Machine Learning
Week 6 Section 2
Machine Learning System Design

To optimize a machine learning algorithm, you'll need to first understand where the biggest improvements can be made. In this module, we discuss how to understand the performance of a machine learning system with multiple parts, and also how to deal with skewed data.

<u>Building a Spam Classifier</u>

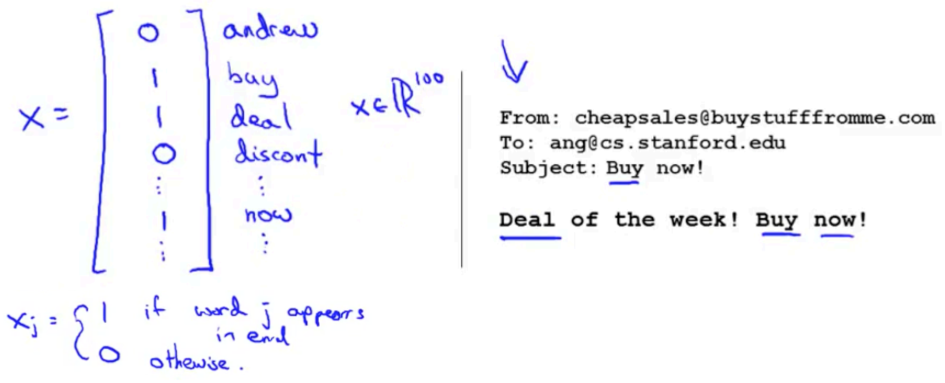
Prioritizing What to Work On

How do we strategize what to work on when putting together a complex machine learning system?

Ex. Building a spam classifier

The first decision we must make is deciding what our set of 'X' features will be.

- Should we mark specific words as indicative of spam?
  - Ex. Deal, buy, misspelt words



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

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

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

Deal of the week! Buy now!
```

**Note in practice, we take the most frequently occurring n words (10,000 to 50,000) in a training set rather than picking ~100 words manually.

How should you spend your time to make your spam classifier have low error?

1. Collect lots of data
2. Develop sophisticated features based on email routing information (from email headers)
3. Develop sophisticated features for the message body, ex. Should "discount" and "discounts" or "deal" and "dealer" be treated as the same word? What about punctuation?
4. Develop sophisticated algorithm to detect misspelt words ex. m0rtgage, med1cine, w4tches

There is not a good way to tell which approach is the best way to spend your time.

Often people will pick and fixate on just one option.

Whatever you do, don't just pick an option based on your gut feeling.

What you should do is try to create a list of all the possible solutions / approaches you can think of and then begin to perform basic error analysis (next video).

Error Analysis

The best approach is to systematically make some of these decisions.

Recommended approach:

- Start with a simple algorithm that you can implement quickly (and dirty).
    - Spend only like a single day on it and test it on your cross-validation error.
- Plot Learning Curves to decided if more data, more features, regularization, etc. are likely to help.
- Error Analysis:
    - Manually examine the examples. (In the 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 which may provide insight on the shortcomings of your algorithm.
        - Ex. See which feature cues you think would have helped / didn't help.

This helps us avoid 'premature optimization' and allows us to let evidence guide our decisions instead of wasting our time pursuing gut feelings or guesses.

This is the importance of 'numerical evaluation'.

'Stemming software':

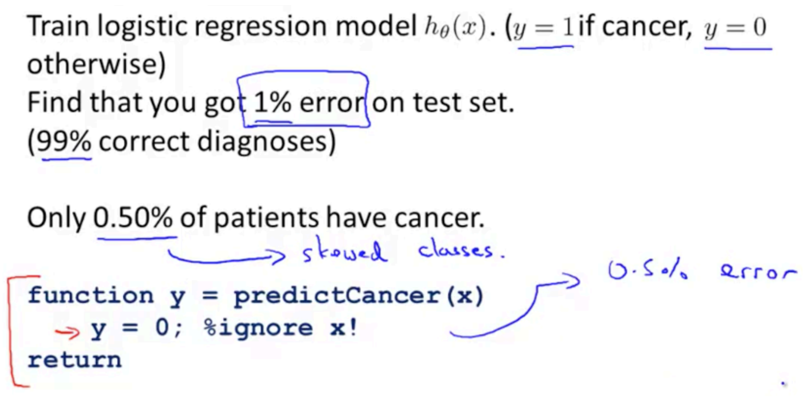
- Ex. Treating discount/discounts/discounted/discounting as the same word.
- Ex. Mom / mom as the same word.
- May not always be helpful but can help in some cases.
    - Ex. Universe / university should not be treated as the same word.
- Briefly try it and see what it does to performance with / without stemming.

It is important to look at a single, numerical value (the cross validation error) as an 'error metric' to quickly gauge performance during this process.

Error Metrics for Skewed Classes

What are skewed classes?

Let us revisit the cancer classification problem.



Train logistic regression model $h_\theta(x)$. ($\underline{y=1}$ if cancer, $\underline{y=0}$ otherwise)
Find that you got $\boxed{1\% \text{ error}}$ on test set.
($\underline{99\%}$ correct diagnoses)

Only $\underline{0.50\%}$ of patients have cancer. $\longrightarrow$ skewed classes.

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

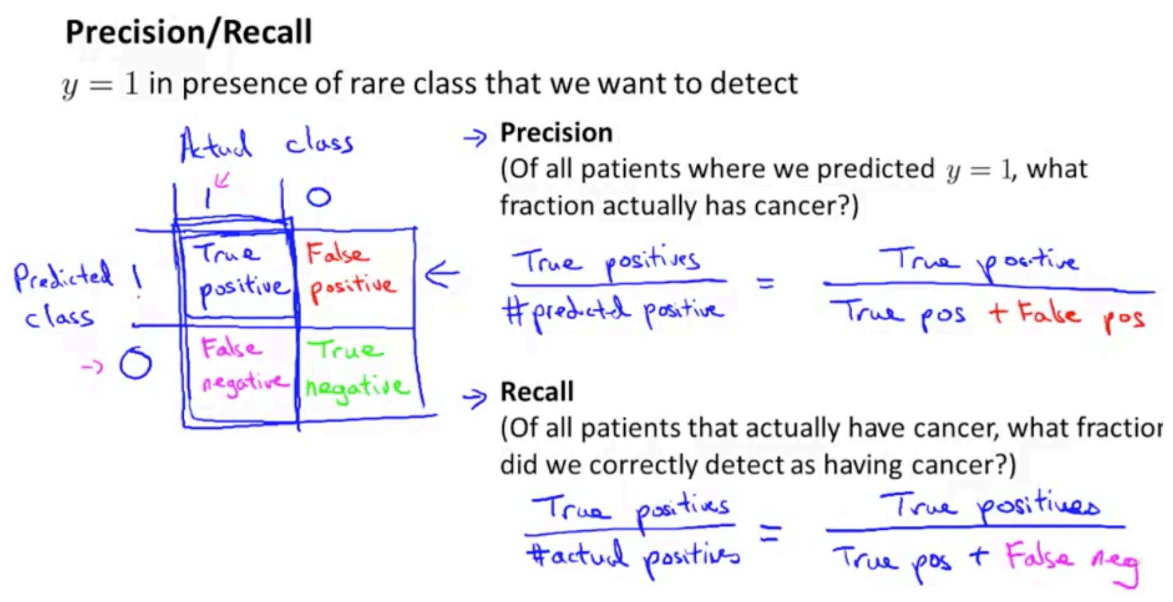
$\longrightarrow$ 0.5% error

Oh no! That is terrible.

If you have a classification problem with skewed classes and make changes that 'improve the error' from 99.2% accuracy to 99.5% accuracy, did you really 'improve' the algorithm or just make something that predicts '0' more often?

When you are faced with such skewed classes, simply looking at a single error metric like the cross validation error is no longer a good practice.

One evaluation metric we can use instead is called 'precision / 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 negative	True negative

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 fractio did we correctly detect as having cancer?)

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

Both high precision and high recall are 'good'.

If we applied this to the cancer classifier above using the function to just predict 0 every time, we would quickly see that the recall = 0 which shows us that the classifier is very bad.

For any data set that we have very skewed classes, there isn't a way for the classifier to cheat this metric and have a high recall / precision when in fact it is performing poorly.

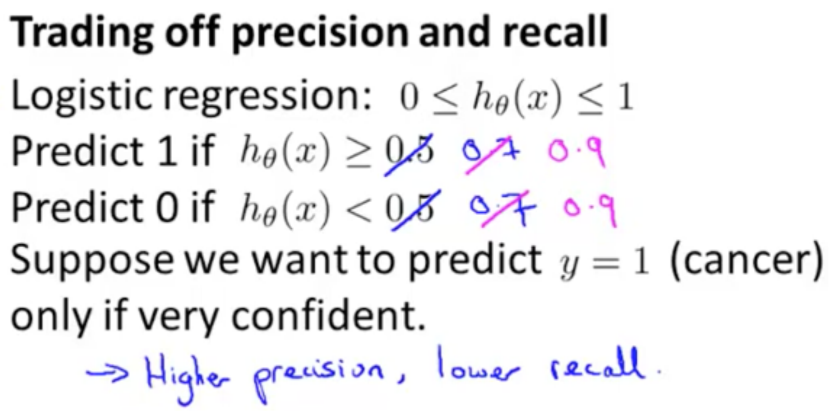
**Note we use the convention y = 1 in the presence of a rare class that we want to detect when we have skewed classes.

Trading Off Precision and Recall

For many applications, we will want to control the trade-off between precision and recall.

Let us continue the cancer classification problem:

What if we only want to predict 1 if the confidence is very high? Ex. Only tell someone they have cancer if our confidence is very high that they do have it.



Trading off precision and recall

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

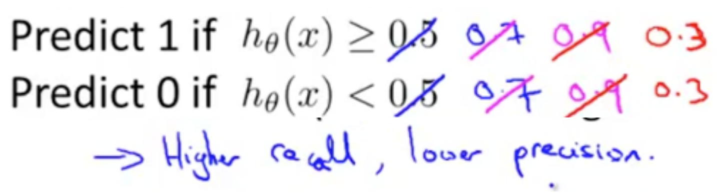
Predict 1 if $h_\theta(x) \ge \cancel{0.5}\ \cancel{0.7}\ 0.9$

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

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

$\rightarrow$ Higher precision, lower recall.

But on the flip side, what if we want to be safer and rather tell people that they do have cancer when they might not in order to make sure they all get treatment in case they all may have cancer? (Avoid false negatives)

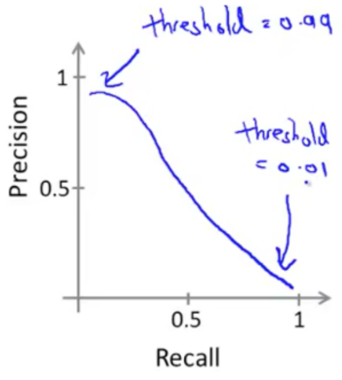


Predict 1 if $h_\theta(x) \ge \cancel{0.5}\ \cancel{0.7}\ \cancel{0.9}\ 0.3$

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

$\rightarrow$ Higher recall, lower precision.

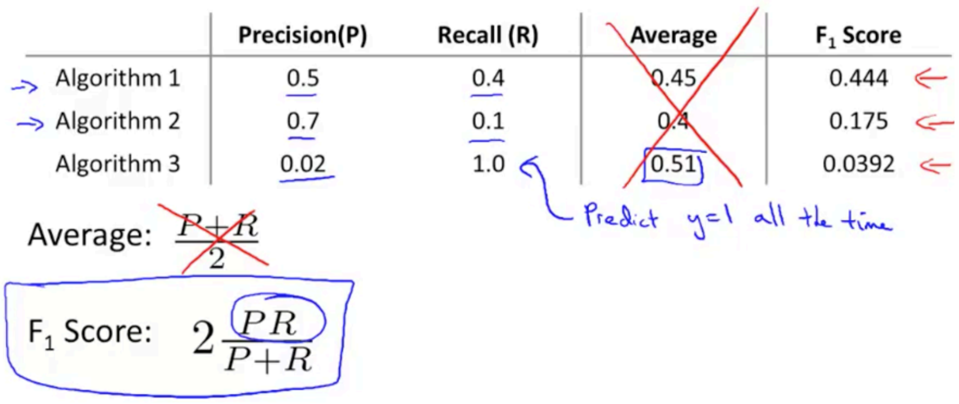
More generally: depending on whether we want higher precision or higher recall, we predict 1 if h_theta ≥ threshold



** Very idealized curve, can have many different shapes.

As such, is there a way to choose some value for our threshold automatically?

How do we compare different precision / recall numbers?



	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	~~$\boxed{0.51}$~~	0.0392 ←

Predict y=1 all the time

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

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

If you end up stuck on thinking / deciding which precision & recall values are better for your algorithm influencing which changes to make, what do you do?

Looking at the average is a terrible idea.

One possible method is to look at the 'F score' / 'F1 score'.

A perfect F score would be P = 1, R = 1, => F = 1

Note there are many other possible 'scoring' methods we could use but this is a typical one used in ML.

Using Large Datasets: Data for Machine Learning

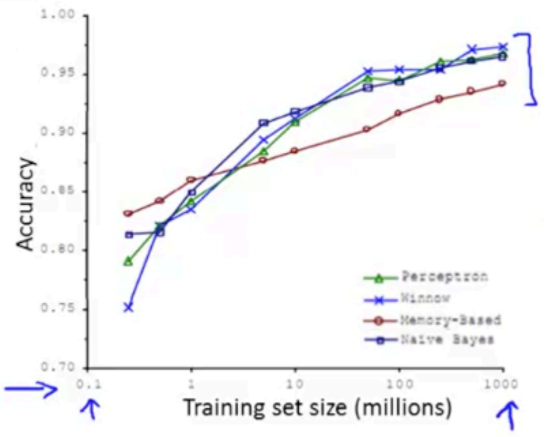
In machine learning system design, we face the issue of deciding how much data to train on.

It is not a good idea to simply spend a lot of time collecting lots of data, but there are certain conditions in which getting a lot of data and training on a certain type of learning algorithm can be a good way to create a high performance algorithm.

Story time: Banko and Brill 2001

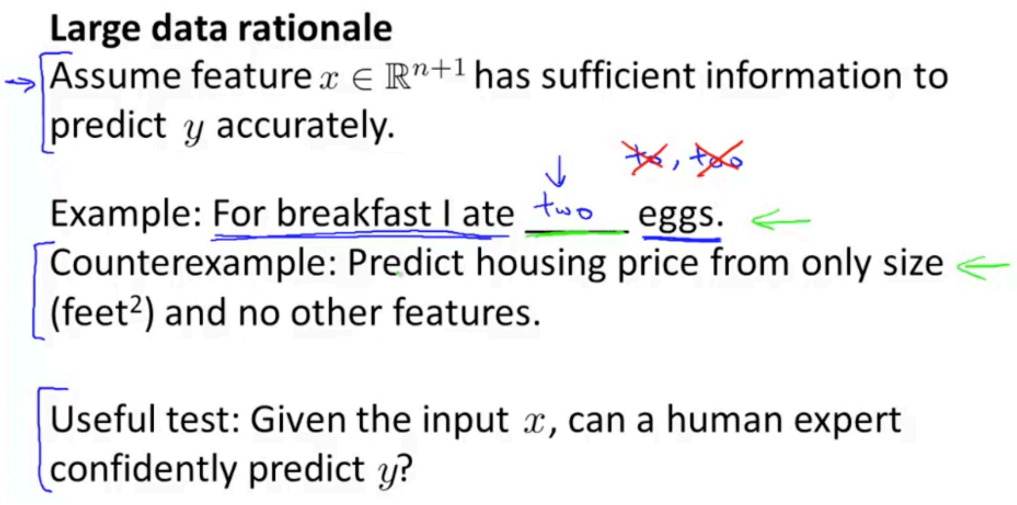
Classifying confusable words - different algorithms vs. lots of data.

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



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

[Banko and Brill, 2001]



Large data rationale

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

~~to~~, ~~too~~

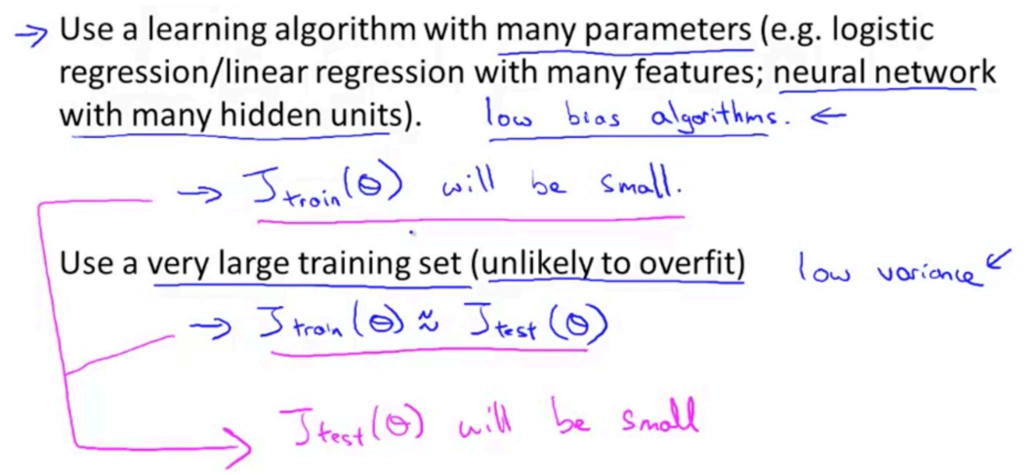
Example: For breakfast I ate two eggs.

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

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

Super good test:
1. Can a human expert look at the feature set X and make an accurate prediction y?
2. Can we find a very large data set?

Maintaining this assumption that the features have sufficient information to make an accurate prediction:



→ 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

Putting these together we are able to create a low bias low variance algorithm that will perform well on the test set.