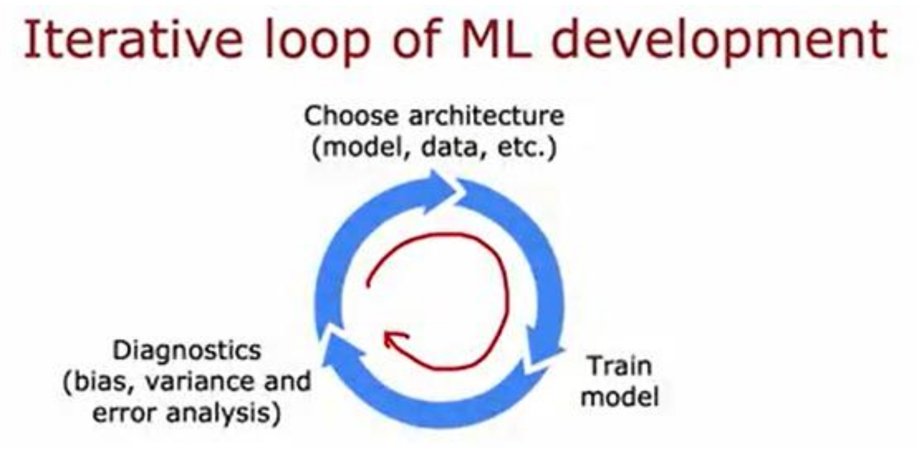
Iterative loop of ML development

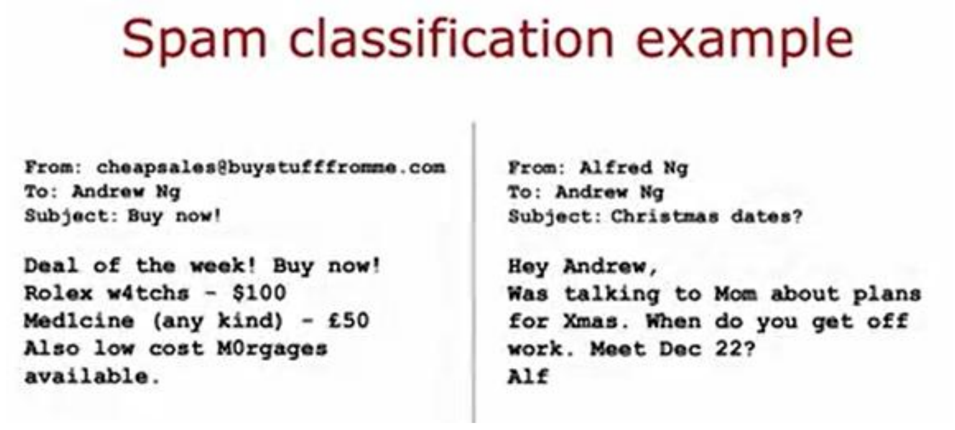
In the next few videos, I'd like to share with you what is like to go through the process of developing a machine learning system so that when you are doing so yourself, hopefully, you'd be in a position to make great decisions at many stages of the machine learning development process. Let's take a look first at the iterative loop of machine learning development. This is what developing a machine learning model will often feel like.

First, you decide on what is the overall architecture of your system. That means choosing your machine learning model as well as deciding what data to use, maybe picking the hyperparameters, and so on. Then, given those decisions, you would implement and train a model. As I've mentioned before, when you train a model for the first time, it will almost never work as well as you want it to.


Iterative loop of ML development
Choose architecture (model, data, etc.)
Train model
Diagnostics (bias, variance and error analysis)

The next step that I recommend then is to implement or to look at a few diagnostics, such as looking at the bias and variance of your algorithm as well as something we'll see in the next video called error analysis. Based on the insights from the diagnostics, you can then make decisions like do want to make your neural network bigger or change the Lambda regularization parameter, or maybe add more data or add more features or subtract features.

Then you go around this loop again with your new choice of architecture, and it will often take multiple iterations through this loop until you get to the performance that you want. Let's look at an example of building an email spam classifier. I think many of us passionately hate email spam and this is a problem that I worked on years ago and also was involved in starting an anti-spam conference once years ago. The example on the left is what a highly spammy email might look like. Deal of the week, by now, Rolex watches.



Spam classification example

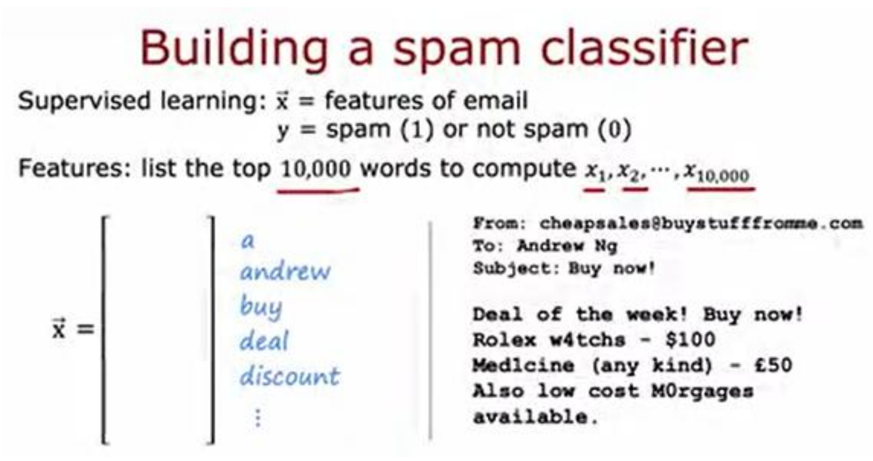
```
From: cheapsales@buystufffromme.com
To: Andrew Ng
Subject: Buy now!

Deal of the week! Buy now!
Rolex w4tchs - $100
Med1cine (any kind) - £50
Also low cost M0rgages
available.
```

```
From: Alfred Ng
To: Andrew Ng
Subject: Christmas dates?

Hey Andrew,
Was talking to Mom about plans
for Xmas. When do you get off
work. Meet Dec 22?
Alf
```

Spammers will sometimes deliberately misspell words like these, watches, medicine, and mortgages in order to try to trip up a spam recognizer. In contrast, this email on the right is an actual email I once got from my younger brother Alfred about getting together for Christmas. How do you build a classifier to recognize spam versus non-spam emails?

One way to do so would be to train a supervised learning algorithm where the input features x will be the features of an email and the output label y will be one or zero depending on whether it's spam or non-spam. This application is an example of text classification because you're taking a text document that is an email and trying to classify it as either spam or non-spam.

One way to construct the features of the email would be to say, take the top 10,000 words in the English language or in some other dictionary and use them to define features x_1, x_2 through x_10,000. For example, given this email on the right, if the list of words we have is a, Andrew buy deal discount and so on.



Building a spam classifier

Supervised learning: $\vec{x}$ = features of email
y = spam (1) or not spam (0)

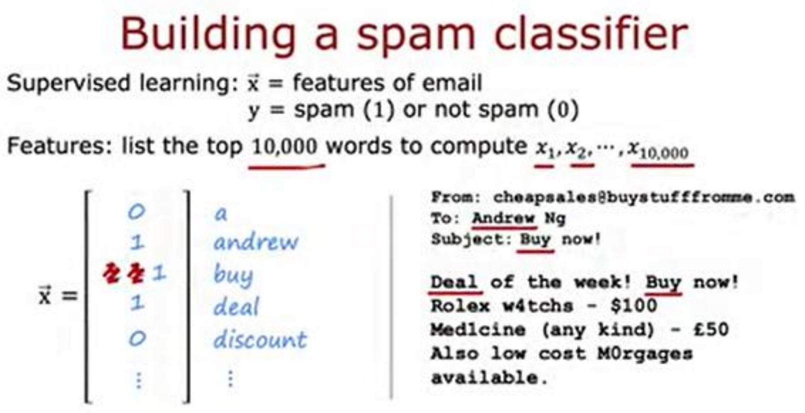
Features: list the top $\underline{10{,}000}$ words to compute $\underline{x_1}, \underline{x_2}, \cdots, \underline{x_{10,000}}$

$$\vec{x} = \begin{bmatrix} \\ \\ \\ \\ \\ \\ \end{bmatrix} \begin{array}{l} a \\ andrew \\ buy \\ deal \\ discount \\ \vdots \end{array}$$

```
From: cheapsales@buystufffromme.com
To: Andrew Ng
Subject: Buy now!

Deal of the week! Buy now!
Rolex w4tchs - $100
Med1cine (any kind) - £50
Also low cost M0rgages
available.
```

Then given the email on the right, we would set these features to be, say, 0 or 1, depending on whether or not that word appears. The word a does not appear. The word Andrew does appear. The word buy does appear, deal does, discount does not, and so on, and so you can construct 10,000 features of this email.

There are many ways to construct a feature vector. Another way would be to let these numbers not just be 1 or 0, but actually, count the number of times a given word appears in the email. If buy appears twice, maybe you want to set this to 2, but setting into just 1 or 0.



Building a spam classifier

Supervised learning: $\vec{x}$ = features of email
y = spam (1) or not spam (0)

Features: list the top $\underline{10{,}000}$ words to compute $\underline{x_1}, \underline{x_2}, \cdots, \underline{x_{10,000}}$

$$\vec{x} = \begin{bmatrix} 0 \\ 1 \\ 2\ 2\ 1 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \begin{array}{l} a \\ andrew \\ buy \\ deal \\ discount \\ \vdots \end{array}$$

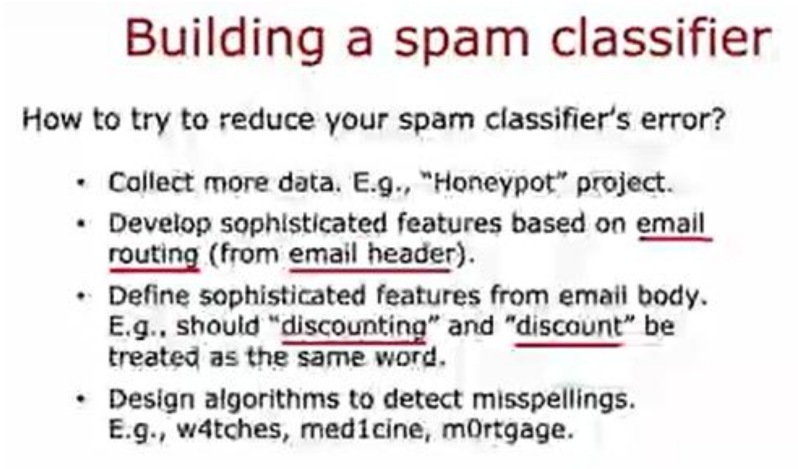
```
From: cheapsales@buystufffromme.com
To: Andrew Ng
Subject: Buy now!

Deal of the week! Buy now!
Rolex w4tchs - $100
Med1cine (any kind) - £50
Also low cost M0rgages
available.
```

It actually works decently well. Given these features, you can then train a classification algorithm such as a logistic regression model or a neural network to predict y given these features x. After you've trained your initial model, if it doesn't work as well as you wish, you will quite likely have multiple ideas for improving the learning algorithm's performance.

For example, is always tempting to collect more data. In fact, I have friends that have worked on very large-scale honeypot projects. These are projects that create a large number of fake email addresses and tries to deliberately to get these fake email addresses into the hands of spammers so that when they send spam email to these fake emails well we know these are spam email messages and so this is a way to get a lot of spam data. Or you might decide to work on developing more sophisticated features based on the email routing.

Email routing refers to the sequence of compute service. Sometimes around the world that the email has gone through all this way to reach you and emails actually have what's called email header information. That is information that keeps track of how the email has traveled across different servers, across different networks to find its way to you. Sometimes the path that an email has traveled can help tell you if it was sent by a spammer or not.

Or you might work on coming up with more sophisticated features from the email body that is the text of the email. In the features I talked about last time, discounting and discount may be treated as different words, and maybe they should be treated as the same words. Or you might decide to come up with algorithms to detect misspellings or deliberate misspellings like watches, medicine, and mortgage and this too could help you decide if an email is spammy.



Building a spam classifier

How to try to reduce your spam classifier's error?

- Collect more data. E.g., "Honeypot" project.
- Develop sophisticated features based on email routing (from email header).
- Define sophisticated features from email body. E.g., should "discounting" and "discount" be treated as the same word.
- Design algorithms to detect misspellings. E.g., w4tches, med1cine, m0rtgage.

Given all of these and possibly even more ideas, how can you decide which of these ideas are more promising to work on? Because choosing the more promising path forward can speed up your project easily 10 times compared to if you were to somehow choose some of the less promising directions. For example, we've already seen that if your algorithm has high bias rather than high variance, then spending months and months on a honeypot project may not be the most fruitful direction.

But if your algorithm has high variance, then collecting more data could help a lot. Doing the iterative loop of machinery and development, you may have many ideas for how to modify the model or the data, and it will be coming up with different diagnostics that could give you a lot of guidance on what choices for the model or data, or other parts of the architecture could be most promising to try. In the last several videos, we've already talked about bias and variance. In the next video, I'd like to start describing to you the error analysis process, which has a second key set of ideas for gaining insight about what architecture choices might be fruitful.