# 11. Machine learning
## - System design

## Prioritizing what to work on:

### Building a spam classifier:

- Supervised learning
- $x$ = Features of mail

  $y$ = spam $(1)$, no spam $(0)$

Features: choose 100 word indicative of spam/not spam

e.g. deal, discount, buy, andrew, now ...

$$x = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \begin{matrix} \text{andrew} \\ \text{buy} \\ \text{deal} \\ \text{discount} \\ \vdots \\ \text{now} \end{matrix}$$

verify if this words occure in mail

Note:

In practice, take most frequently occuring $n$ words (10'000 - 50'000) in training set, rather then picking 100 words manually.

$x \in \mathbb{R}^{100}$

$$x_j = \begin{cases} 0 & - \text{ if word does not} \\ 1 & \text{appear in email, } 1 \text{ otherwise} \end{cases}$$

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 messages body, should "discount" and "discounts" be treated as the same word? How about "deal" and "Dealer"? Features about punctuation!
- Develop sophisticated features to detect misspellings (e.g. m0rtgages, med1cine, w4tches.)

## Recommended approach:

- Start with an 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
  that your algorithm made errors on. See if you spot any
  systematic trend in what type of examples it is making errors on.

  → let evidence guide your decisions.　　　　　　try to spot systematic
  　　　　　　　　　　　　　　　　　　　　　　　　　　　errors?

## Error analysis

$m_{cv}$ = 500 examples in cross validation set

Algorithm misclassifies 100 emails

Manually examine the 100 errors, and categorize them based on:
  1) What type of email it is　→ eg. pharma, replica, steal pw...
  2) What cues (features) you think would have helped the algorithm
     classify them correctly.

| | | | |
|---|---|---|---|
| Pharma: | 12 | Deliberately misspelling: | 5 |
| Replica/fake: | 4 | Unusual email routing: | 16 |
| Steal pw: | 53 → target this problem | Unusual punctuation: | 32 |
| Others: | 31 | | |

## The importance of numerical evaluation:

- Should discount / discounted / discounting to be treated as the same word?
  ↳ Can use "stemmer" software → eg. "Porter stemmer", but this can
    also be missleading　universe / university
- Error analysis might not be helpful for deciding if this is likely to improve
  performance. Only solution is to try and see if it works.
- Need numerical evaluation (e.g. cross val. error) of algorithms with and
  without stemming.　　　without stemming: 5% err.　with stemming: 3% err.
  Distinguish upper vs. lower case (Mom/mom) → 3% → 2% ...

## Cancer classification example

Train logistic regression problem model $h_\theta(x)$. $y=1$ if cancer, $y=0$ if not
Find that you get 1% error on the test set. (99% correct diagnoses)
=◦ Only 0,5% of patients have cancer.
↳ skewed classes      =◦ a lot more y=0 classes as y=1

function y = predictCancer (x)
    y=∅;   % ignore x!                    =◦ Error 0,5%
    return                         not a good approach

## Precision / Recall

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

|  | Actual class | |
|---|---|---|
|  | 1 | 0 |
| Predicted 1 | True positive | False positive |
| Class 0 | False negative | True negative |

$y = \emptyset$
Recall $= \emptyset$

### Precision

Of all patients where we predicted
$y=1$, what fraction actually has cancer?

$$\frac{\text{True pos.}}{\#\text{predict. pos}} = \frac{\text{True positives}}{\text{True pos. + False pos.}}$$

↳ high precision would be good

### Recall

Of all patients that actually have cancer,
what fraction did we correctly detect as
having cancer?

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

$y=1$ by presence of rare class ?

# Trading off Precision and Recall

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

Predict 1 if $h_\theta(x) \geq 0.5 \Rightarrow 0.7 \Rightarrow 0.9$
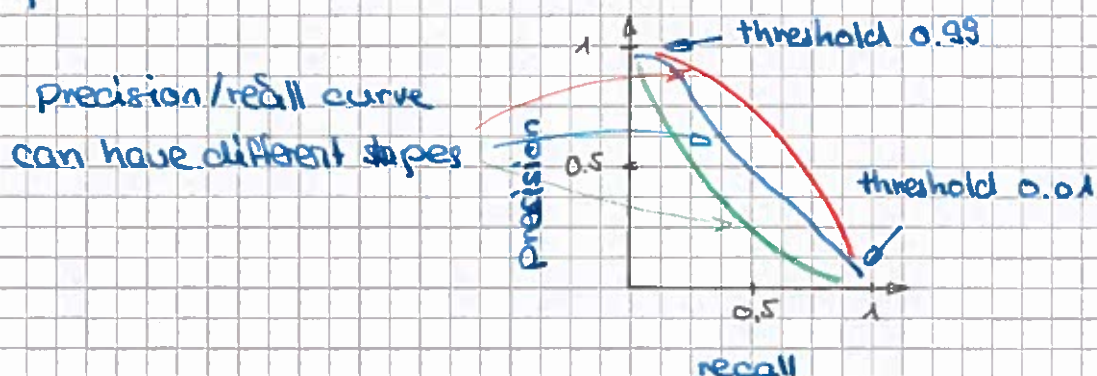Predict 0 if $h_\theta(x) < 0.5 = 0.7 \Rightarrow 0.9$

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

↳ setting thresholds to 0.7 or 0.9 ⇒ higher precision, lower recall

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

→ more conservative

↳ setting thresholds to 0.3 ⇒ higher recall, lower precision

More generally: Predict 1 if $h_\theta(x) \geq$ threshold

Precision/recall curve
can have different shapes



## F₁ Score (F score)

How to compare precision/recall numbers?

|  | Precision (P) | Recall (R) | Average | F₁ 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 ⇒ predict y=1 all the time ⇒ not very good |

$$Average = \frac{P+R}{2}$$

$$F_1\ Score = 2 \cdot \frac{PR}{P+R}$$

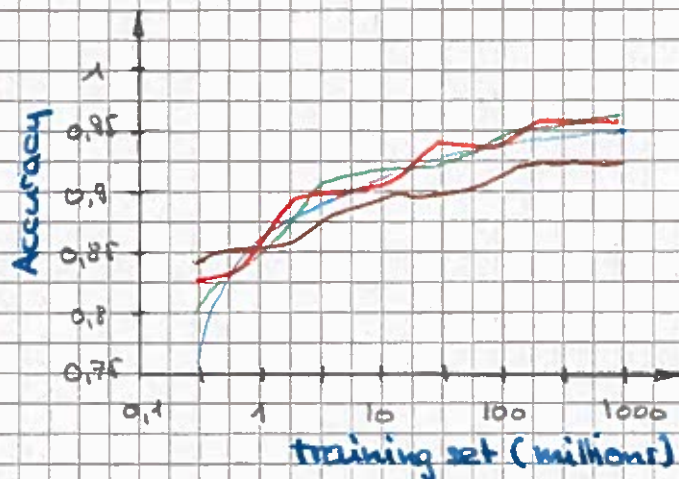$P = 0$ or $R = 0$ ⇒ F Score $= 0$
$P = 1$ and $R = 1$ ⇒ F Score $= 1$

Designing a high accuracy learning system

e.g Classify between confusable words $\{to, two, too\}$, $\{then, than\}$
→ For breakfast I ate ___ eggs.

Algorithms:
- Perception   —
- Winnow        —
- Memory based  —
- Naïve Bayes   —



Accuracy vs training set (millions)

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


## Large data rationale

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

Example: For breakfast I ate ___ eggs.    to /two / too

Counterexample: Predict housing prices from only size (feat²)
                and no other features.

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

## 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).

$\Rightarrow$ low bias algorithm

$\Rightarrow$   $J_{train}(\Theta)$  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