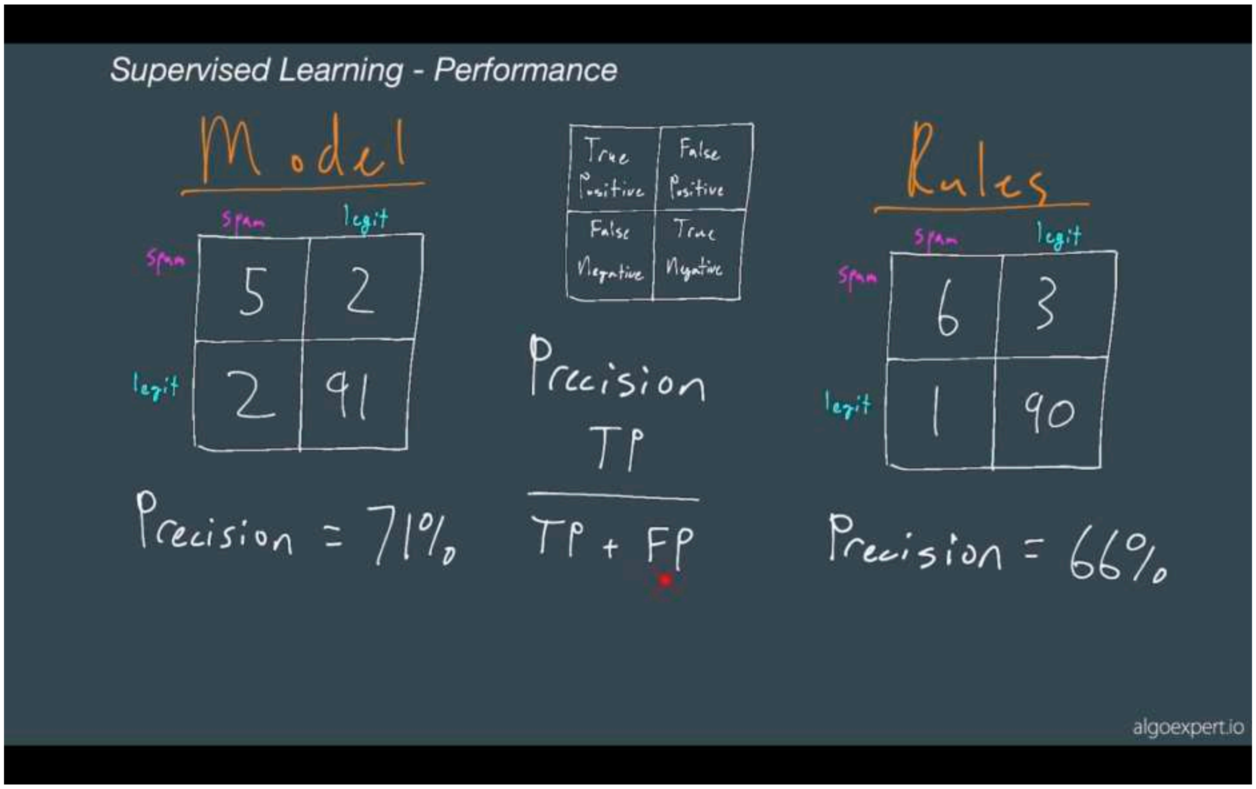
Sensitivity = 85%

Specificity = 96.7%

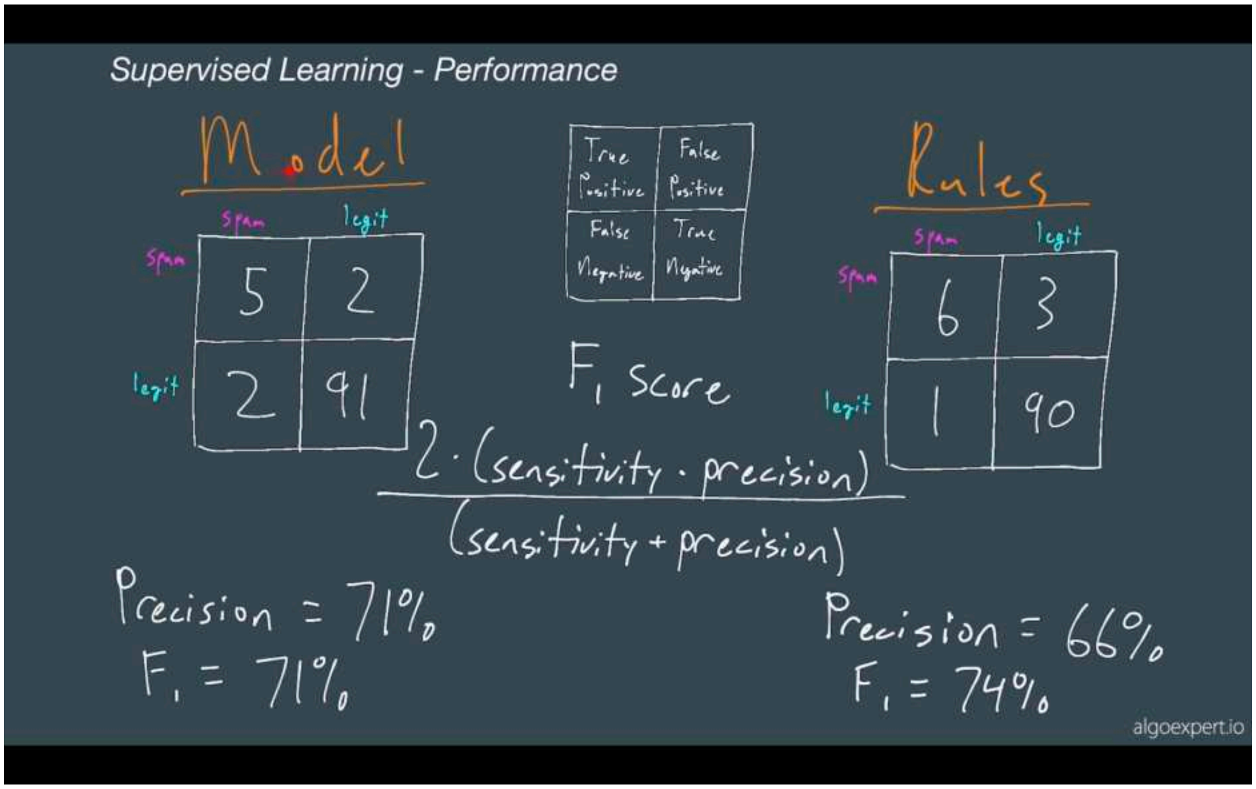
Here **specificity** represents the classifier's ability to correctly classify legitimate messages. With a higher specificity, you'll have fewer false positives and with a higher sensitivity, you'll have fewer false negatives. So, in the case of spam filter, I would actually prefer a higher specificity, such that an important message wouldn't be falsely classified as spam such that I may never see it. However, if we are reclassifying cancerous cells, I would want there to be as few false negatives as possible.

We would look generally for higher sensitivity. This really depends on your application. So, just use your judgment when it comes to selecting between models with higher sensitivities or specificities.

Now the next metric we're going to look at is called precision. It's the true positives divided by the true positives plus false positives.



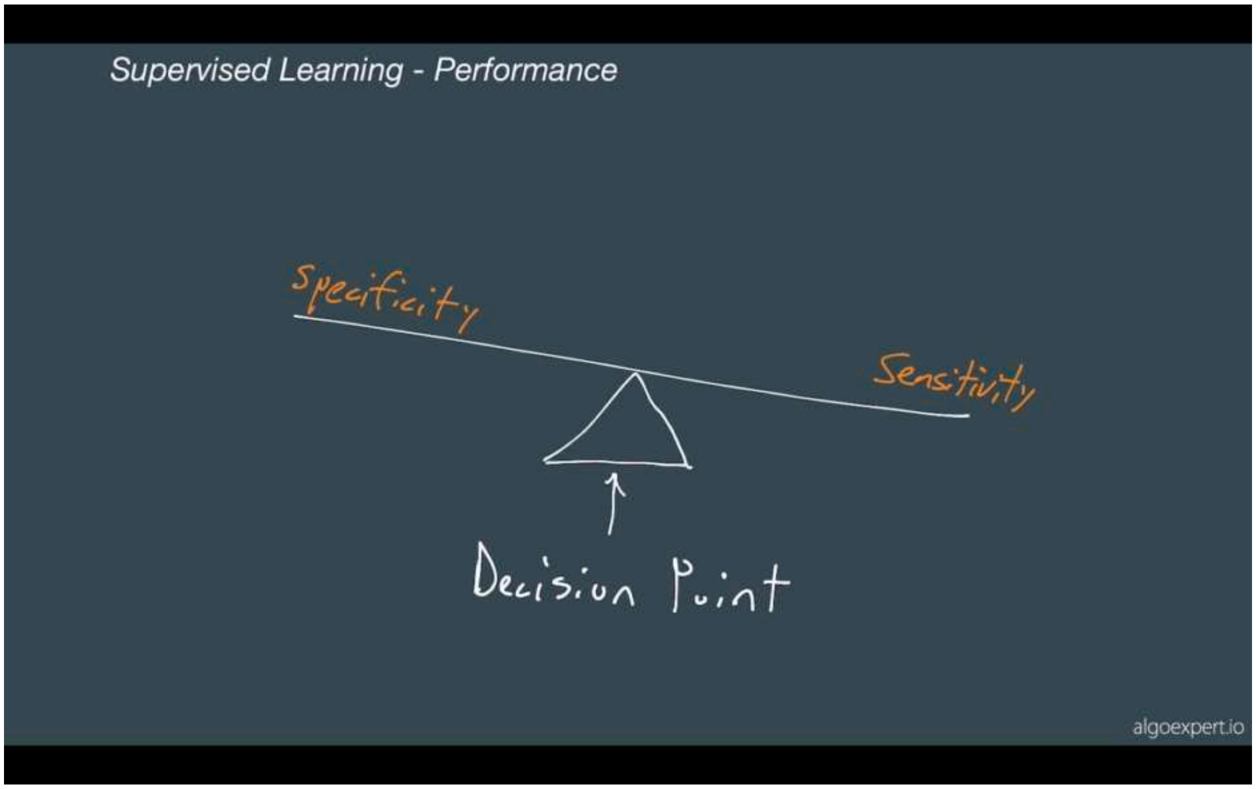
They quite literally asks out of every time, I've classified something as spam. How many of them actually were spam. Here we can see that our precision of our model is far higher than the precision of our rules. I think this is a really good sign for the model.



The final thing that we can look at is something called an F1 score. The formula is two times the sensitivity times the precision divided by the sensitivity plus the precision. For our model, since the sensitivity and the precision are the same, the F1 score is just 71%. However, the filter rules, F1 score is actually 74.
That's because the precision was 66 and the sensitivity was 85. The F1 score is actually just the harmonic mean of the sensitivity and the precision.

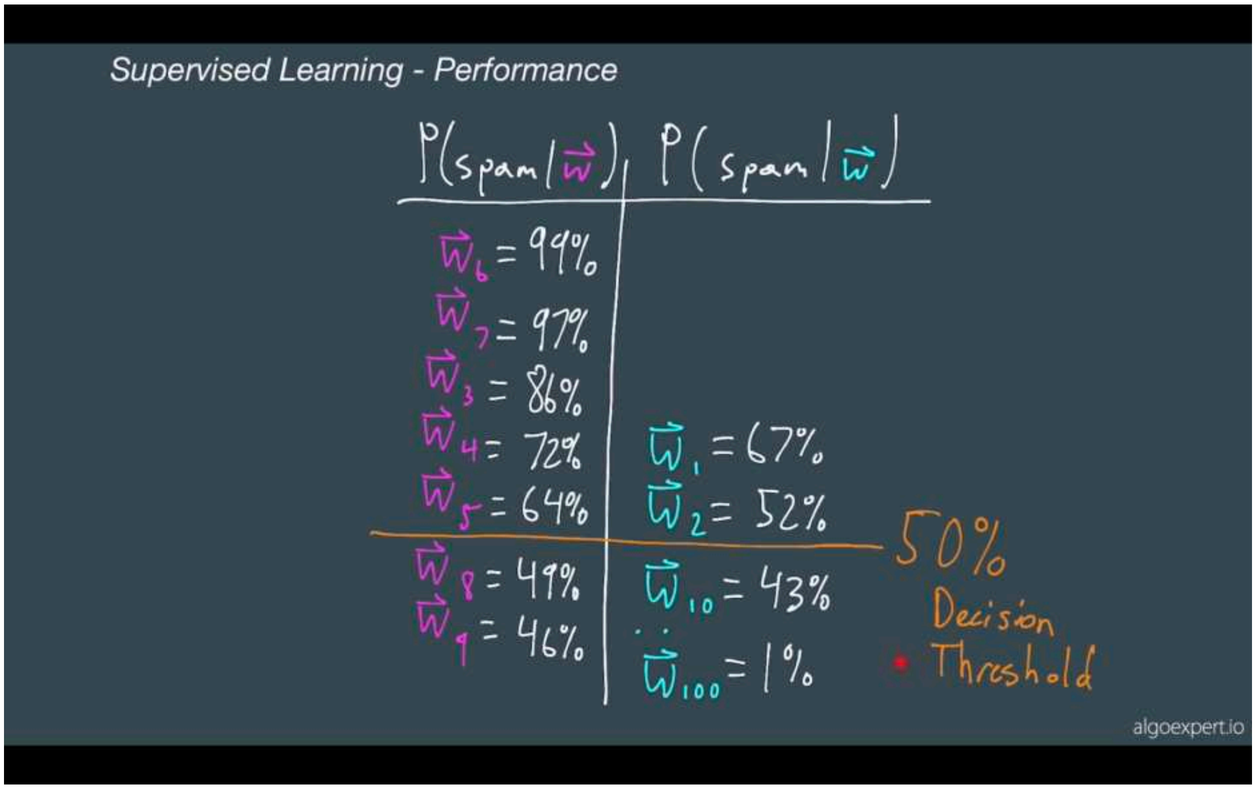
So, even though that this F1 score is a single number, **we actually lose a bit of information when it comes to describing why we would prefer our model over the filter rules.** Again, my main reason would be that the number of false positives is lower for the model than for the rules. Personally, this means I'm willing for more spam messages to be sent to me over me potentially missing important legitimate messages.

So, now this begs the question, can we control this trade-off of specificity and sensitivity? Remember a higher sensitivity generally means less false negatives and a higher specificity generally means less false positives.



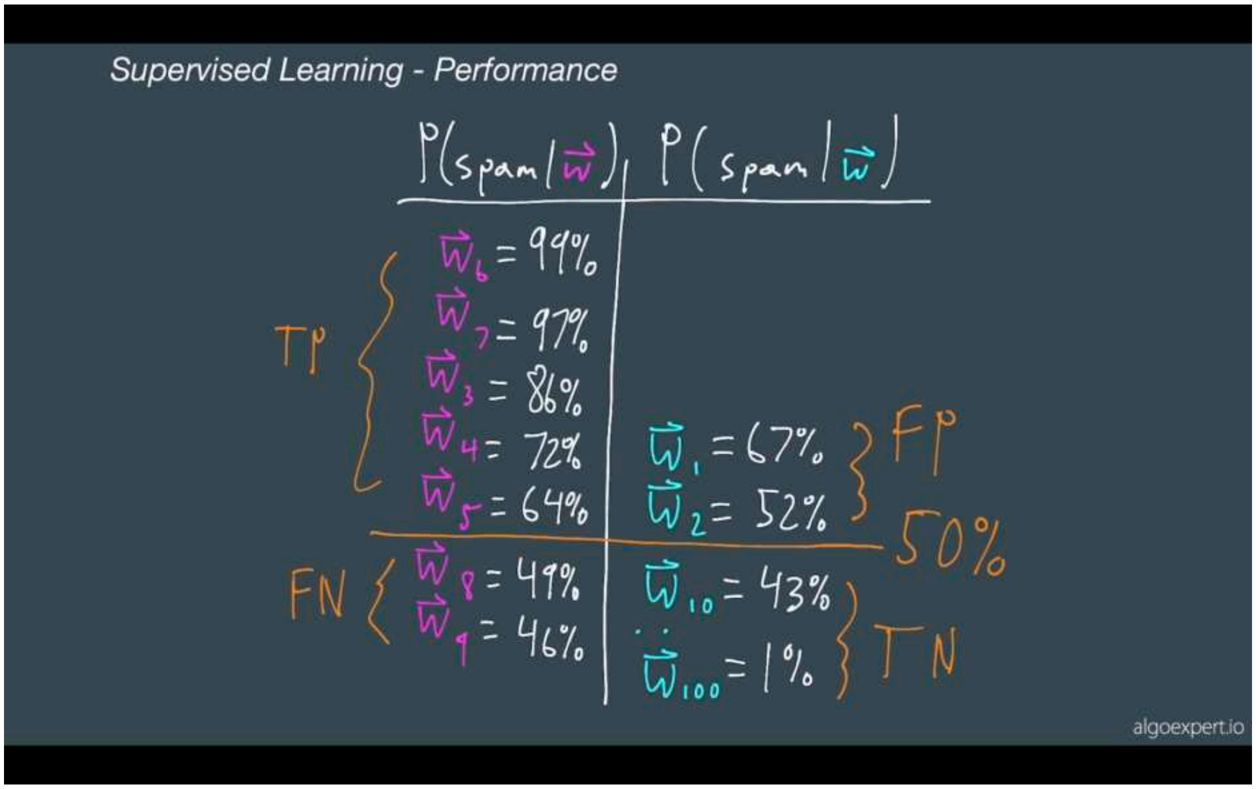
We can trade them off by adjusting our decision point or our **decision threshold**. Remember the decision threshold for our model was point five such that if the probability predicted for a particular message was 50% or greater, we'd classify it as spam or else we would classify it as not spam. Now, if we increase this threshold to 75%, we can imagine that fewer messages would be classified as spam. Likewise, if we lowered the threshold to 25%, we could imagine that more messages would be classified as spam.

This is because the probability requirement for a message to be deemed spam, is now less. Let's go over this in more detail. Here, we have two lists. On the left list, we have all of these spam messages and the predicted probability of that message being spam, according to our model.


Supervised Learning - Performance

$P(spam \mid \vec{w})$	$P(spam \mid \vec{w})$
$\vec{w}_6 = 99\%$	
$\vec{w}_7 = 97\%$	
$\vec{w}_3 = 86\%$	
$\vec{w}_4 = 72\%$	$\vec{w}_1 = 67\%$
$\vec{w}_5 = 64\%$	$\vec{w}_2 = 52\%$
$\vec{w}_8 = 49\%$	$\vec{w}_{10} = 43\%$
$\vec{w}_9 = 46\%$	$\vdots$ $\vec{w}_{100} = 1\%$

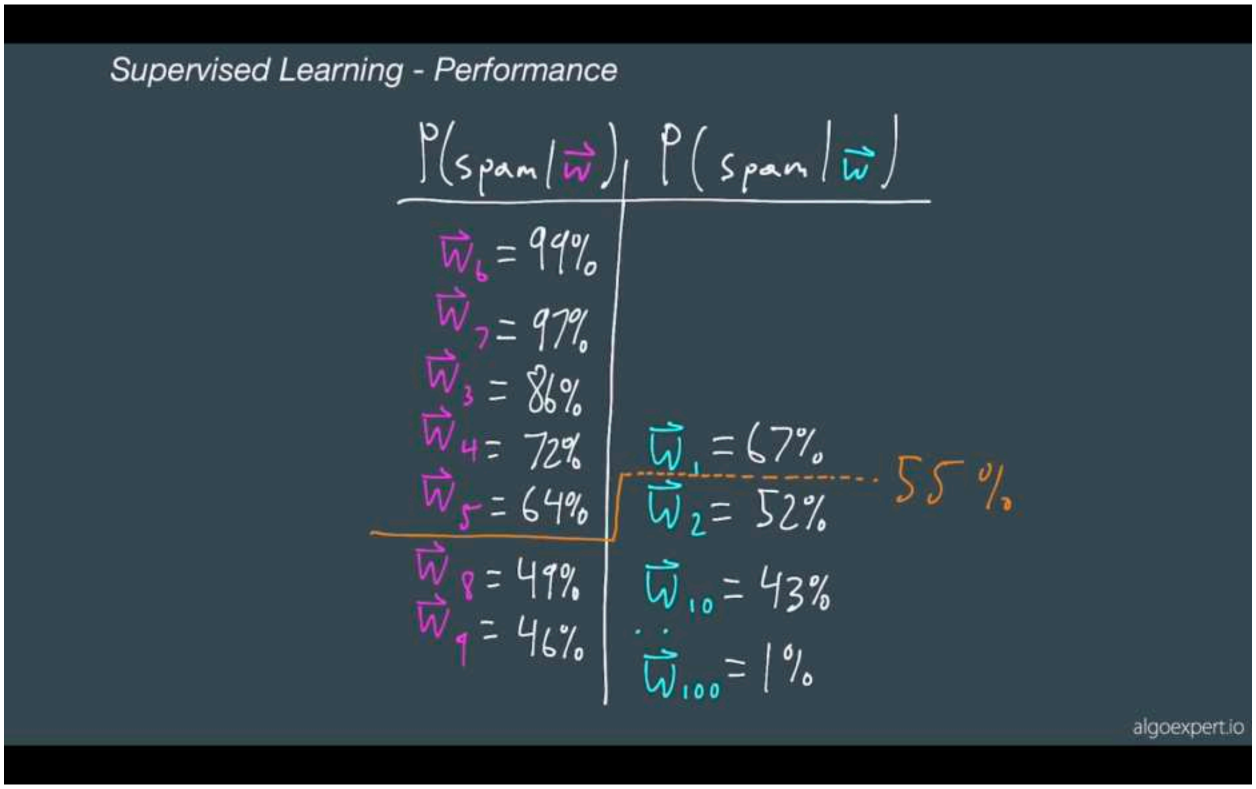
So, here message six, which has the label of spam, receives a 99% probability of being spam from our model. Likewise message nine which is also spam, only gets a 46% chance of being spam according to our model. On the right hand side, we have all of the legitimate messages and their respective probabilities of being spam according to our model. The interesting part happens right where our decision threshold is.



With our 50% decision threshold, we can see that some of these examples are on the wrong side of the threshold. For instance, these two spam messages got classified as not spam because of their probability of being spam, was less than the decision threshold. Likewise, these two messages here which were not spam or legitimate, they are classified as spam, because the probability of them being spam, according to our model with higher than our decision threshold.

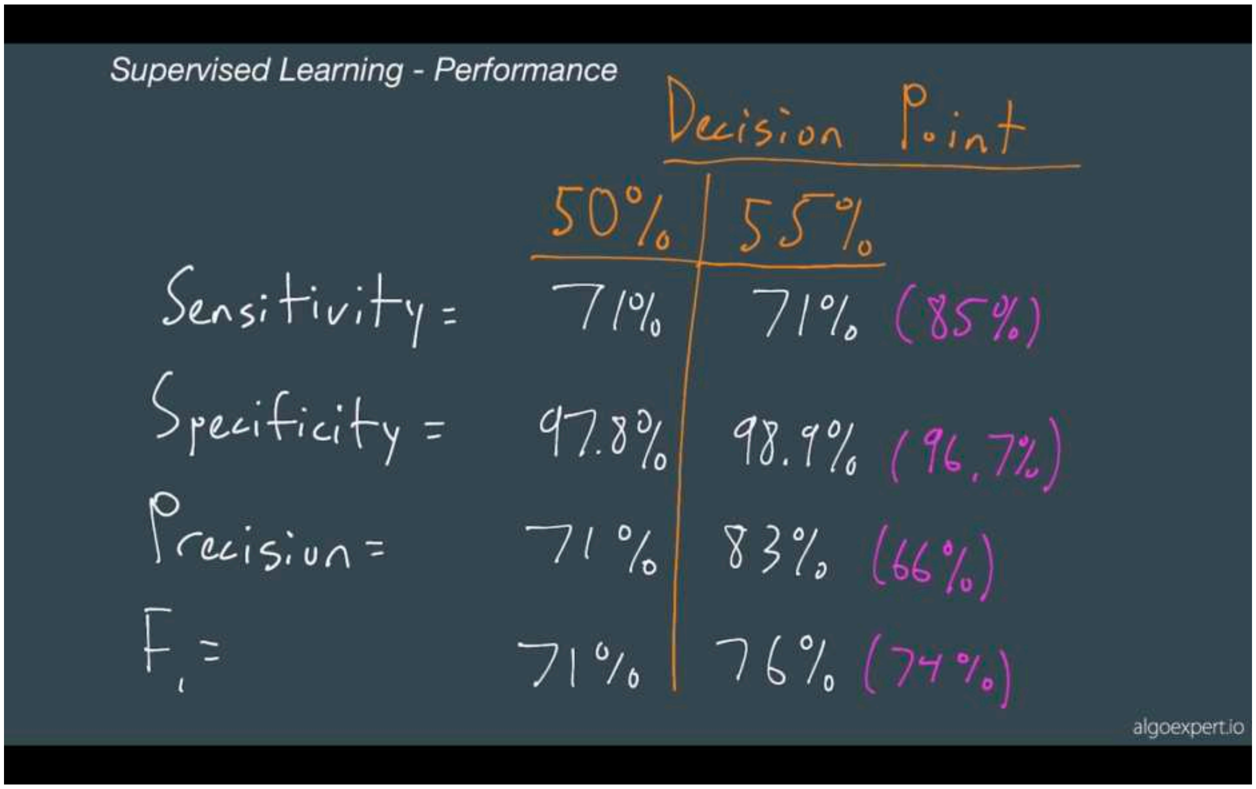
This directly reflects what we saw in our **confusion matrix**. So, here are the true positives, the false negatives, the false positives and the true negatives. I mentioned earlier that for me personally, I'd like a spam filter that would potentially send me more spam, as long as we decrease the chance, that a legitimate message gets classified as spam. And I end up never seeing it. This means that I want a higher specificity.

So, if we raise this 50% decision threshold to 55%, we actually correct this false positive because now our decision threshold is higher than the probability that this message was predicted to be spam.



So, in addition to now correcting this particular message, we also didn't affect any of the spam messages, but the number of true positives and false negatives are the same.

Let's see this affects our performance metrics we talked about earlier. Here, the sensitivity, specificity, precision and F1 are all listed from our 50% decision threshold. With our new decision threshold of 55%, our sensitivity didn't change, because the number of true positives and false negatives didn't change. The specificity went up because the number of true negatives went up and the false positives went down. Our increased precision means that we have a higher predictive value than we had before at our old threshold. Finally, since our precision changed, we got an updated F1 score. For reference here are the **metrics of our heuristic.**

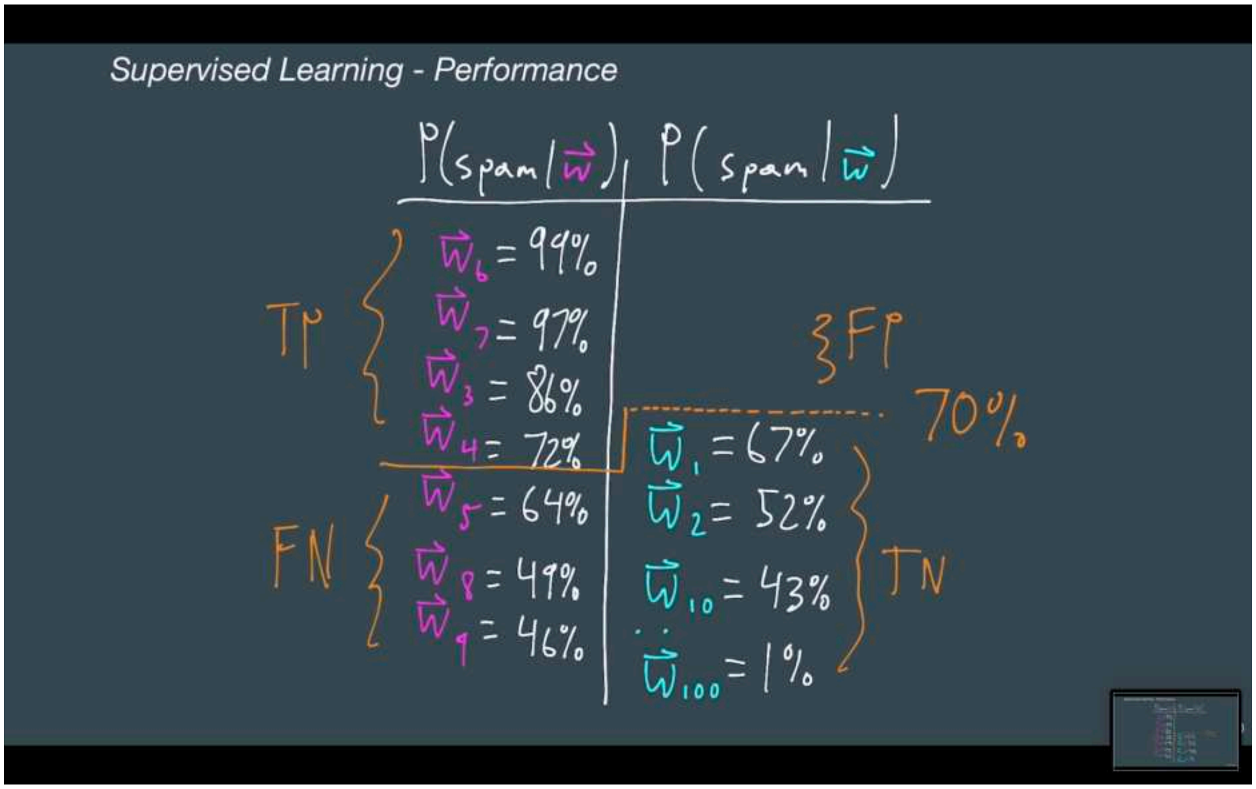


Supervised Learning - Performance

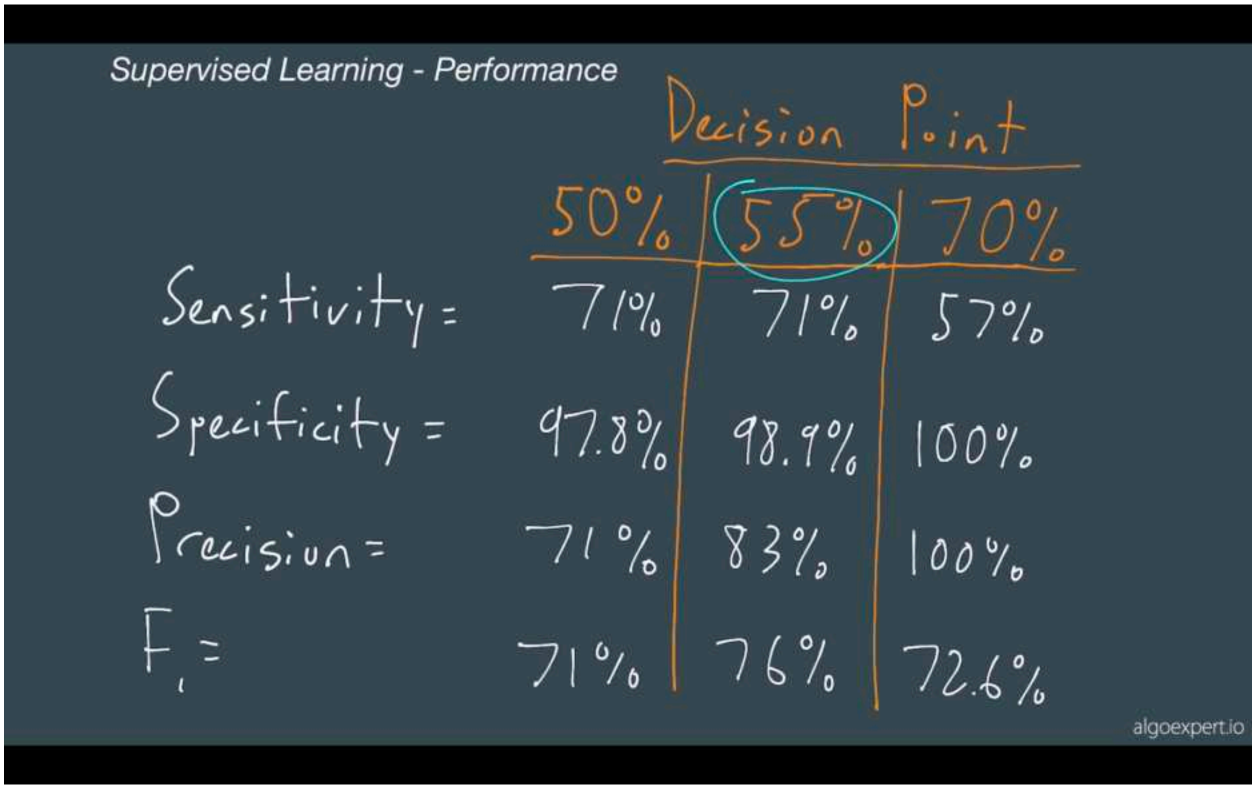
	Decision Point		
	50%	55%	
Sensitivity =	71%	71%	(85%)
Specificity =	97.8%	98.9%	(96.7%)
Precision =	71%	83%	(66%)
F_1 =	71%	76%	(74%)

algoexpert.io

So, the sensitivity was 85, specificity was 96, precision was 66 and our F1 score was 74. Fortunately, it's a little more clear now that our model at this new threshold now bids are heuristic in several metrics.



Let's see what happens when we try to get greedy and make this last message turn into a true negative. This would require a threshold of anything greater than 67%. And in our case, we're just gonna make it 70%. What this means now is that the number of false negatives has increased by one because w5 is now a member. The false positives have decreased by one, because message one is now correctly classified.



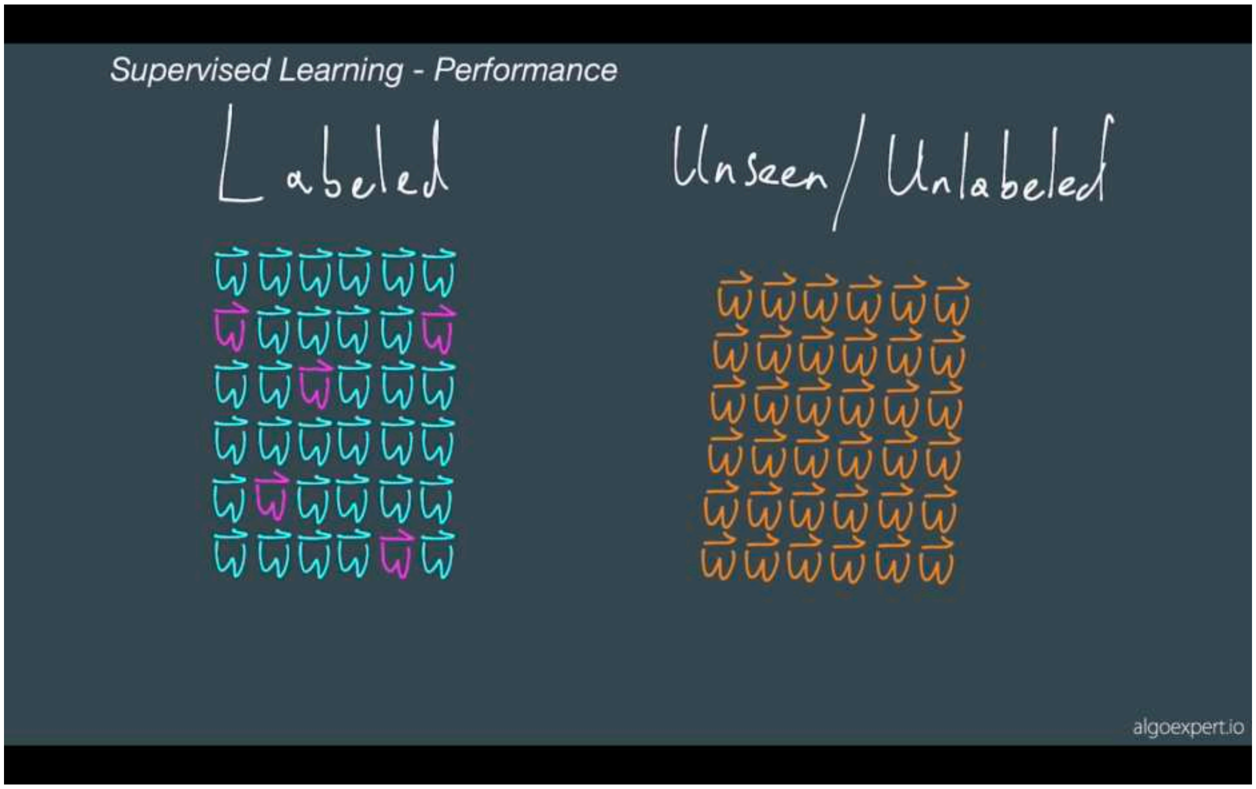
Supervised Learning - Performance

	Decision Point		
	50%	[✓ 55%]	70%
Sensitivity =	71%	71%	57%
Specificity =	97.8%	98.9%	100%
Precision =	71%	83%	100%
F_1 =	71%	76%	72.6%

algoexpert.io

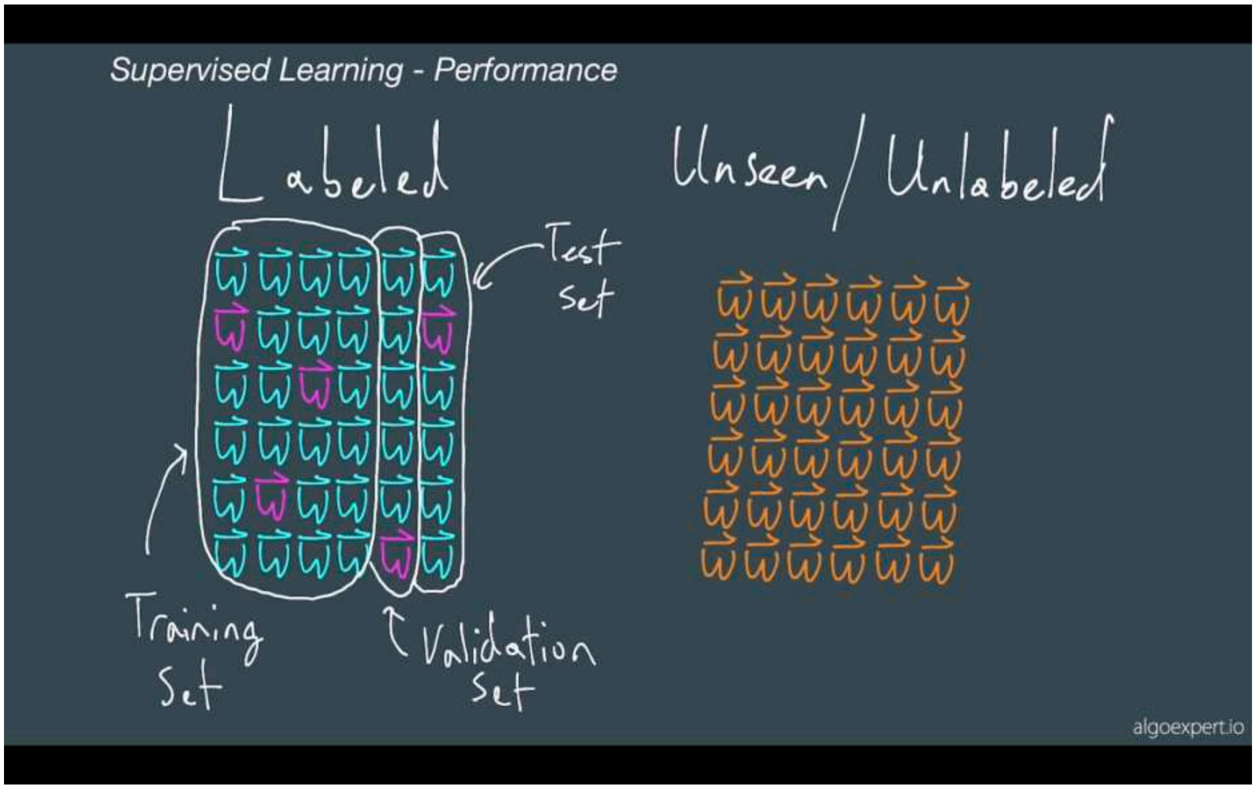
Let's see how this affects our metrics. Our sensitivity plummeted, our specificity and precision are 100% because the number of false positives is zero. And our F1 score decreased by a bit. The decision point that I would settle on is 55%. I would also now reasonably be confident to say that this model **beats the heuristic.**

This is a couple more things that we'll cover. **Remember** that the whole **goal of supervised learning** is to take labeled examples here spam or not spam messages, and use these to train a model and predict unseen or unlabeled data that we haven't trained on. To make the most out of these examples that we do have labeled, we need to split them up.



We'll have a training set, a validation set and a test set. Earlier we mentioned that the performance metrics we looked at, were generated from 100 new examples that we hadn't seen before or trained on.

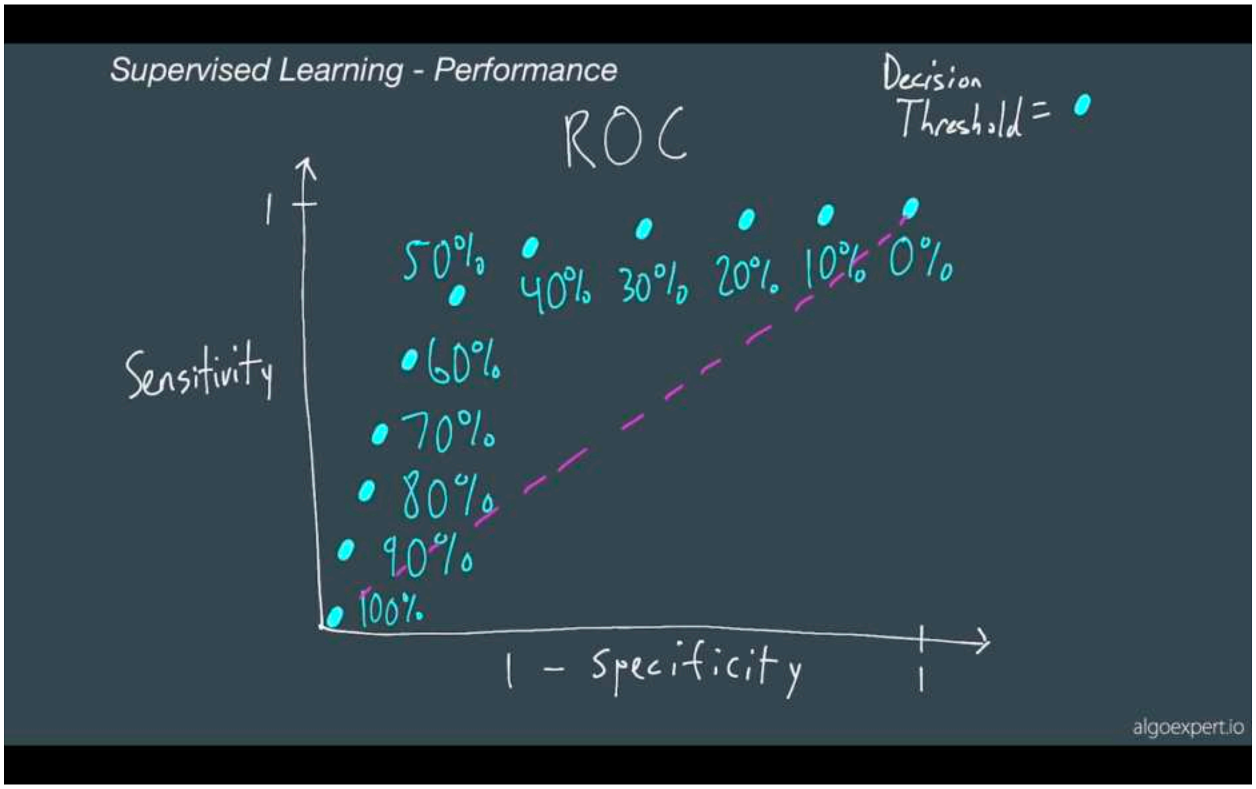
This is important because it means that we didn't use our training set to evaluate our model performance. Instead, we actually used a test set which is more important in determining if our model which was trained on these examples, will generalize or perform similarly to unseen or unlabeled examples.



So, we know about we need a training set to train on, and we need to test that so we can get some degree of confidence on whether or not our model will generalize to unseen or unlabeled examples. So, why is this validation set here? The validation set gives us the opportunity to **tune our model** without using the test set itself.

In this video, we actually tuned on the test set, which means that the tuning that we did, could have very easily not generalized to unseen examples 'cause we have no other labeled examples to run it against. In practice, we would have had a validation set, that we would have conducted our tuning on. And then once we were done tuning our model and by done I mean finished completely, we would then use our tests a single time to evaluate our model performance. This would be a better way to get an estimate of how our model will generalize to these unseen examples.

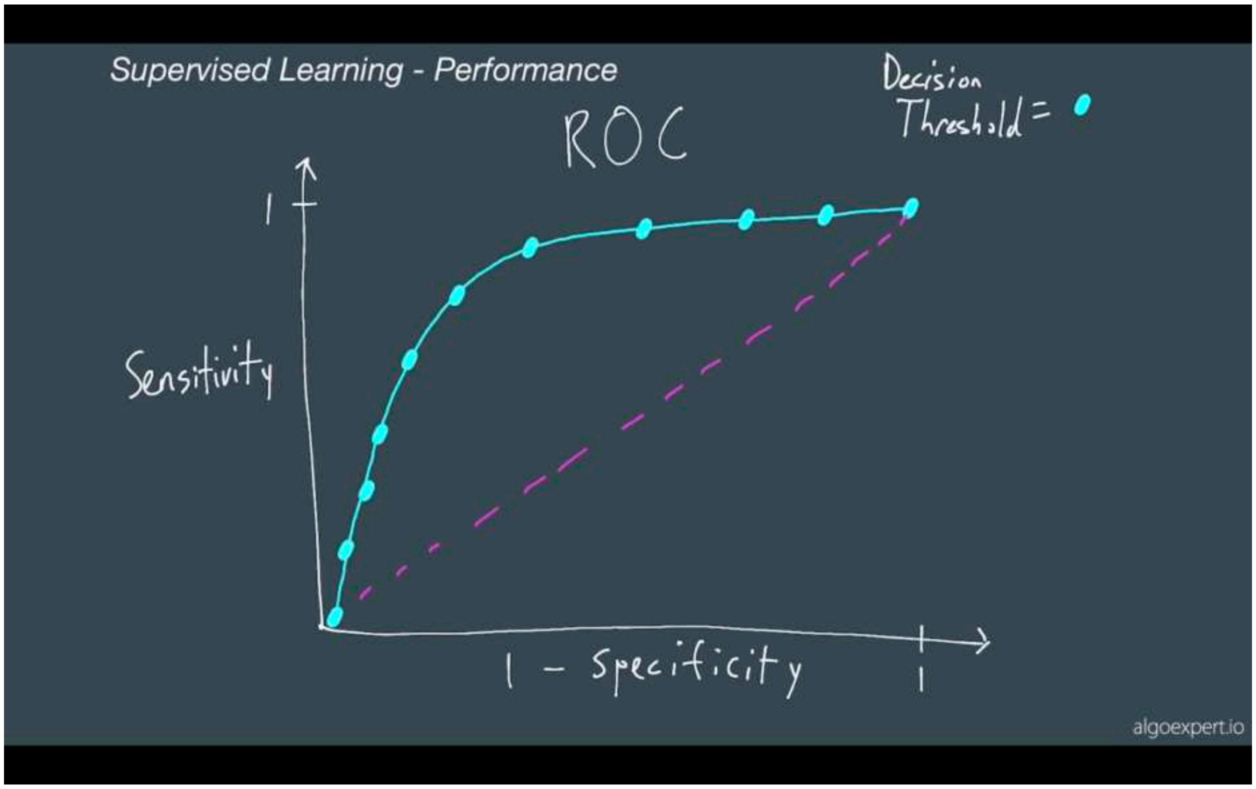
So, what can we actually tune with this validation set? Well, really anything. In our example, we would have tuned the validation set on this decision point. By the way, as we are tuning these different decision points we can actually plot something called a **Receiver Operating Characteristic curve.**



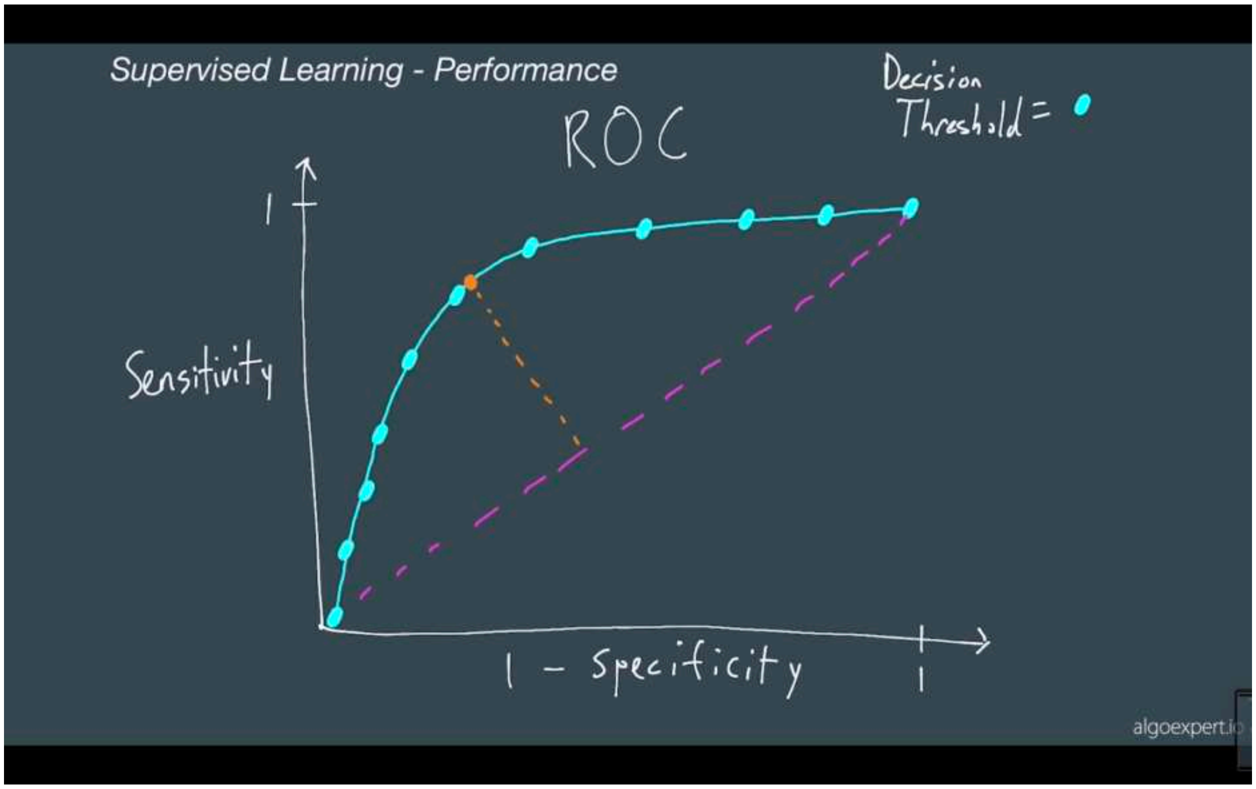
This ROC curve is plotted along the sensitivity on the y-axis, and the one minus specificity on the axis. As we tune our model on the validation set, we can actually plot the sensitivities and specificities that each decision threshold produces. In the top right the decision threshold of zero, means that we classified every single example as spam.

This means the sensitivity is one or 100%. However, this also means that our specificity is zero. The here one minus zero is also one, that gives us our point here for our decision threshold of zero. On the other end of the spectrum, here at the decision threshold of 100%, means that we classify every single example as legitimate. This means our sensitivity is equal to zero. Well, our specificity is equal to 100%.

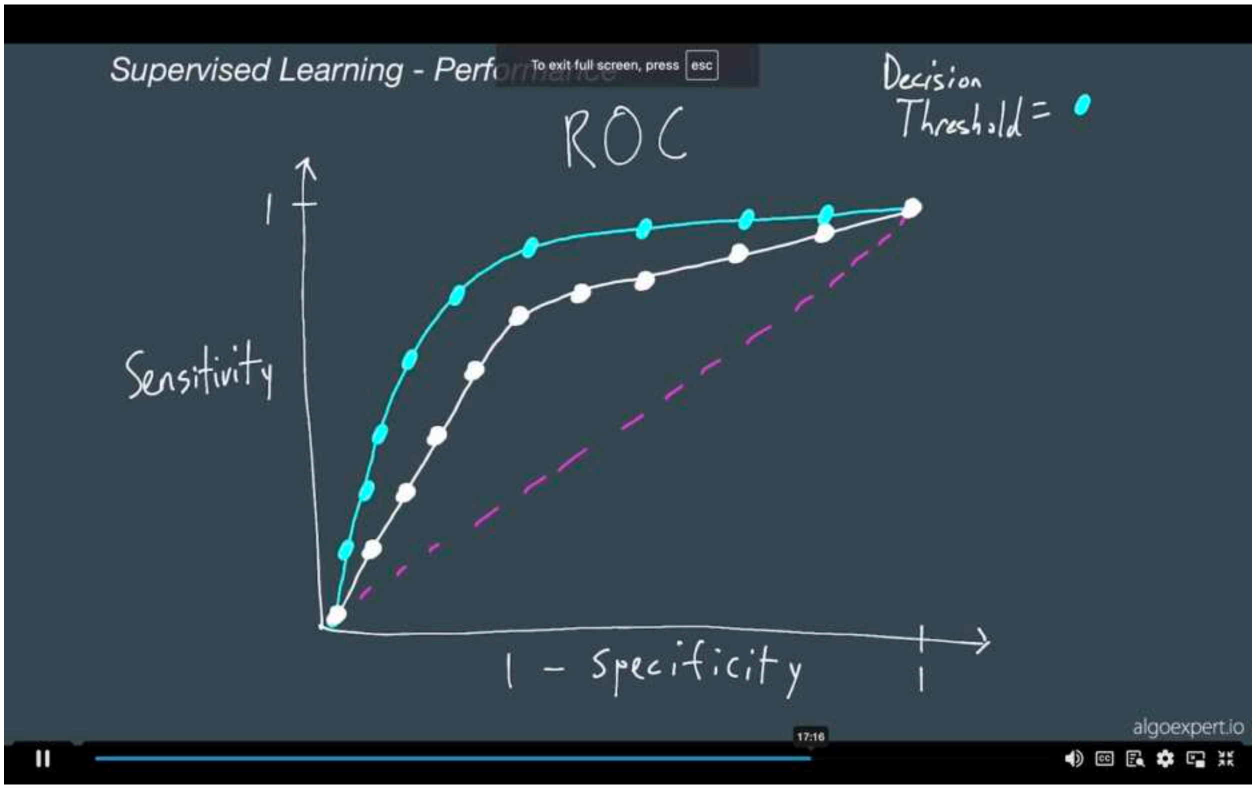
So, here our point of 100% decision threshold lies at zero. Now, every other threshold lends us somewhere between those two extremes. For reference this line is where sensitivity is equal to one minus specificity. This means, for every spam message that we correctly classify, we also incorrectly classify a legitimate message. The goal for any model should be to always lie above or be better than that line (purple line).



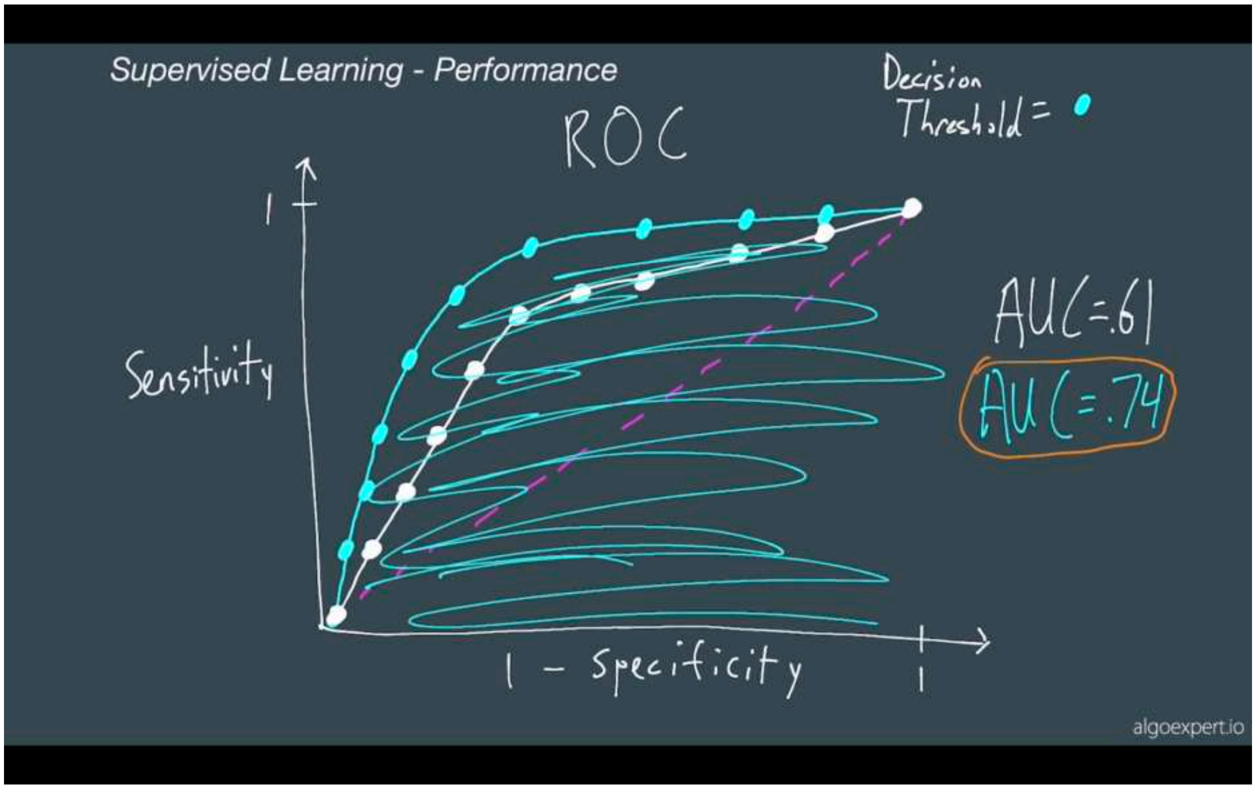
So, let's clean this up and remove those numbers. And now we can plot a particular decision threshold. To obtain a good balance between specificity and sensitivity, the idea is to pick a threshold which maximizes the distance away from this line.



Now let's say we have another model that we'd like to compare to ours. This would be the other models ROC curve.

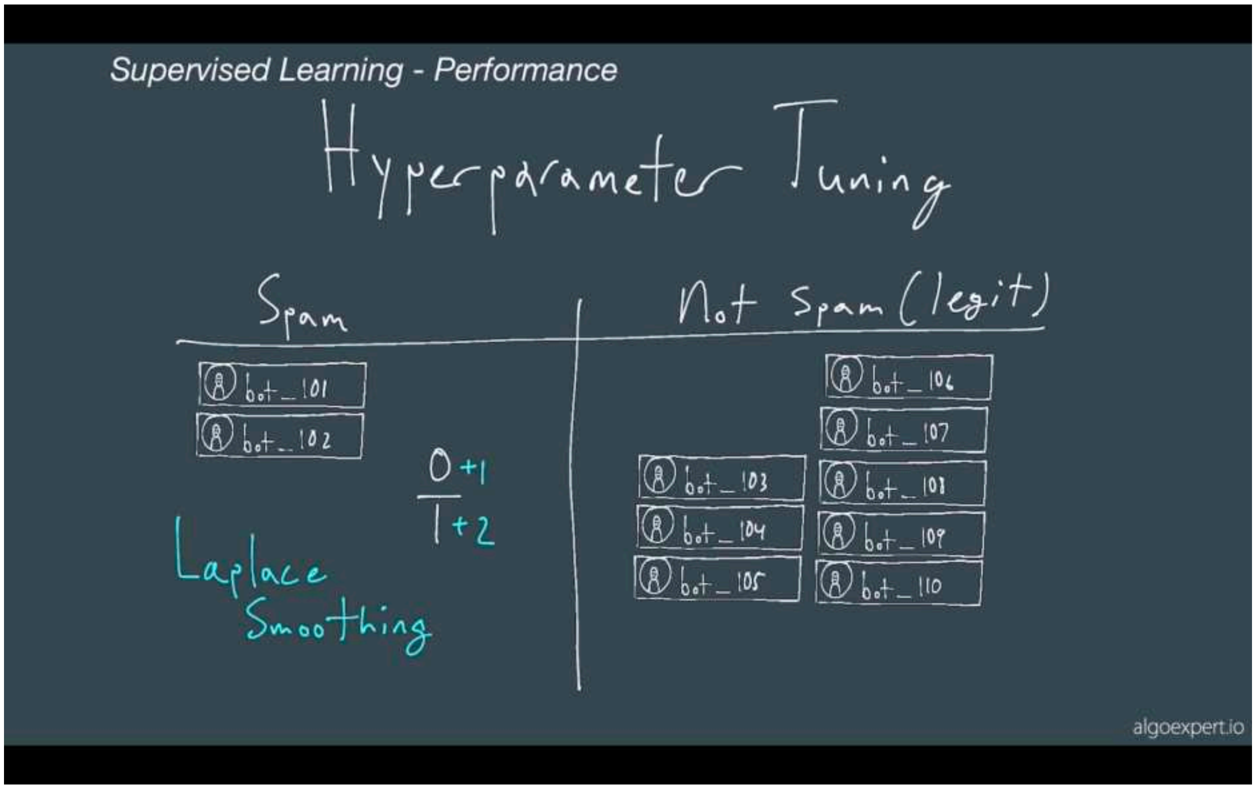


And instead of just eyeballing these two and saying, well this one might be tallest, so I guess that's better. A more exact approach is to take the area under the curve of the first model. So, here the area under the curve is 0.61. And then we take the area under the curve of the second model, here this AUC or area under the curve is 0.74.



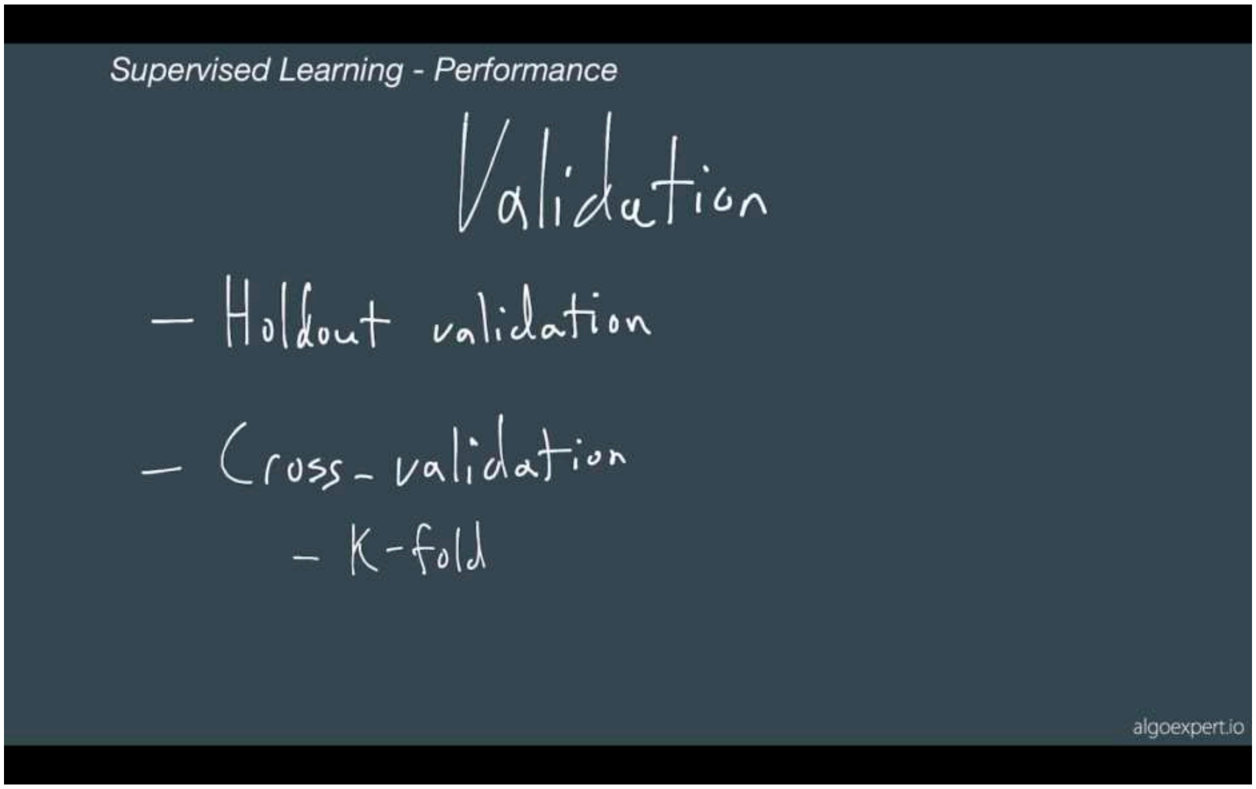
Then we compare the two values. And whichever value here is higher is the model that we could confidently say is a better predictor. This number here is technically the probability that a randomly chosen spam example we'll get a higher probability of being spam according to our model. Then a randomly chosen legitimate message. It's a good way to compare two models against all thresholds.

Finally, using our validation set, we can tune hyperparameters. **Hyperparameters** are parameters that go along with the model that you don't necessarily train. For instance, our naive base model, only had one hyperparameter.



And that was the amount or the degree to which we apply **Laplace smoothing**. So, these values here can technically be any other value and those would be hyperparameters to tune. Later, we'll go over models that have far more hyperparameters.

Finally, I wanna talk about particular ways in which we can validate. The one that we've talked about so far is called **holdout validation**, where we effectively assign a subset of the examples to be our validation set. Another way is to perform something called **K-fold cross validation**.

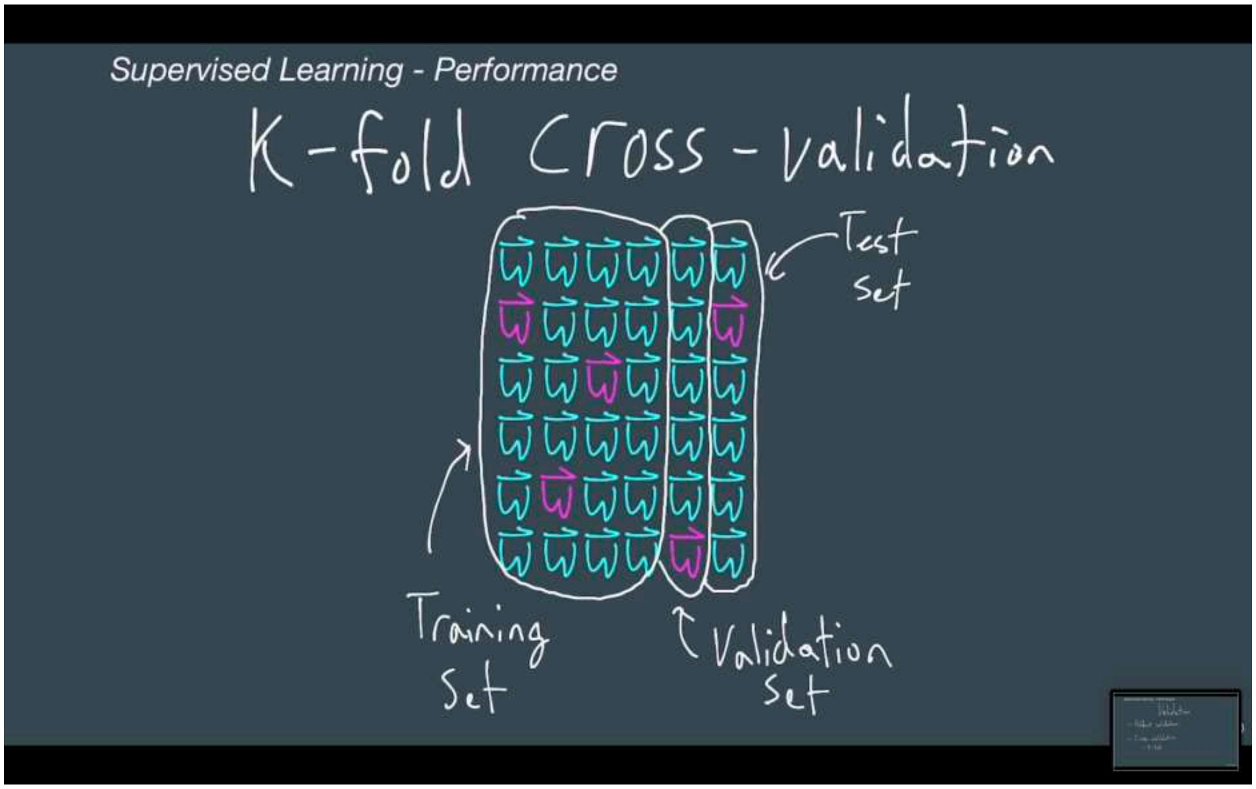


Supervised Learning - Performance

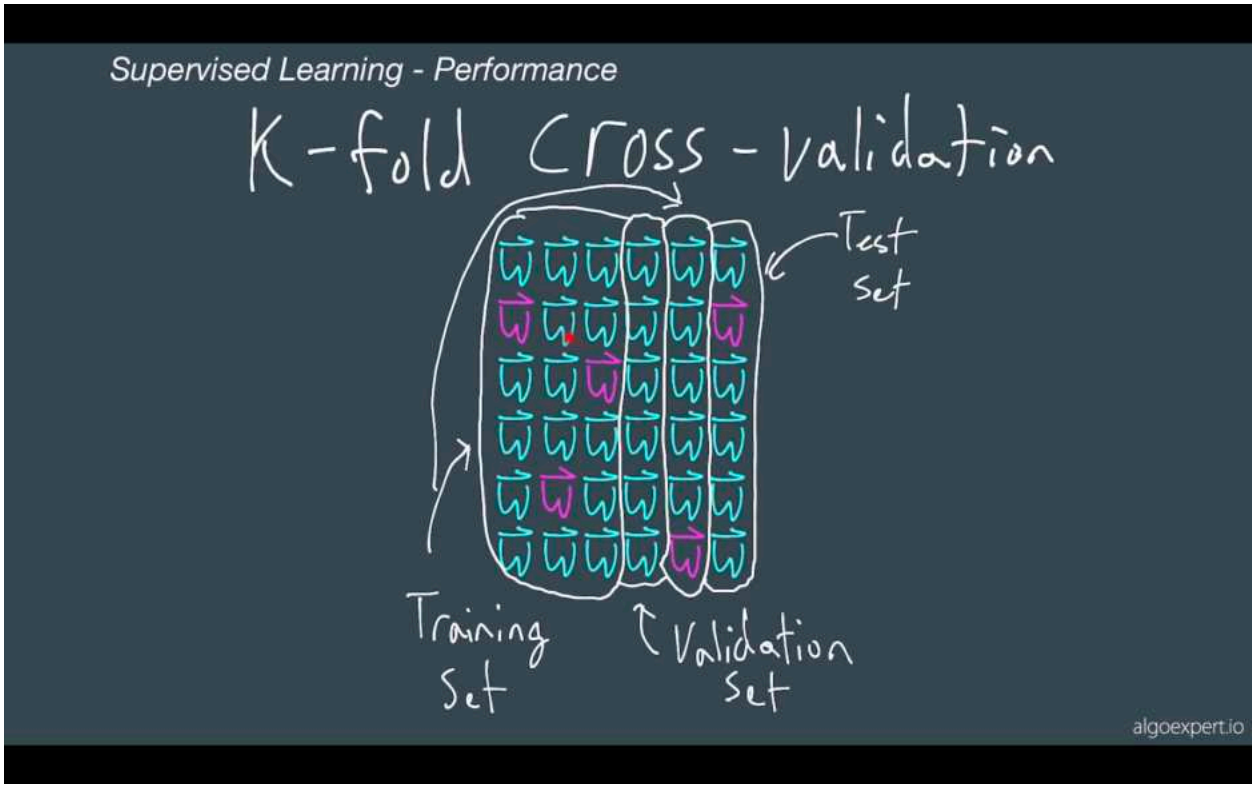
Validation

- Holdout validation
- Cross-validation
    - K-fold

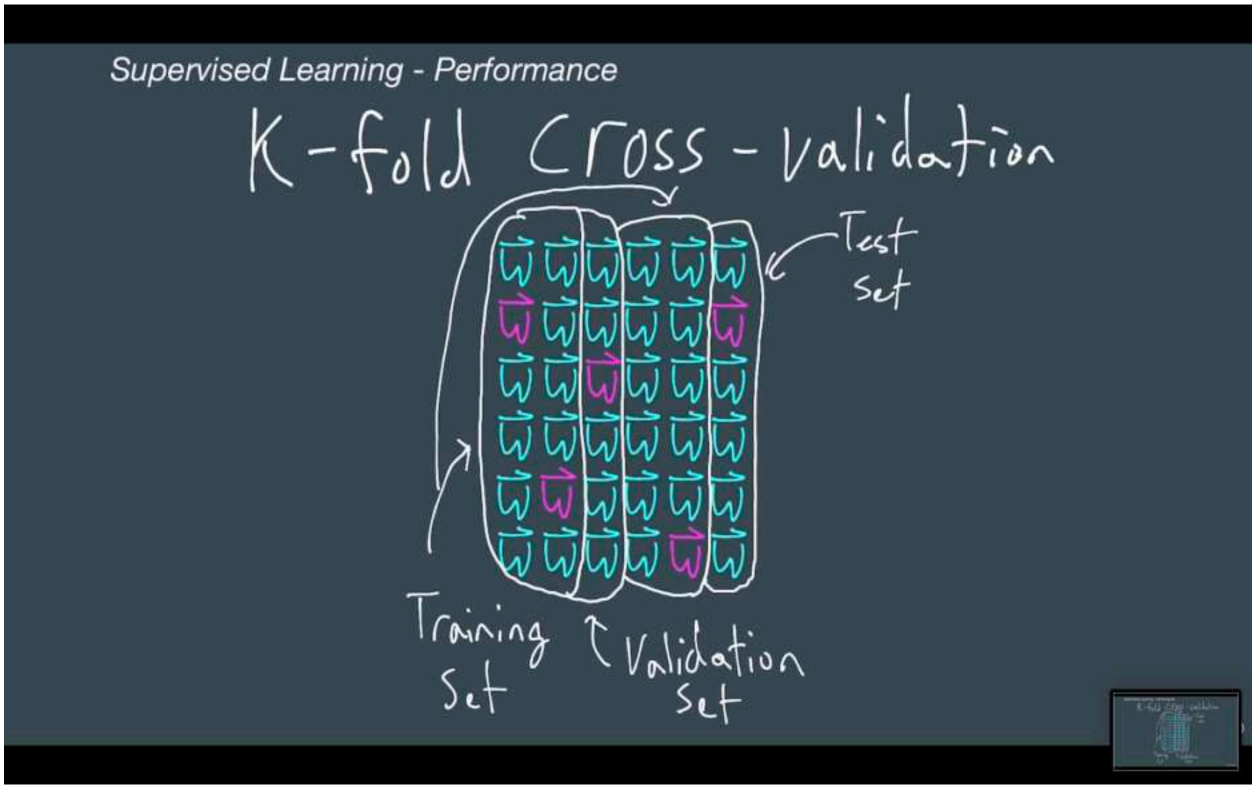
All that means is instead of just taking the random subset that we have here as our validation set, we have our tests that still set aside, but we train K different models and use a different validation set each time.



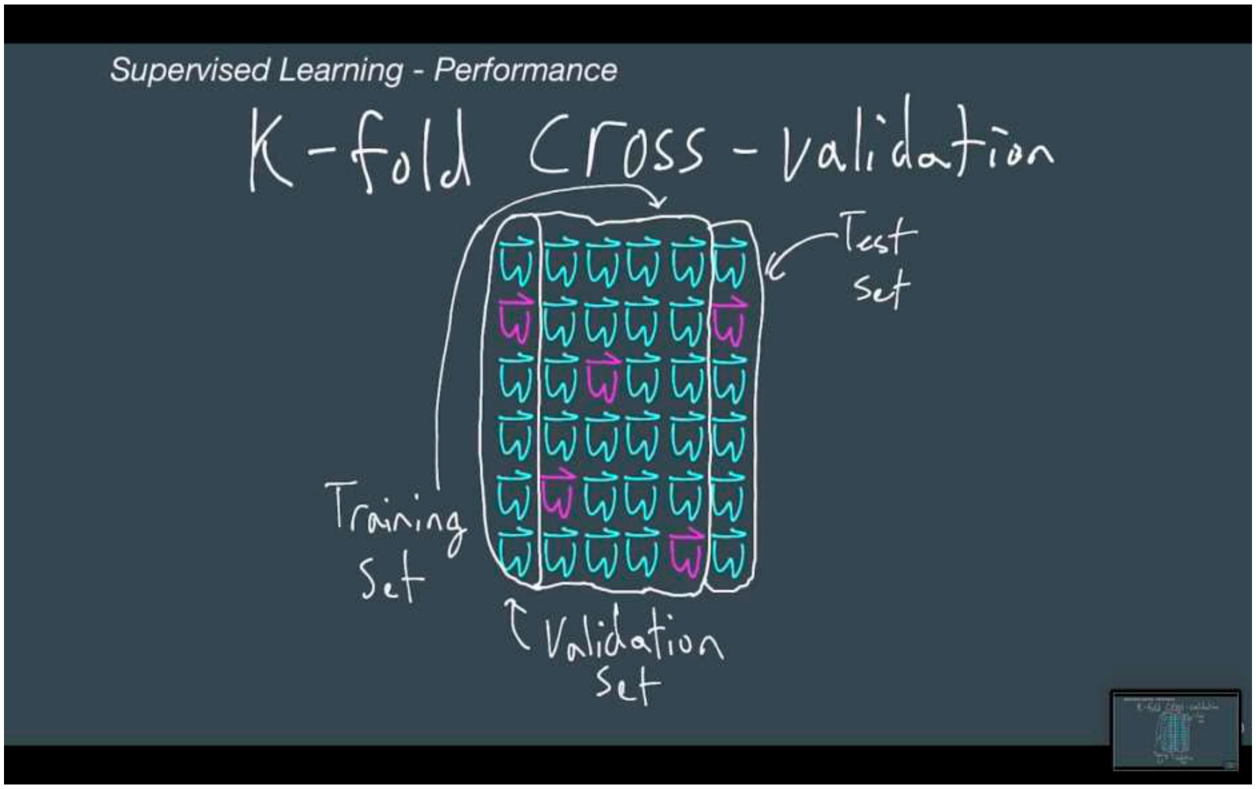
Supervised Learning - Performance



Supervised Learning - Performance



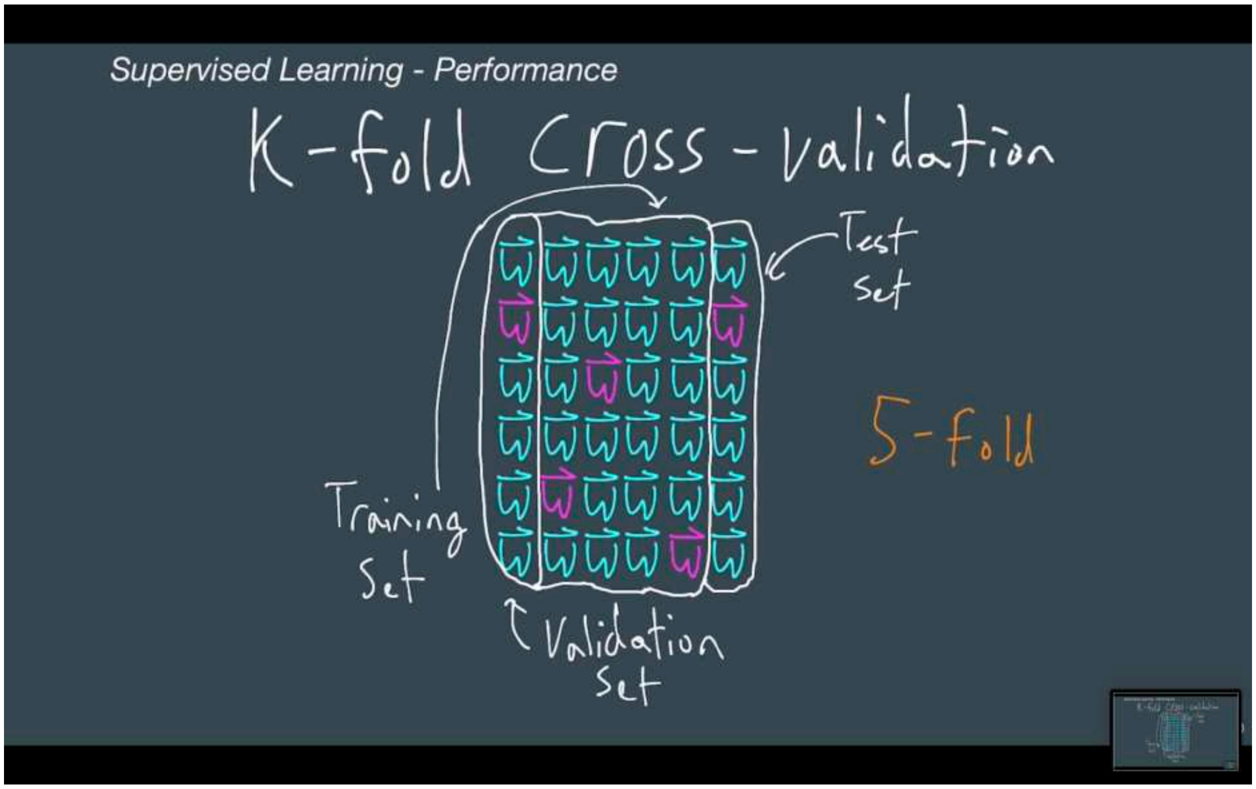
Supervised Learning - Performance



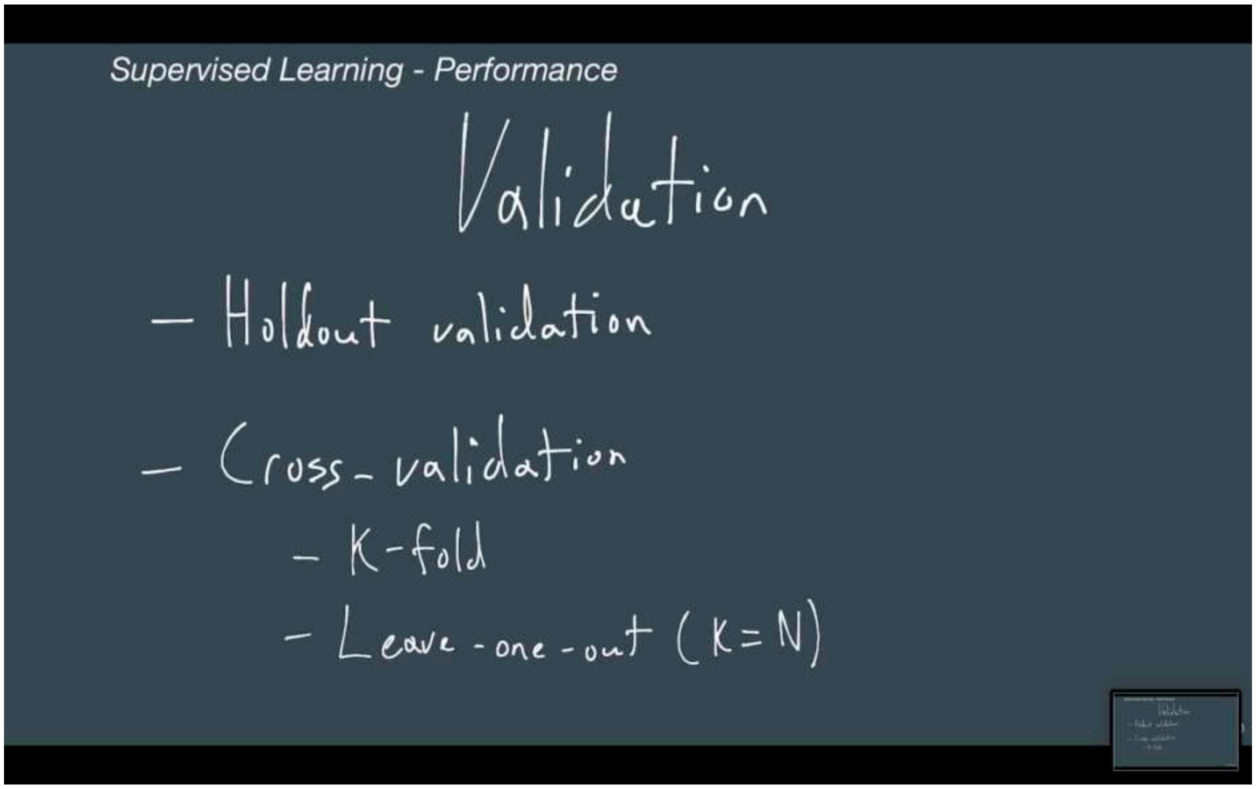
Supervised Learning - Performance

So for instance, we would train a model with this training set, and use this validation set to gauge its performance. Then we would train another model on these examples and these examples, while leaving these examples for the validation set. And we would repeat this process for every other subset or fold in the data.

So, here we had a five different folds that we used for K-fold cross validation. Every performance metric generated by a validation set, will be averaged together over the five different validation folds.



One problem with this is it will take five times longer than our original holdout validation. However, it reduces the chance that we randomly chose a validation set that produces better performance metrics by **sheer luck**. Another strategy we can use is called **leave-one-out validation**. All that means is that the K and K-fold validation is equal to N the number of examples that we have. This is generally reserved for cases where we have a very small amount of data.

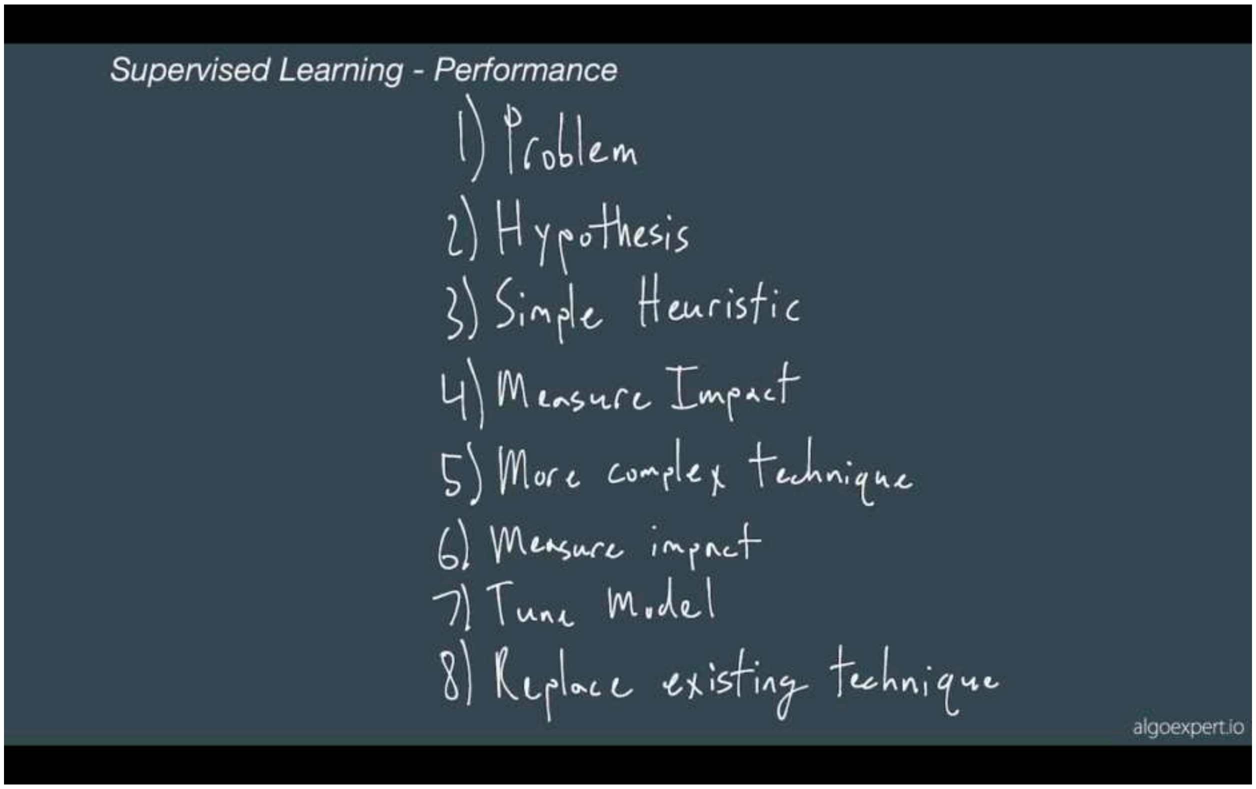

Supervised Learning - Performance

Validation

- Holdout validation
- Cross-validation
  - K-fold
  - Leave-one-out (K=N)

For the remainder of the crash course, when I've referenced cross validation, I'm generally referring to holdout validation because we will generally deal with a large enough amount of data.

So far, in the supervised learning series, we've started with a **problem**. In our case, we were needlessly interrupted by spam messages. Two we created a hypothesis. The **hypothesis** was that if we created some sort of filter rules, then it would reduce the degree of our problem.



Supervised Learning - Performance

1) Problem
2) Hypothesis
3) Simple Heuristic
4) Measure Impact
5) More complex technique
6) Measure impact
7) Tune Model
8) Replace existing technique

algoexpert.io

Three, we did create a **simple heuristic** which was our filter rules. Four, we **measured a positive impact** that the heuristic had on our initial problem. This means we validated our hypothesis. Five, we decided to invest in a **more complex technique**. In our case, it was a naive based model. We then measure the impact difference between the naive based model or the more complex technique versus the simple heuristic filter rules. Upon measuring these impact differences, we found that there wasn't shortcomings. So, we had to **tune our model.** In our case, this was the decision threshold. And finally, after tuning our model was good enough to **replace the existing technique** of the simple heuristic.

This is a very common process to go through in practical machine learning. If I was interviewing someone and they said, well we had a spam problem. So, we use naive based and it fixed it. I would be somewhat skeptical, because I've personally never seen something go that smoothly. Often interviewers will want to hear the details of this process in the iterations that you had to go through.

Now, some general tips about this process. If you're stuck on step four, I would revisit your hypothesis. And potentially I would even revisit the problem to check if I was missing anything. If you get to step seven and you can't beat the heuristic, you may need to either one invest in a more complex technique, or two revisit what features you're looking at in terms of your model.

Finally, when do you get to step eight, be sure to tie it back to some meaningful business objective. I think about our spam use case. Not only does it help me from being needlessly interrupted, as well, it helps Instagram because it ensures that I don't get drawn away from their ad revenue machine.

So, even though we're optimizing specificity and comparing F1 scores, my manager or my manager's manager only really cares about how much **additional ad revenue** Instagram will get, by having users less distracted by spam messages. We'll talk more about how to measure business impacts later.

Finally, keep in mind that this process happens at different rates for different companies. The companies that are faster to iterate on this process are typically ones where experiments are cheap to run, and where failed experiments don't ruin lives. This process will go far more slowly for companies where experiments are expensive to run in either terms of money or time, and where failed experiments can directly affect people's health. But the remainder of the crash course, we're gonna deviate from this spam detection problem to cover the intricacies and different use cases of various machine learning models. Well, that's it for this video, thanks so much for joining. Join us next video as we continue our machine learning journey.