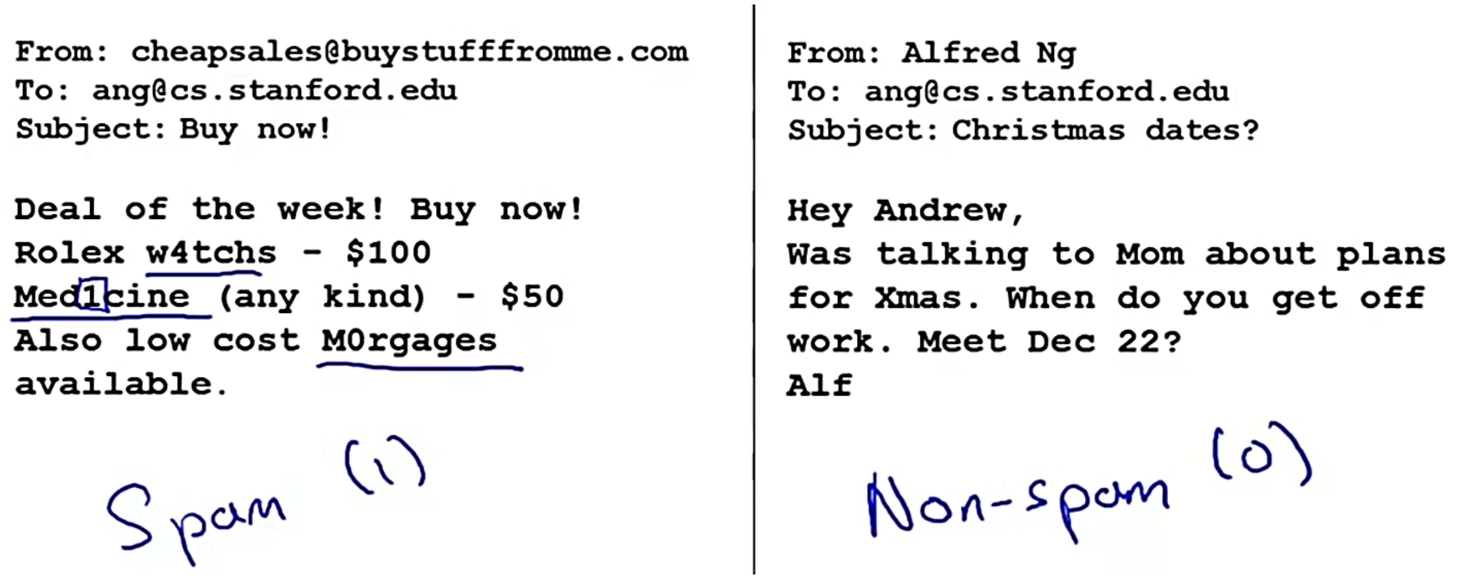
11. Machine Learning System Design:

PRIORITIZING WHAT TO WORK ON: BUILDING A SPAM CLASSIFIER:



```
From: cheapsales@buystufffromme.com
To: ang@cs.stanford.edu
Subject: Buy now!

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

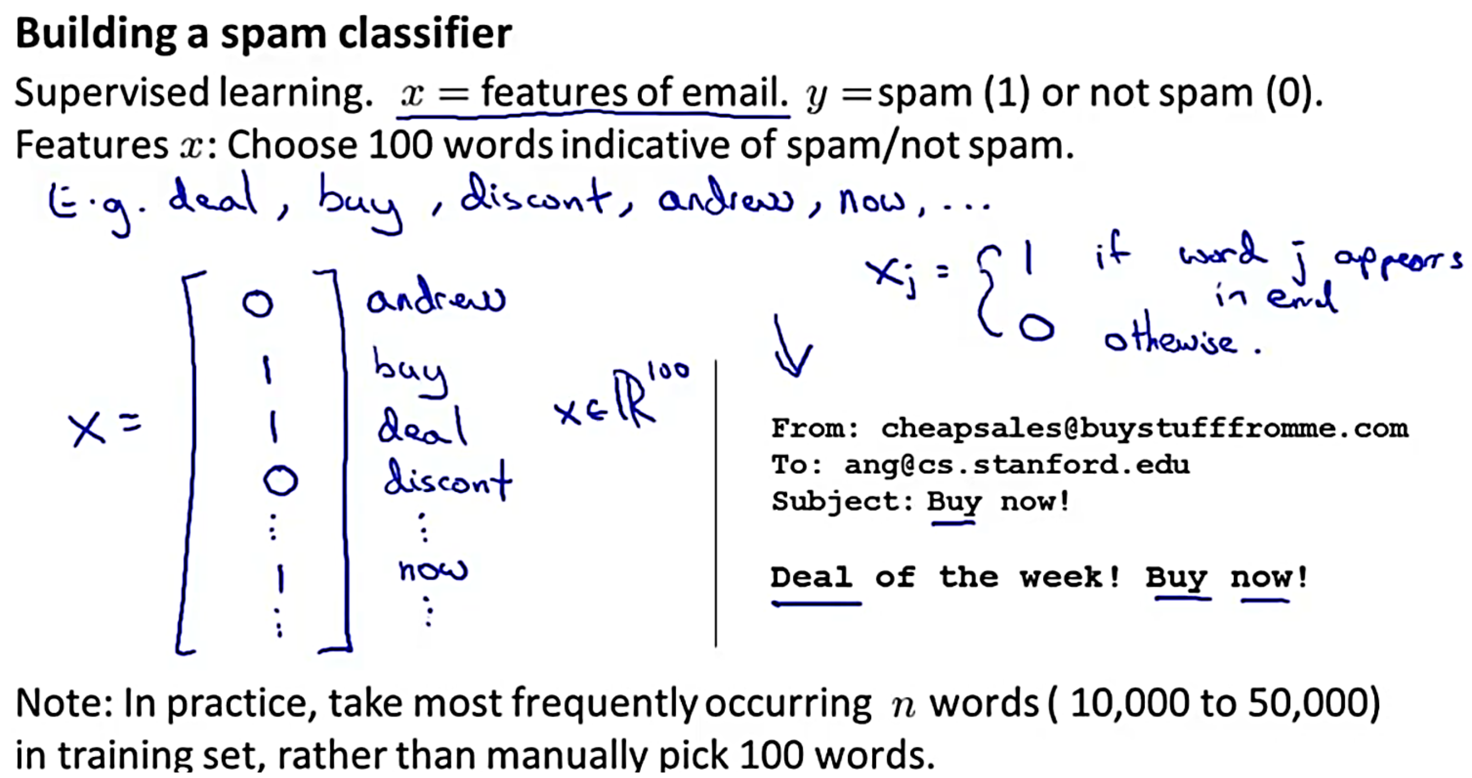
Spam (1)

```
From: Alfred Ng
To: ang@cs.stanford.edu
Subject: Christmas dates?

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

Non-spam (0)

If a word is to be found in the email, we would assign its respective entry a 1, else 0.



Building a spam classifier

Supervised learning. $x =$ features of email. $y =$ spam (1) or not spam (0).

Features x: Choose 100 words indicative of spam/not spam.

E.g. deal, buy, discont, andrew, now, ...

$$x_j = \begin{cases} 1 & \text{if word } j \text{ appears in email} \\ 0 & \text{othewise.} \end{cases}$$

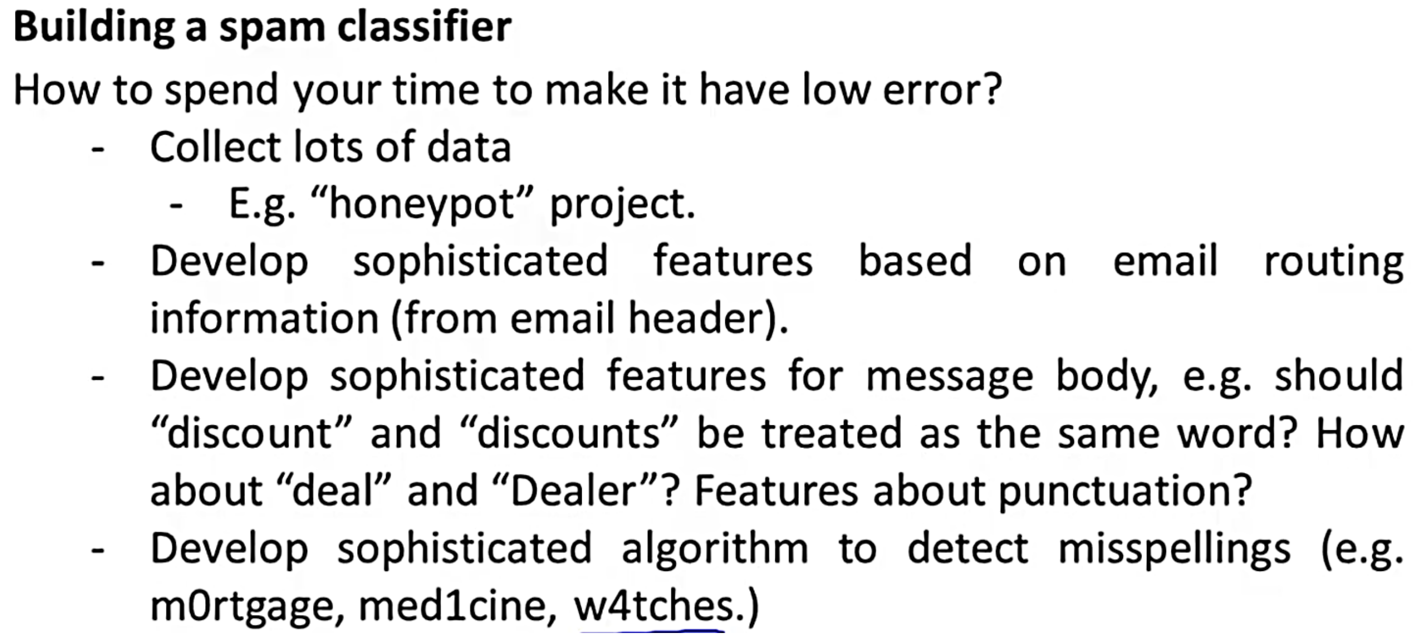
$$x = \left[\begin{array}{c|l} 0 & \text{andrew} \\ 1 & \text{buy} \\ 1 & \text{deal} \\ 0 & \text{discont} \\ \vdots & \vdots \\ 1 & \text{now} \\ \vdots & \vdots \end{array}\right] \quad x \in \mathbb{R}^{100}$$

```
From: cheapsales@buystufffromme.com
To: ang@cs.stanford.edu
Subject: Buy now!

Deal of the week! Buy now!
```

Note: In practice, take most frequently occurring n words (10,000 to 50,000) in training set, rather than manually pick 100 words.

So how could you spend your time to improve the accuracy of this classifier?



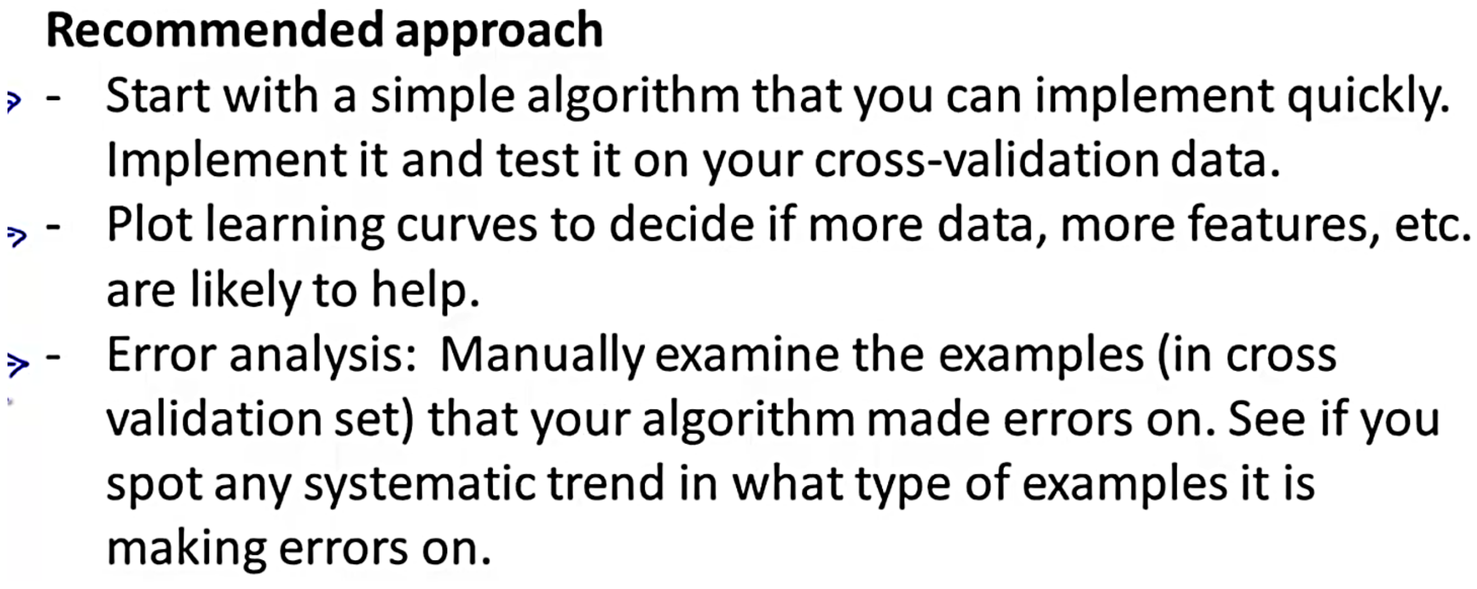
Building a spam classifier

How to spend your time to make it have low error?

- Collect lots of data
  - E.g. “honeypot” project.
- Develop sophisticated features based on email routing information (from email header).
- Develop sophisticated features for message body, e.g. should “discount” and “discounts” be treated as the same word? How about “deal” and “Dealer”? Features about punctuation?
- Develop sophisticated algorithm to detect misspellings (e.g. m0rtgage, med1cine, w4tches.)

ERROR ANALYSIS:

We can try out different quick and dirty implementations of ML algorithms and learning curves.



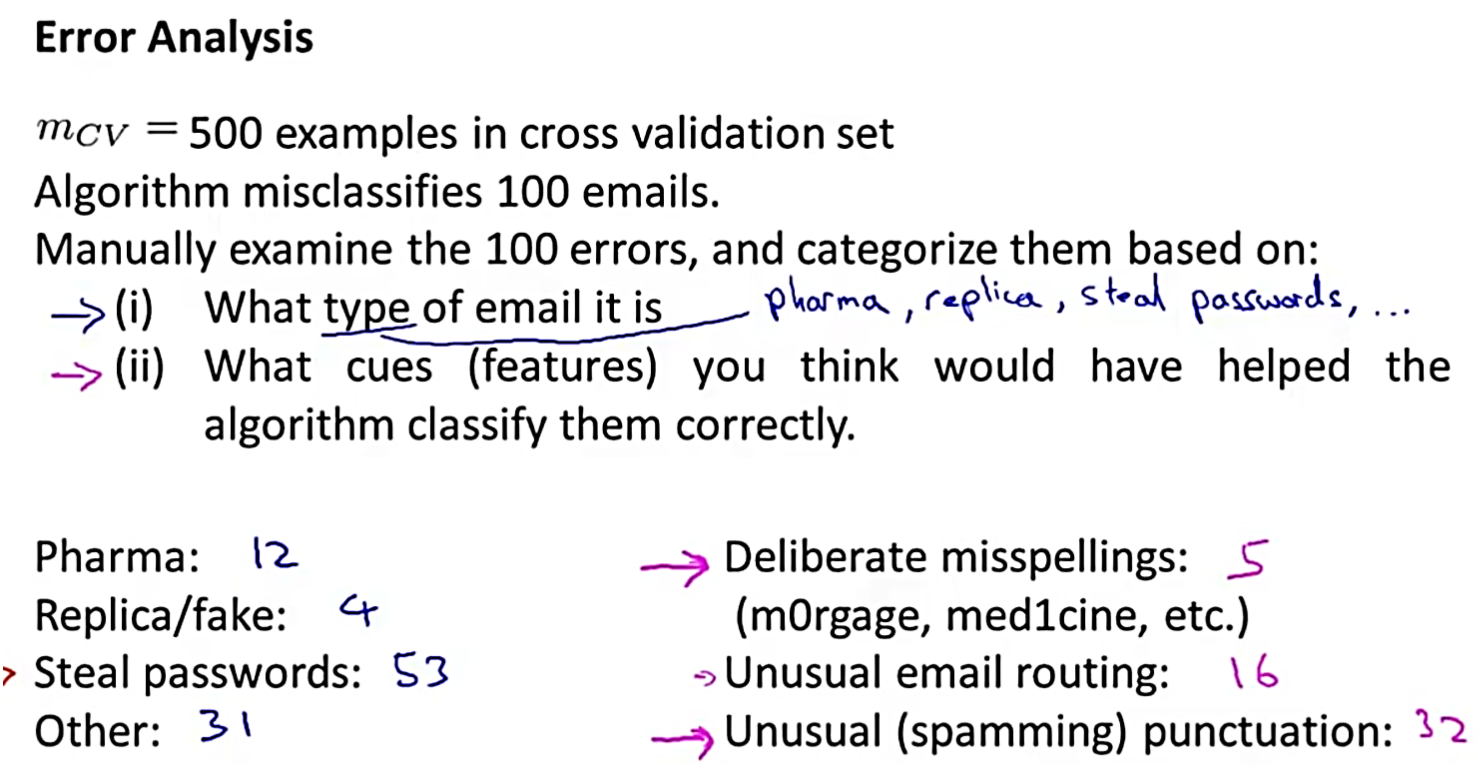
Recommended approach

- Start with a simple algorithm that you can implement quickly. Implement it and test it on your cross-validation data.
- Plot learning curves to decide if more data, more features, etc. are likely to help.
- Error analysis: Manually examine the examples (in cross validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

Lets take a high error example:

For example, assume that we have 500 emails and our algorithm misclassifies a 100 of them. We could manually analyse the 100 emails and categorize them based on what type of emails they are. We could then try to come up with new cues and features that would help us classify these 100 emails correctly.

Hence, if most of our misclassified emails are those which try to steal passwords, then we could find some features that are particular to those emails and add them to our model.



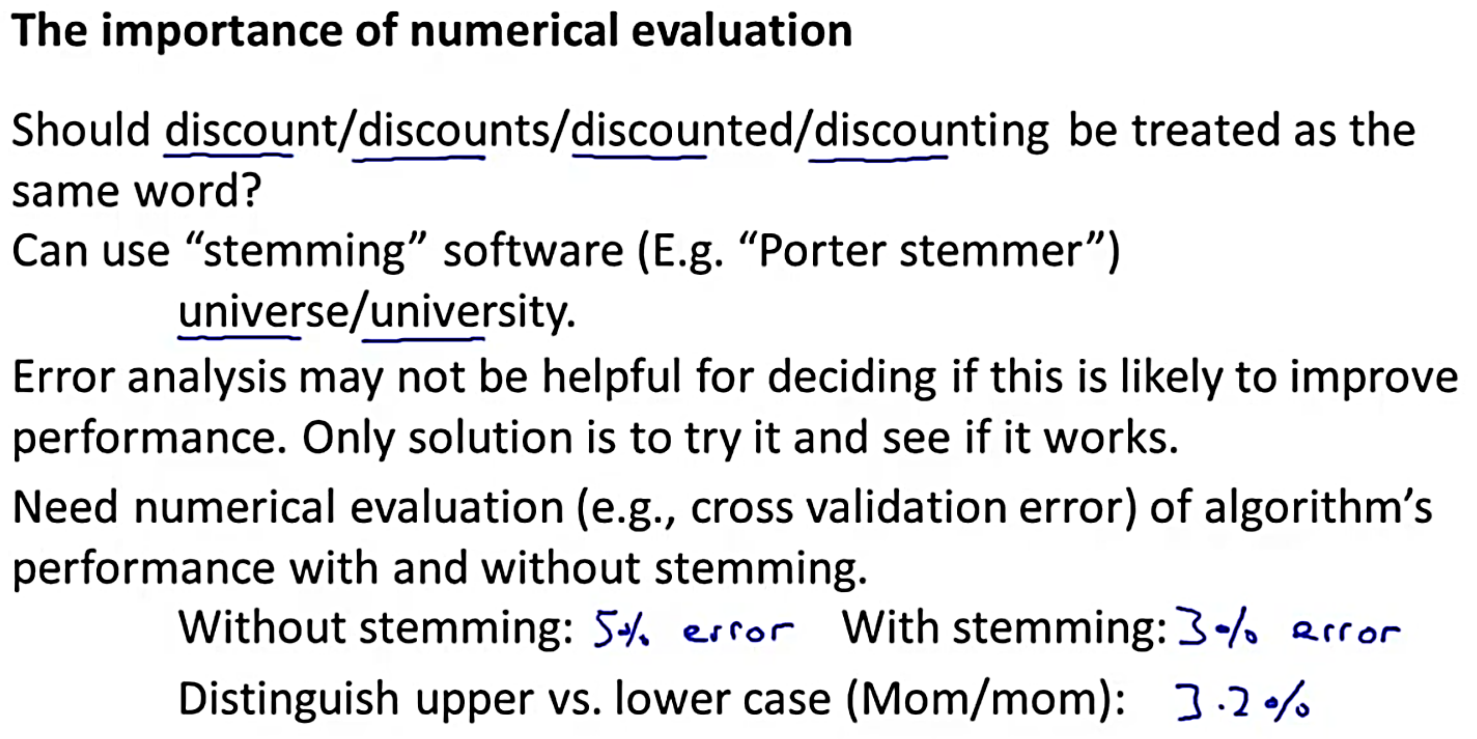
Error Analysis

$m_{CV} =$ 500 examples in cross validation set
Algorithm misclassifies 100 emails.
Manually examine the 100 errors, and categorize them based on:

→ (i) What type of email it is pharma, replica, steal passwords, ...
→ (ii) What cues (features) you think would have helped the algorithm classify them correctly.

Pharma: 12
Replica/fake: 4
> Steal passwords: 53
Other: 31

→ Deliberate misspellings: 5
(m0rgage, med1cine, etc.)
→ Unusual email routing: 16
→ Unusual (spamming) punctuation: 32



The importance of numerical evaluation

Should discount/discounts/discounted/discounting be treated as the same word?
Can use "stemming" software (E.g. "Porter stemmer")
universe/university.
Error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it and see if it works.
Need numerical evaluation (e.g., cross validation error) of algorithm's performance with and without stemming.
Without stemming: 5% error With stemming: 3% error
Distinguish upper vs. lower case (Mom/mom): 3.2%

It is very important to get error results as a single, numerical value. Otherwise it is difficult to assess your algorithm's performance.

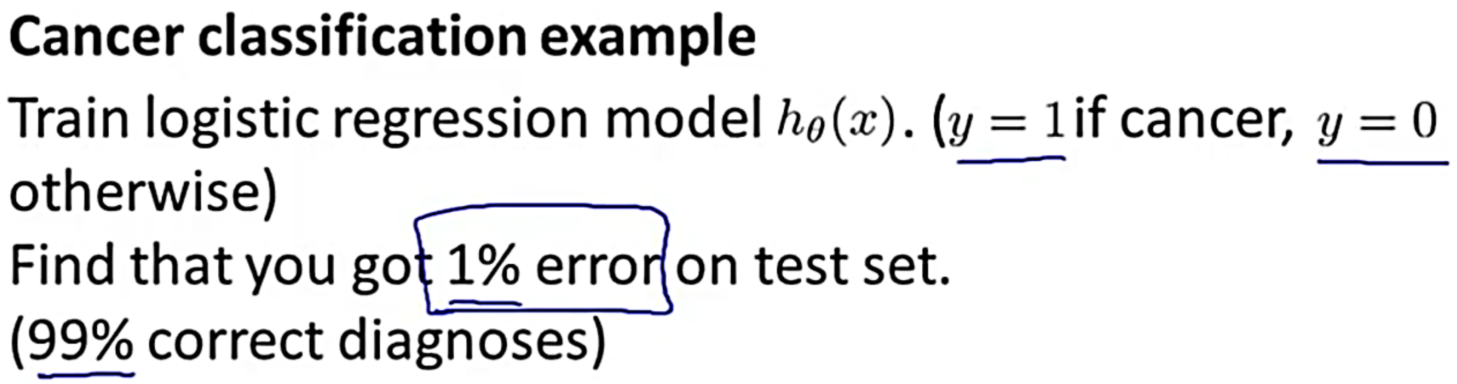
For example, if we use stemming, which is the process of treating the same word with different forms (fail/failing/failed) as one word (fail), and get a 3% error rate instead of 5%, then we should definitely add it to our model.

However, if we try to distinguish between upper case and lower-case letters and end up getting a 3.2% error rate instead of 3%, then we should avoid using this new feature.

Hence, we should try new things, get a numerical value for our error rate, and based on our result decide whether we want to keep the new feature or not.

ERROR METRICS FOR SKEWED CLASSES:

Suppose we train a model and it gives an error of 1%.



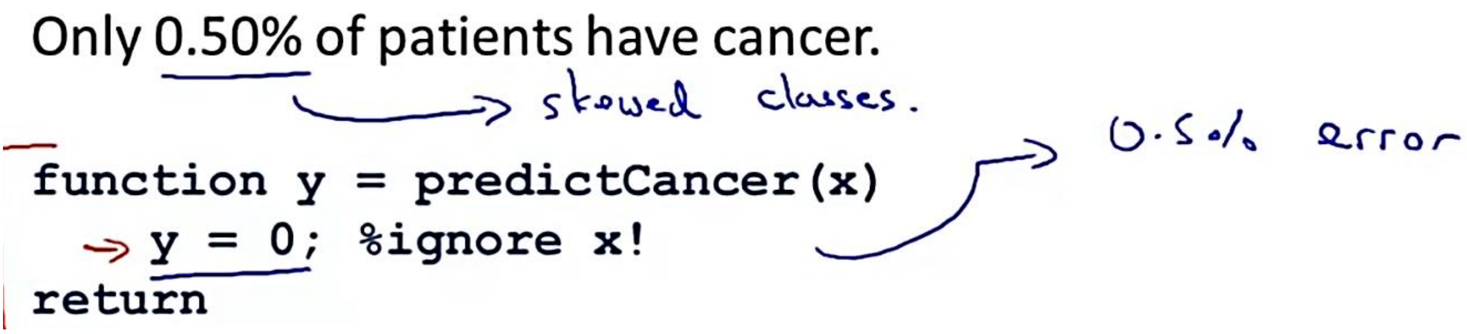
Cancer classification example

Train logistic regression model $h_\theta(x)$. ($\underline{y=1}$ if cancer, $\underline{y=0}$ otherwise)

Find that you got $\boxed{\underline{1\%}\text{ error}}$ on test set.

($\underline{99\%}$ correct diagnoses)

But in the training examples, only 0.5 % people actually have cancer… so predicting y=0 always is better choice as it gives only 0.5% error



Only 0.50% of patients have cancer.

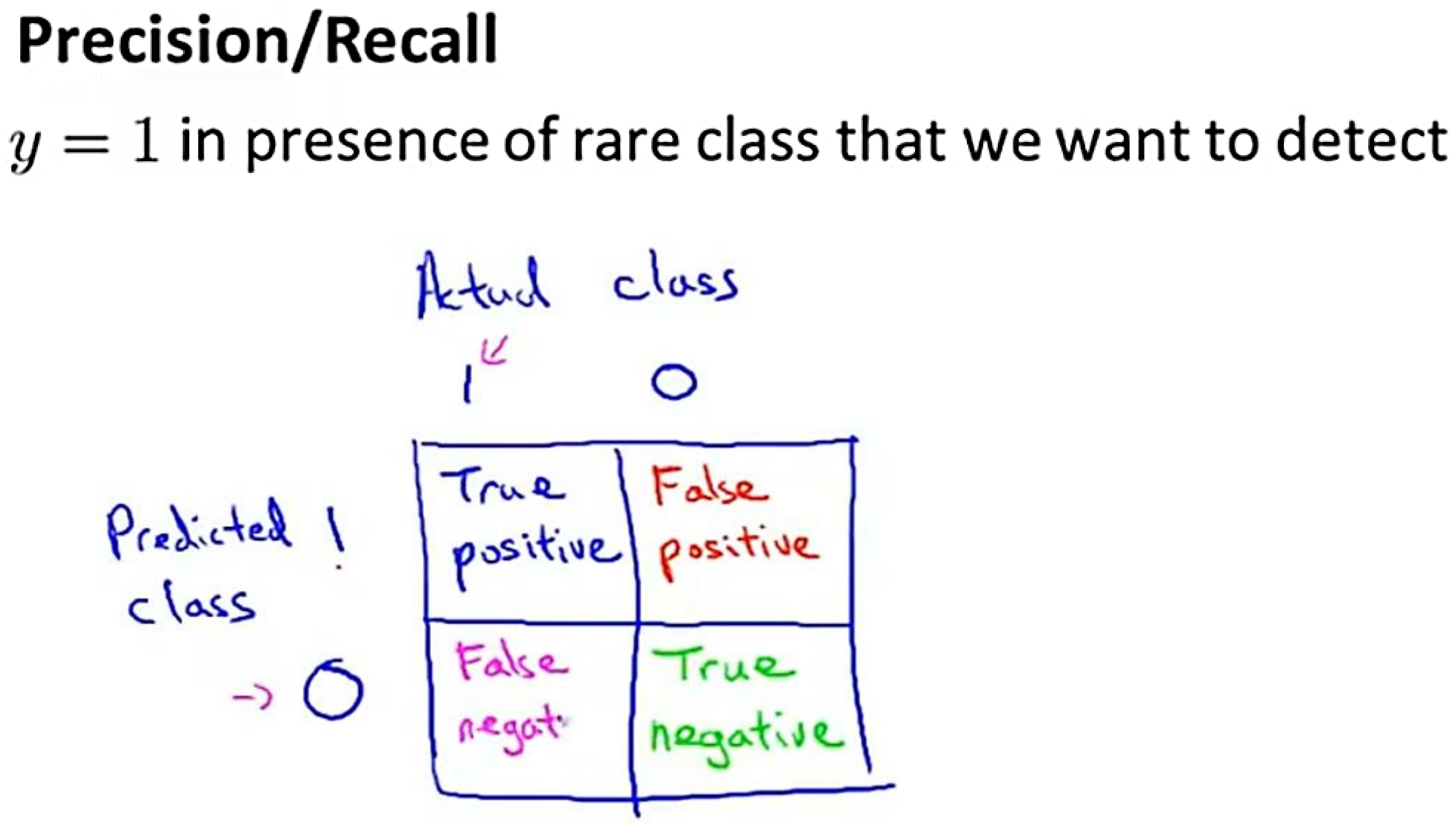
→ skewed classes.

```
function y = predictCancer(x)
  → y = 0; %ignore x!
return
```

→ 0.5% error

This is an ambiguous form of judgement.

So, we come up with a new form of error metric: **Precision and Recall**

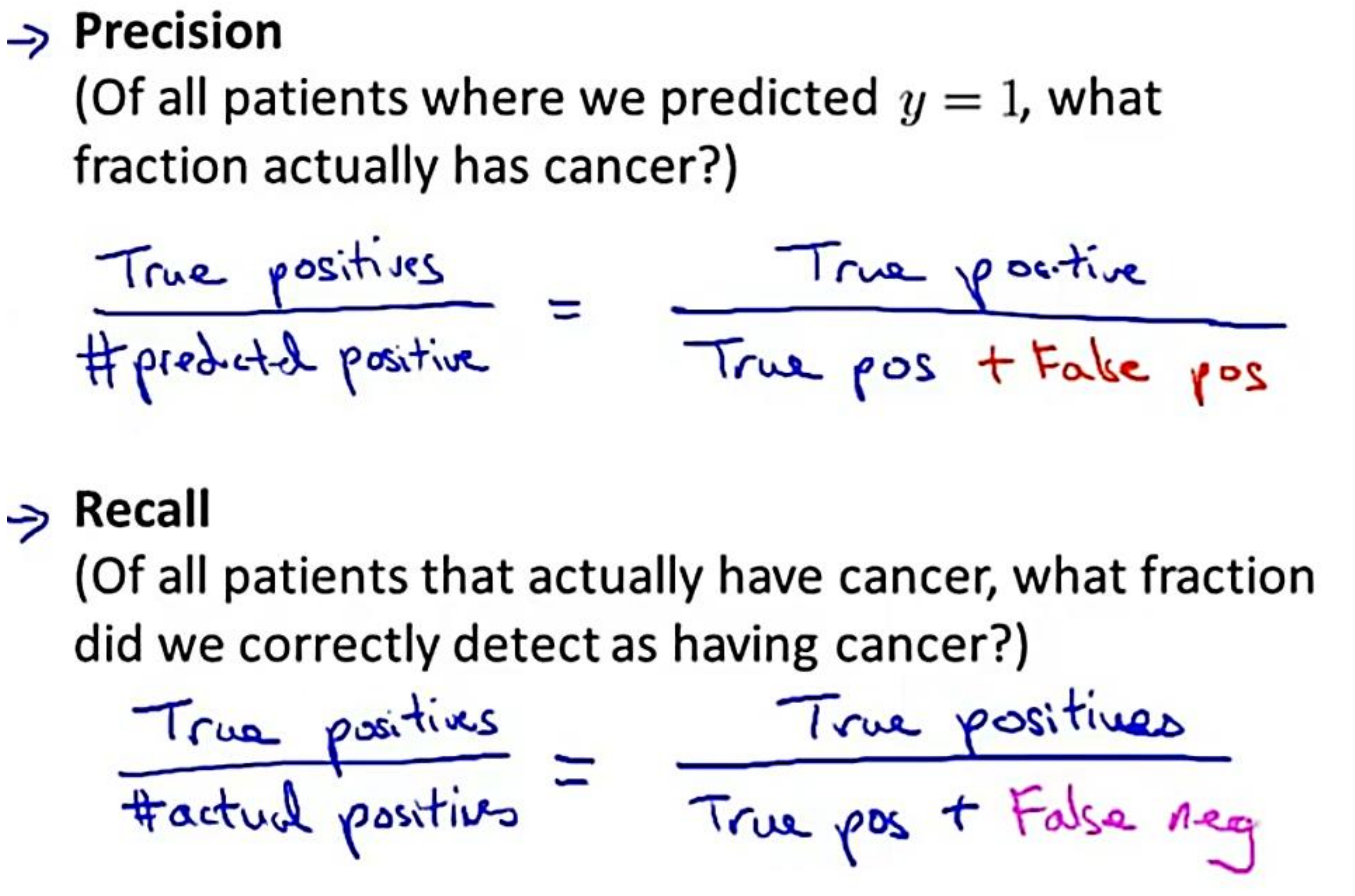


Precision/Recall

$y = 1$ in presence of rare class that we want to detect

		Actual class	
		1	0
Predicted class	1	True positive	False positive
	0	False negat	True negative

Precision and Recall are defined by setting the rare class = 1not =0…



→ **Precision**
(Of all patients where we predicted $y = 1$, what fraction actually has cancer?)

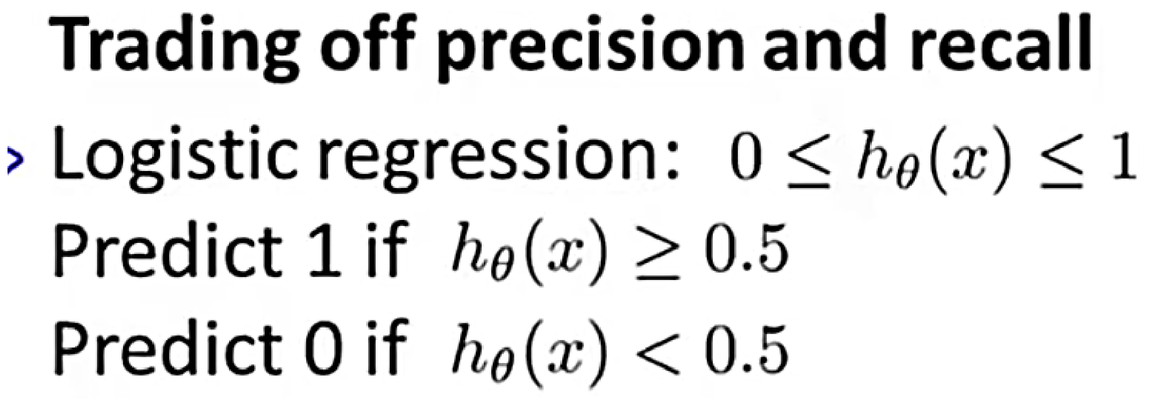
$$\frac{\text{True positives}}{\text{\#predicted positive}} = \frac{\text{True positive}}{\text{True pos} + \text{False pos}}$$

→ **Recall**
(Of all patients that actually have cancer, what fraction did we correctly detect as having cancer?)

$$\frac{\text{True positives}}{\text{\#actual positives}} = \frac{\text{True positives}}{\text{True pos} + \text{False neg}}$$

Now, if the algo predicts y=0 all the time, then this will give Recall = 0

- Classifier with high precision and high recall is a good classifier
- Even with very skewed classes, the algo can't cheat



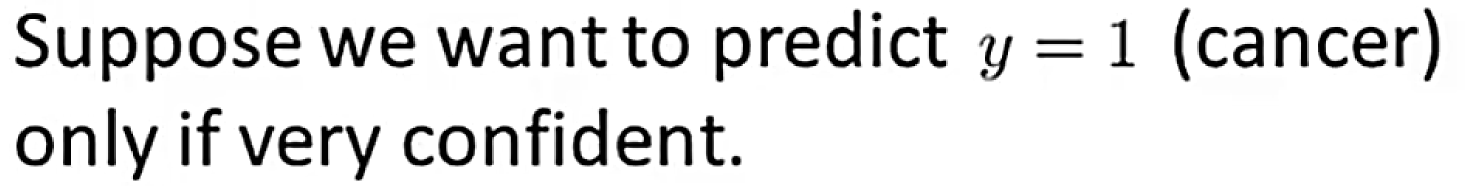
Trading off precision and recall

› Logistic regression: $0 \leq h_\theta(x) \leq 1$

Predict 1 if $h_\theta(x) \geq 0.5$

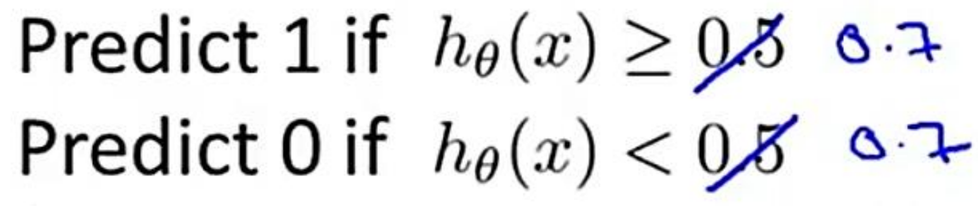
Predict 0 if $h_\theta(x) < 0.5$

But:



Suppose we want to predict $y = 1$ (cancer) only if very confident.

So, to get high precision:



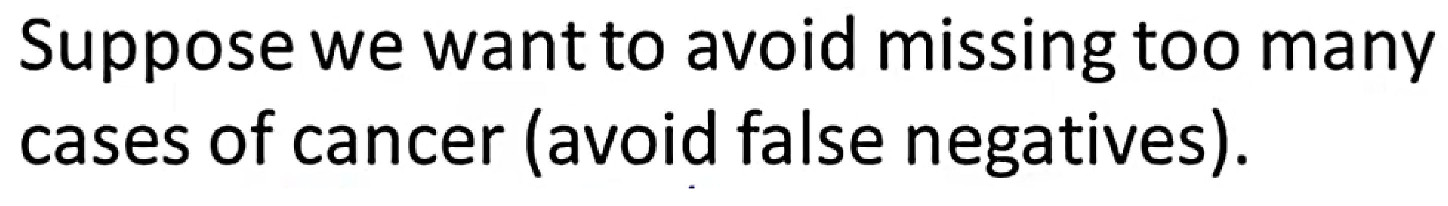
Predict 1 if $h_\theta(x) \geq \cancel{0.5}$ 0.7

Predict 0 if $h_\theta(x) < \cancel{0.5}$ 0.7

This will give **high precision** → as no of predicted positives is decreased

And **low recall**

If:



Suppose we want to avoid missing too many cases of cancer (avoid false negatives).

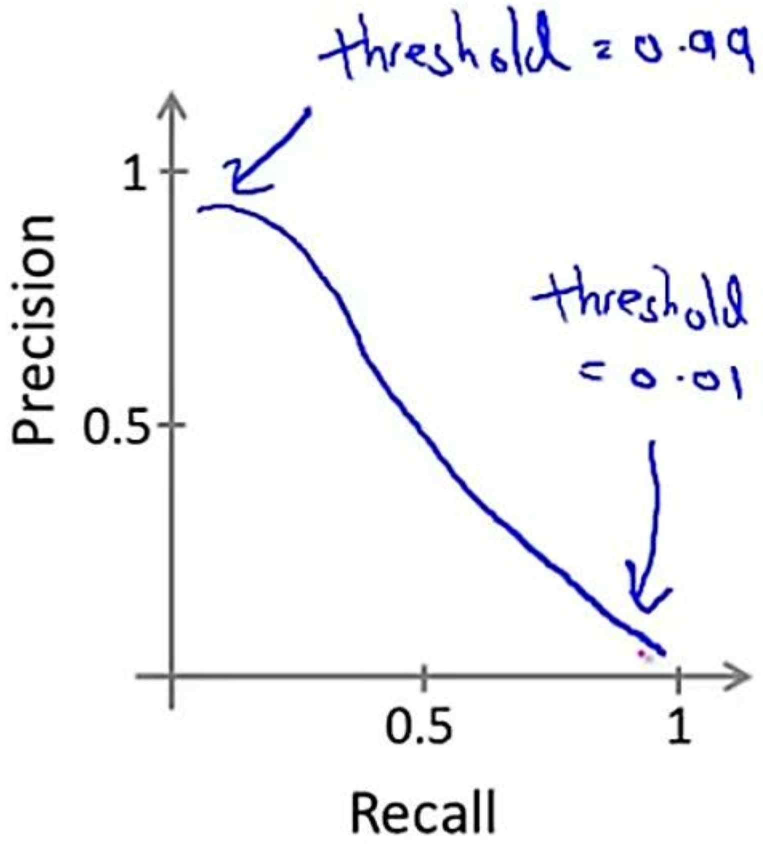
Then:

Predict 1 if $h_\theta(x) \geq \cancel{0.5}$ $\cancel{0.7}$ $\cancel{0.9}$ 0.3

Predict 0 if $h_\theta(x) < \cancel{0.5}$ $\cancel{0.7}$ $\cancel{0.9}$ 0.3

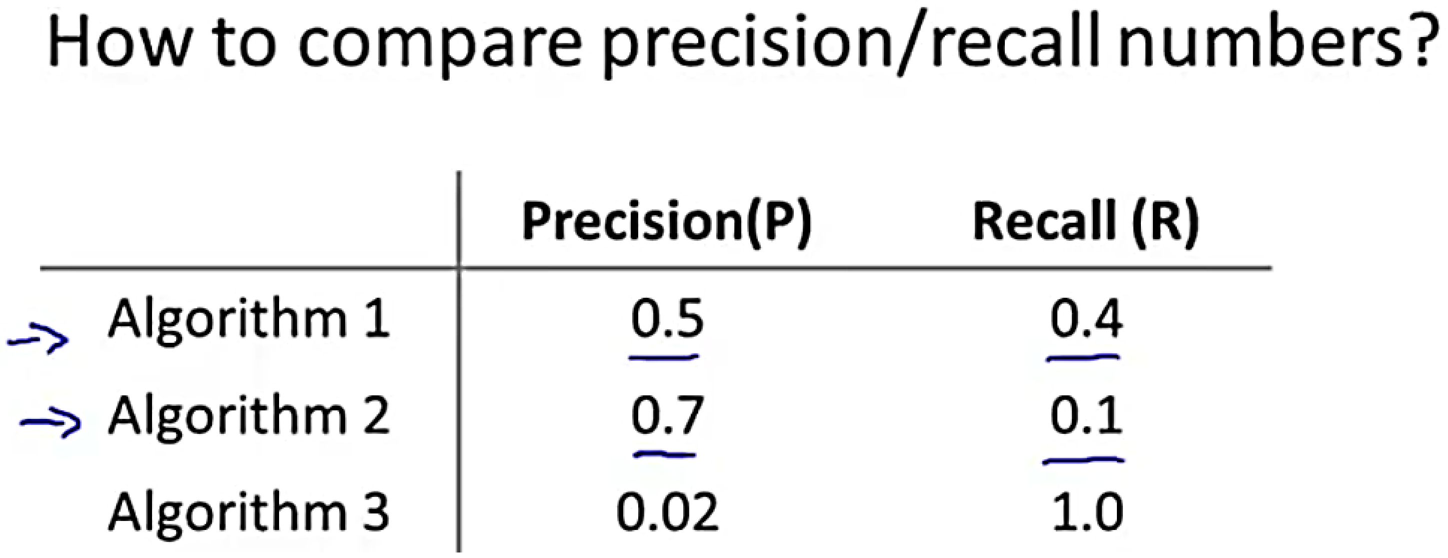
→ Higher recall, lower precision.

More generally: Predict 1 if $\boxed{h_\theta(x) \geq \text{threshold}}$



How to compare precision recall values bw two algos:

Previously we had a single no metric for error analysis, but now we have two nos.

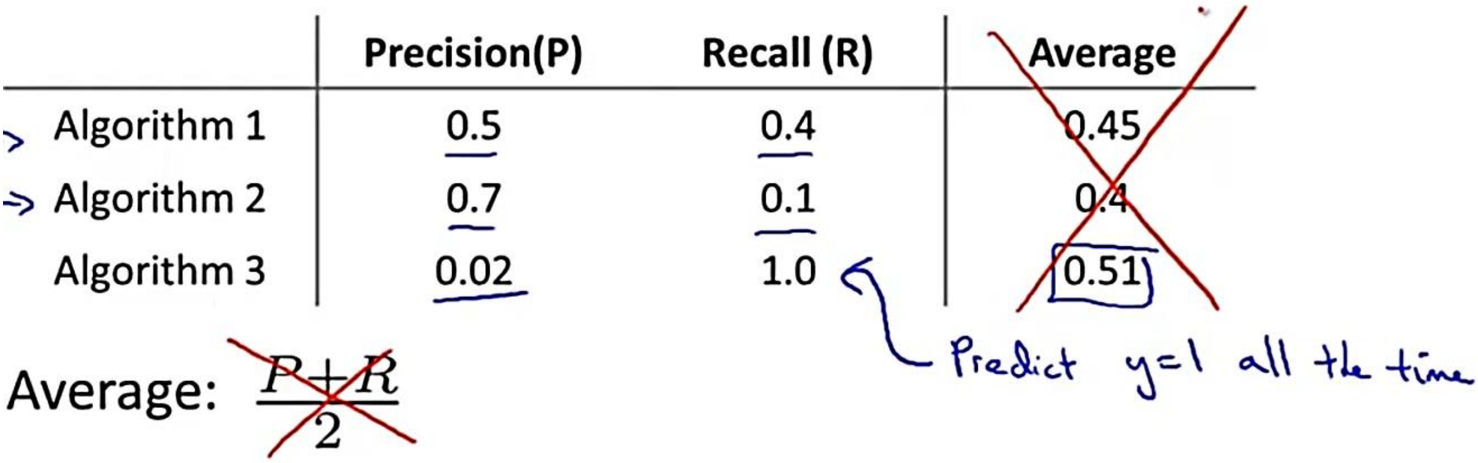


How to compare precision/recall numbers?

	Precision(P)	Recall (R)
Algorithm 1	0.5	0.4
Algorithm 2	0.7	0.1
Algorithm 3	0.02	1.0

How to determine which is better?

Taking average is not a good choice



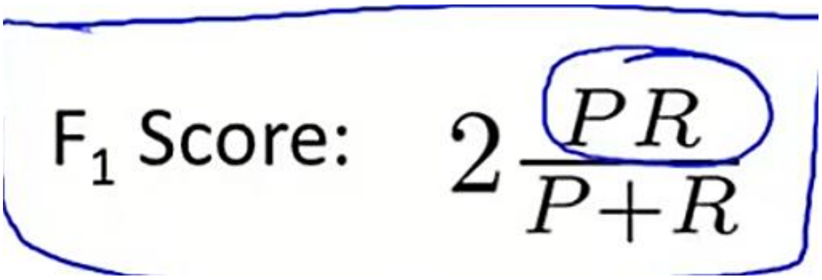
	Precision(P)	Recall (R)	~~Average~~
Algorithm 1	0.5	0.4	~~0.45~~
Algorithm 2	0.7	0.1	~~0.4~~
Algorithm 3	0.02	1.0	~~0.51~~

Predict y=1 all the time

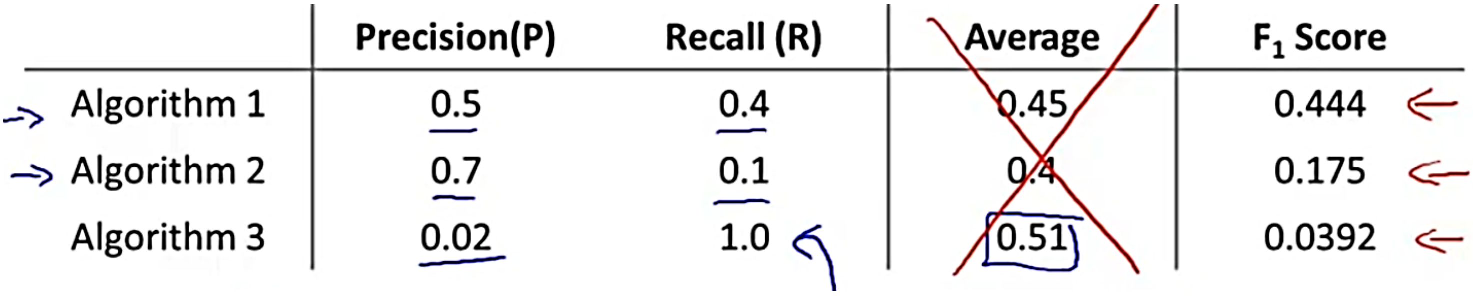
Average: $\cancel{\frac{P+R}{2}}$

A NEW EVALUATION METRICS:

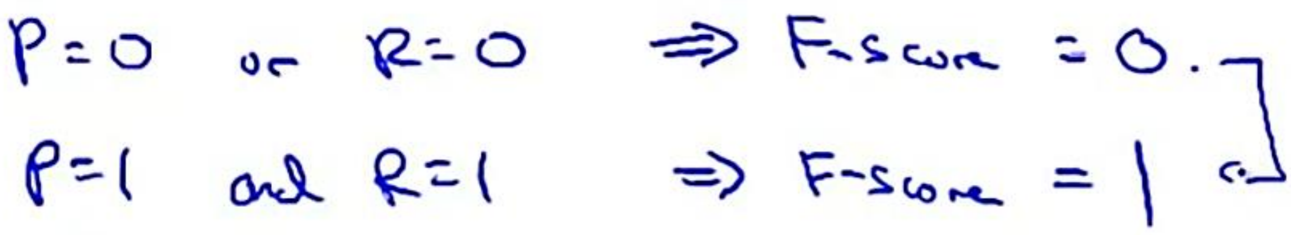
F$_1$ score: also called F score:



$$F_1 \text{ Score: } \quad 2\frac{PR}{P+R}$$



	Precision(P)	Recall (R)	~~Average~~	F$_1$ Score
Algorithm 1	0.5	0.4	~~0.45~~	0.444
Algorithm 2	0.7	0.1	~~0.4~~	0.175
Algorithm 3	0.02	1.0	~~0.51~~	0.0392



$P=0$ or $R=0$ $\Rightarrow$ F-score $=0$.

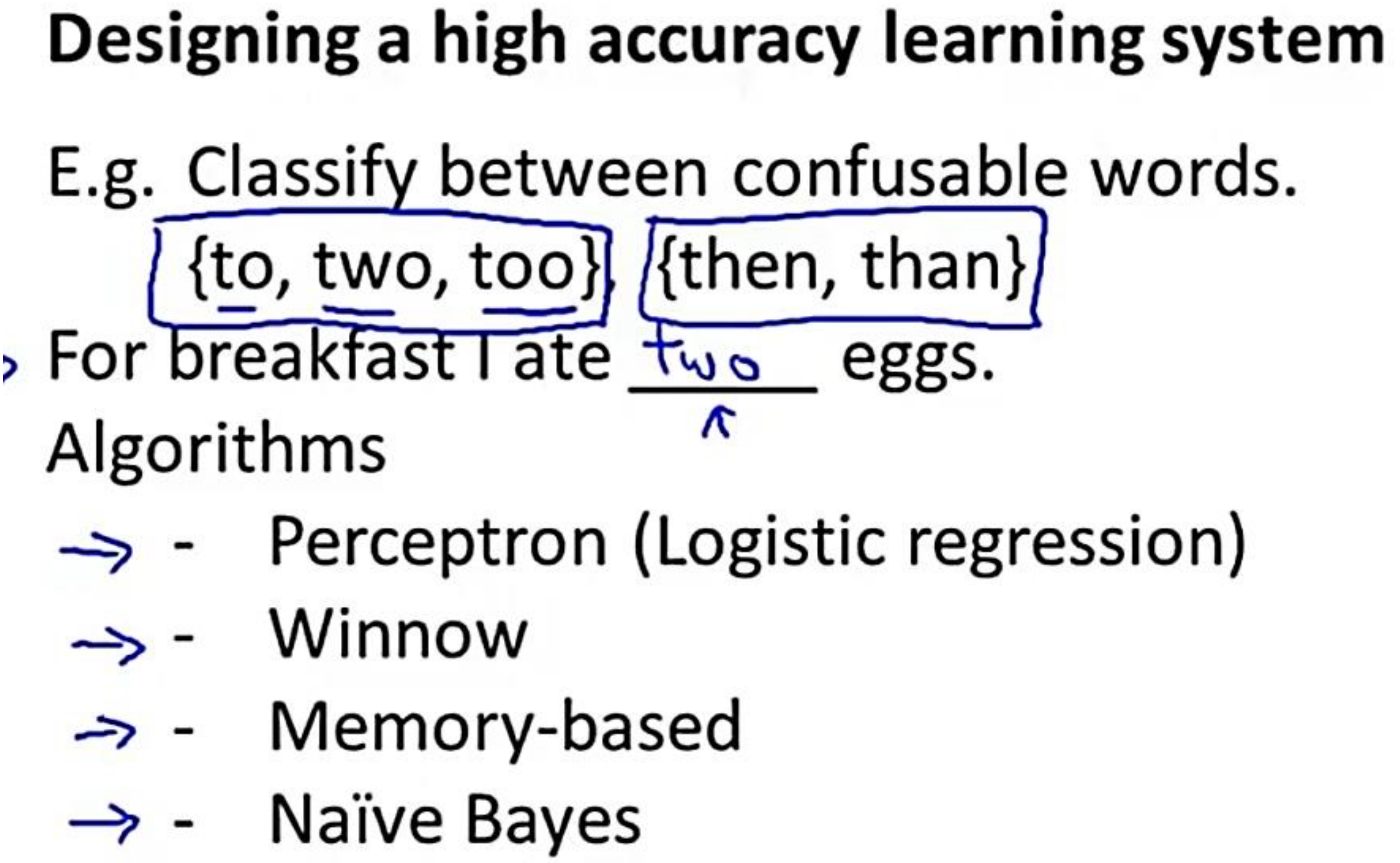
$P=1$ and $R=1$ $\Rightarrow$ F-score $=1$

- To pick a threshold, try different values of threshold and one with the best F-score is to be picked.

IMPORTANCE OF DATA IN MACHINE LEARNING:

Problems like confusable words problem are hard to entertain.

So what can we do to solve that ?



Designing a high accuracy learning system

E.g. Classify between confusable words.

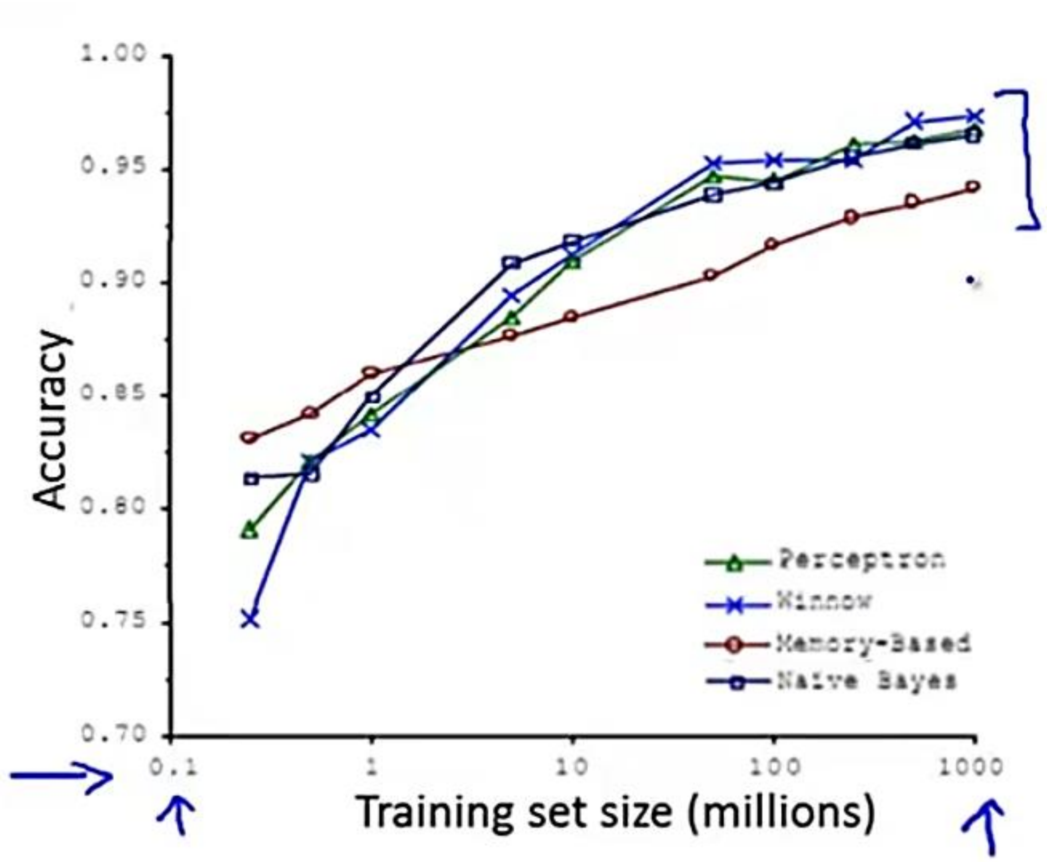
{to, two, too} {then, than}

For breakfast I ate two eggs.

Algorithms

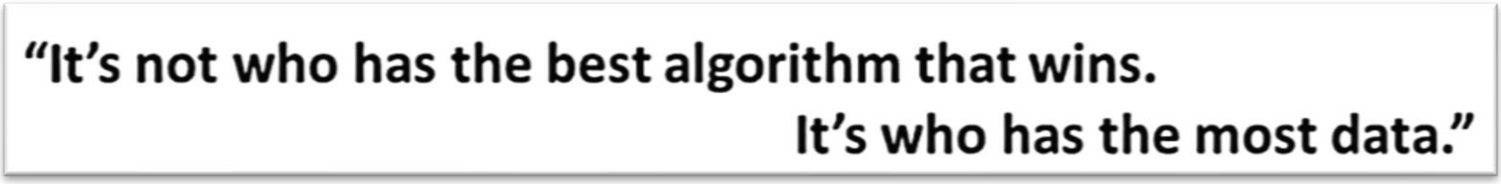
- Perceptron (Logistic regression)
- Winnow
- Memory-based
- Naïve Bayes

These algos were tried on different sizes of training data:



All of them give approx. same accuracy on a large dataset

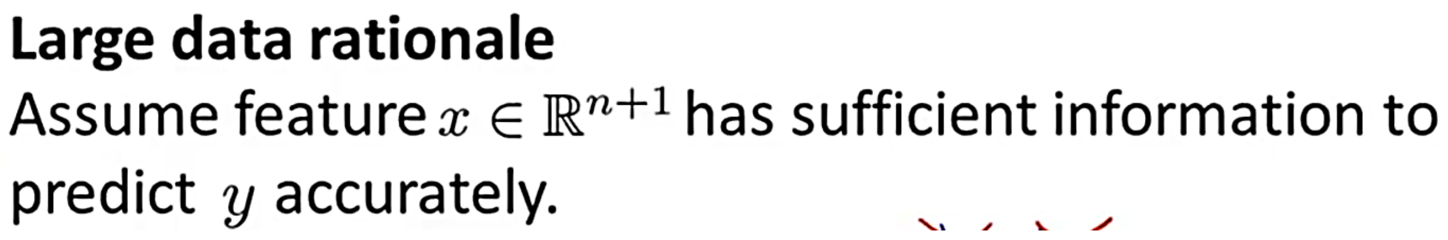
It shows that:



"It's not who has the best algorithm that wins.

It's who has the most data."

So, for confusable words problem:

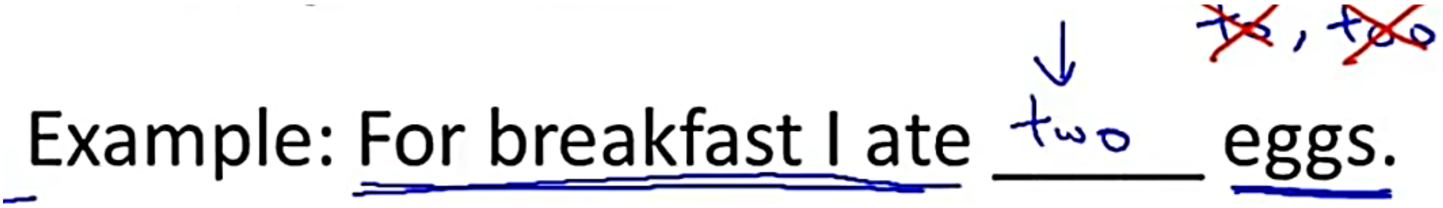


Large data rationale

Assume feature $x \in \mathbb{R}^{n+1}$ has sufficient information to predict y accurately.

We have enough features that capture the surrounding words

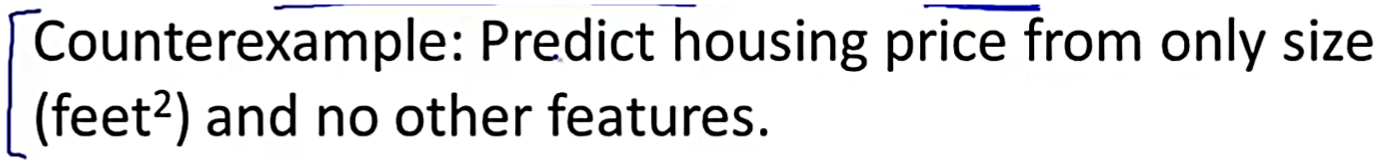
- So, that's enough info to predict the right answer



~~to~~, ~~too~~

Example: For breakfast I ate two eggs.

As a counter example:



Counterexample: Predict housing price from only size (feet^2) and no other features.

We don’t have features like: how old the house is?

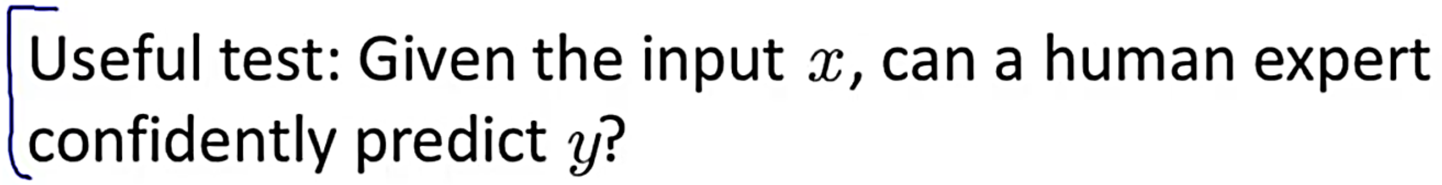
Does it have furniture?

Location?

Etc…

We have just the size!

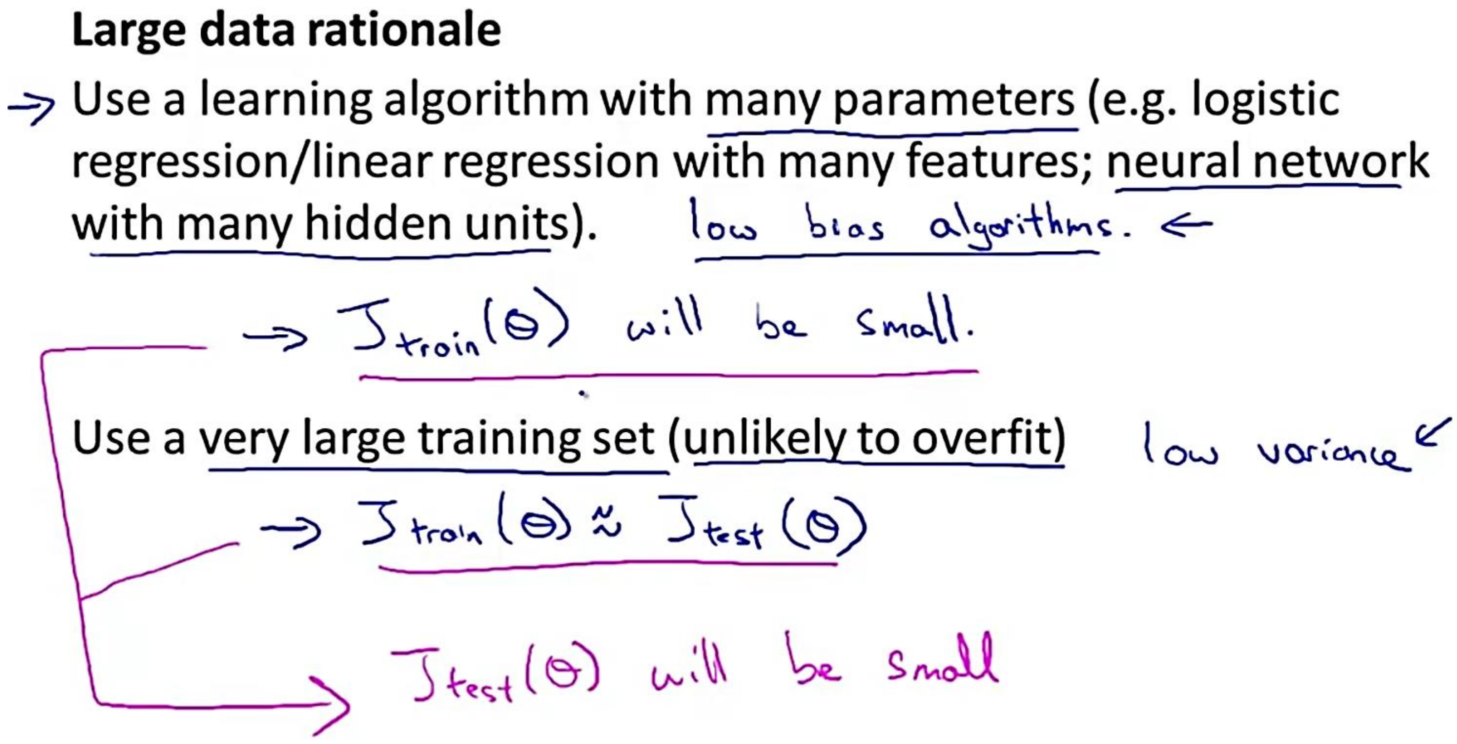
An approach to determine if its possible to predict the o/p with given set of features:



Useful test: Given the input x, can a human expert confidently predict y?

- In case of confusable words, a human expert can easily tell the missing word accurately
- In case of housing price example: even a real estate expert cant tell the exact priec of house based on just the size.

So, can having the lot of data help?



Large data rationale

→ Use a learning algorithm with many parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units). low bias algorithms. ←

→ $J_{train}(\theta)$ will be small.

Use a very large training set (unlikely to overfit) low variance ←

→ $J_{train}(\theta) \approx J_{test}(\theta)$

→ $J_{test}(\theta)$ will be small

- ➢ Assuming we have many features and a rich class of functions → low bias
- ➢ Assuming we have huge huge dataset → low variance

→ This makes up for a very high performance algorithm.