Classification (分類)

Naoaki Okazaki

okazaki at ecei.tohoku.ac.jp

http://www.chokkan.org/

http://twitter.com/#!/chokkanorg

http://www.cl.ecei.tohoku.ac.jp/index.php?InformationCommunicationTheory

Spam mail


I'm interested in your profile - Mozilla Thunderbird
ファイル(F) 編集(E) 表示(V) 移動(G) メッセージ(M) ツール(T) ヘルプ(H)
受信 作成 アドレス帳 タグ
返信 全員に返信 転送 アーカイブ 迷惑マークを付ける 削除
差出人 Marquis Winn <automail@interloper.com>
件名 I'm interested in your profile
2011/07/12 19:04
宛先 Naoaki Okazaki
その他の操作
Hello!
I am Victoriya, I am 27 y.o.
I search for boy-friend.
Ny photos and mew message for you here:
http://sonya201010.com.ua/?message_from=Victoriya

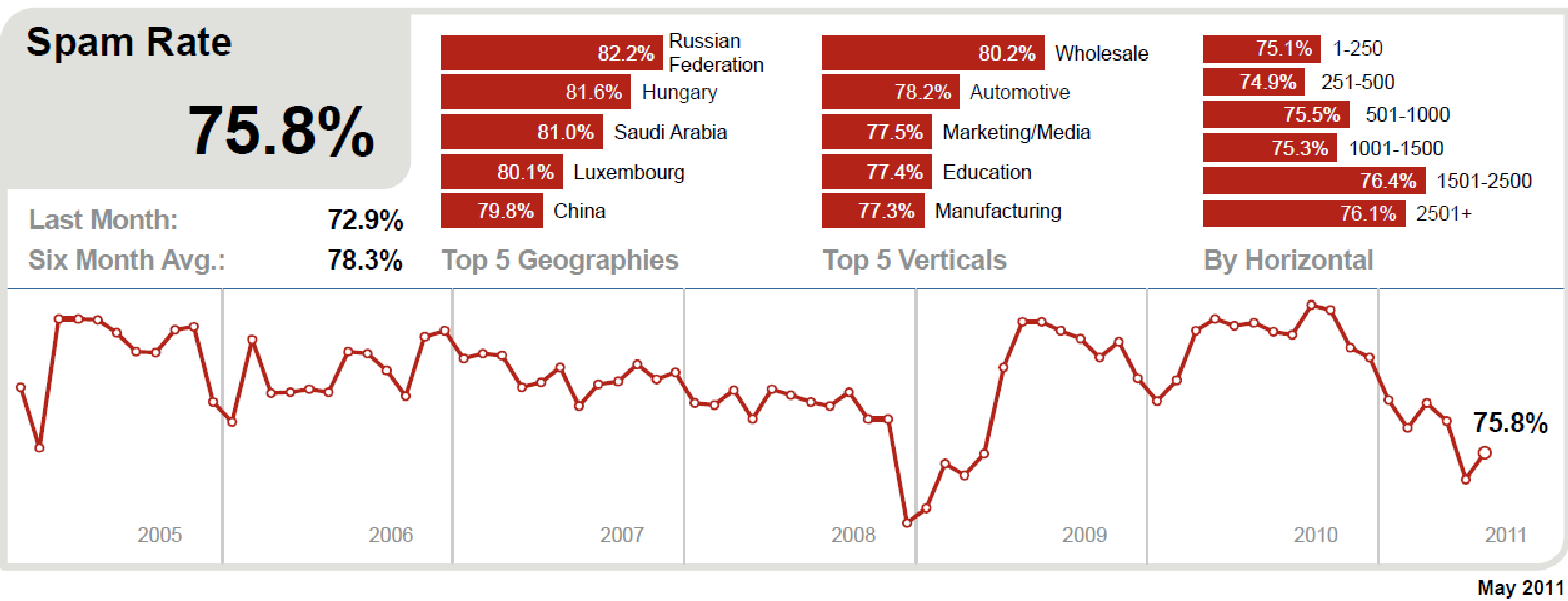
How many spams do you get (per day)?

- My office address: [okazaki at ecei.tohoku.ac.jp]()
  - 10 spams out of 80 emails
  - 5 filtered automatically out of 10 spams
- My private address (leaked by a company)
  - 40 spams out of 40 emails
  - 10 filtered out of 40 spams


Spam (Monty Python)

Symantec's survey

- May 2011 MessageLabs Intelligence Report
  - 75.8% of email in the world was spam (72.3% in Japan)
  - 1 in 1.32 emails was spam
  - http://www.symanteccloud.com/mlireport/MLI_2011_05_May_FINAL-en.pdf



Rule-based spam filtering

- Design heuristic rules to detect spams

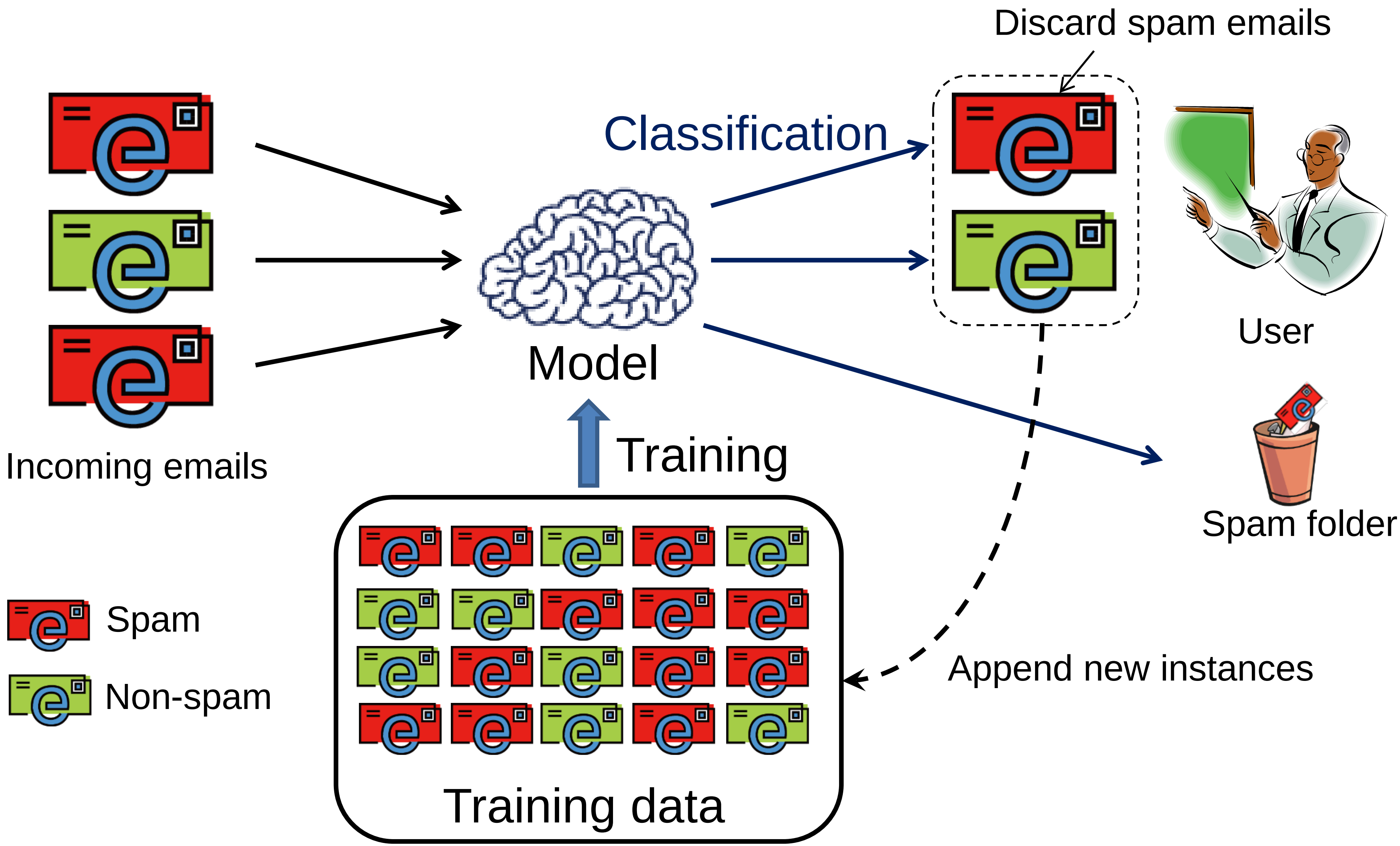
```
def is_spam(text):
    if text.find('my photo attached') != -1:
        return True
    if text.find('very cheap drugs') != -1:
        return True
    ...
    return False
```

Difficult to scale to a wide-variety of spams

- Pros
  - Good initial cost-performance
  - Understandable internals
  - Configurable internals

- Cons
  - High maintenance cost
  - Dependence to the domain
  - Artisanal skill

Supervised spam filtering (with retraining)



Supervised approach

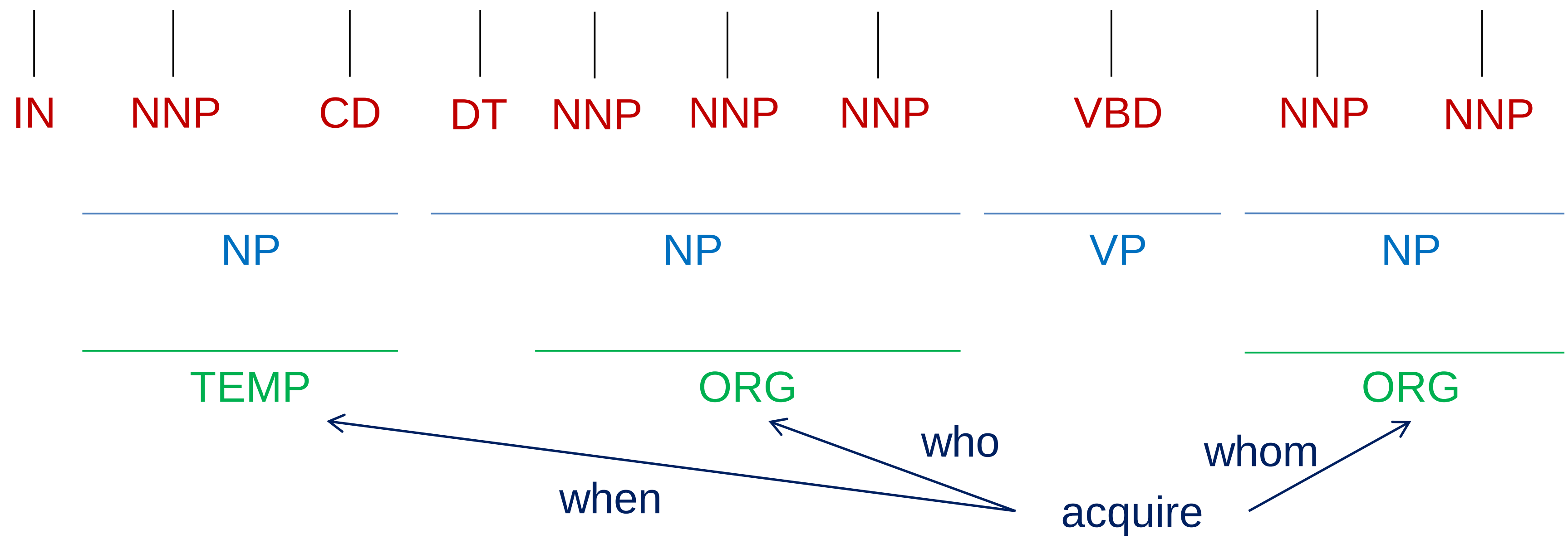
- Learn from examples (supervision data)
  - Spam (positive) and non-spam (negative) emails
- Acquire rules from the supervision data
  - Rules (features) are usually induced from the supervision data
    - e.g., n-grams of mail contents, mail headers
  - (Deliberately) over-generate features, regardless of effectiveness
- Weight rules in terms of their contribution to the task

- Hand-crafted rules for the domain are unnecessary (*except for feature engineering*)
- This approach work surprisingly well if we have a large supervision data

Classification and NLP

- *Many NLP problems can be formalized as classification!*

In March 2005, the New York Times acquired About, Inc.



Today’s topic

- Linear binary classifier
- Feature extraction
  - Tokenization, stop words, stemming, feature extraction
- Training – perceptron
- Training – logistic regression (using SGD)
- Other formalizations
- Evaluation
- Implementations and experiments

Take home messages

- A text is represented by *features* (as a vector)
- *Linear classifier* simply computes the linear combination of *feature weights* as a score
- Training a linear classifier is very straightforward
  - In principle, if the classifier fails with an instance, we update feature weights such that the classifier can classify the instance next time!
- Two kinds of classification failures that have trade-offs:
  - *False positives* decrease *precision*
  - *False negatives* decrease *recall*

Brief Introduction of Linear (Binary) Classifier

Linear (binary) classifier (線形二値分類器)

- Input: feature vector $\boldsymbol{x} \in \mathcal{R}^m$
- Output: prediction $\hat{y} \in \{0,1\}$
- Using: weight vector $\boldsymbol{w} \in \mathcal{R}^m$

$$\hat{y} = \begin{cases} 1 & (\text{if } a(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} > 0) \\ 0 & (\text{otherwise}) \end{cases}$$

(m: number of dimension)

- More concrete example:
  - Feature space ($m = 6$): (darling, honey, my, love, photo, attached)
  - Input: "*Hi darling, my photo in attached file*" $\xrightarrow{\text{Feature extraction}}$ $\boldsymbol{x} = (1\ 0\ 1\ 0\ 1\ 1)$

$$a(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} = \sum_{i=1}^{m} w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

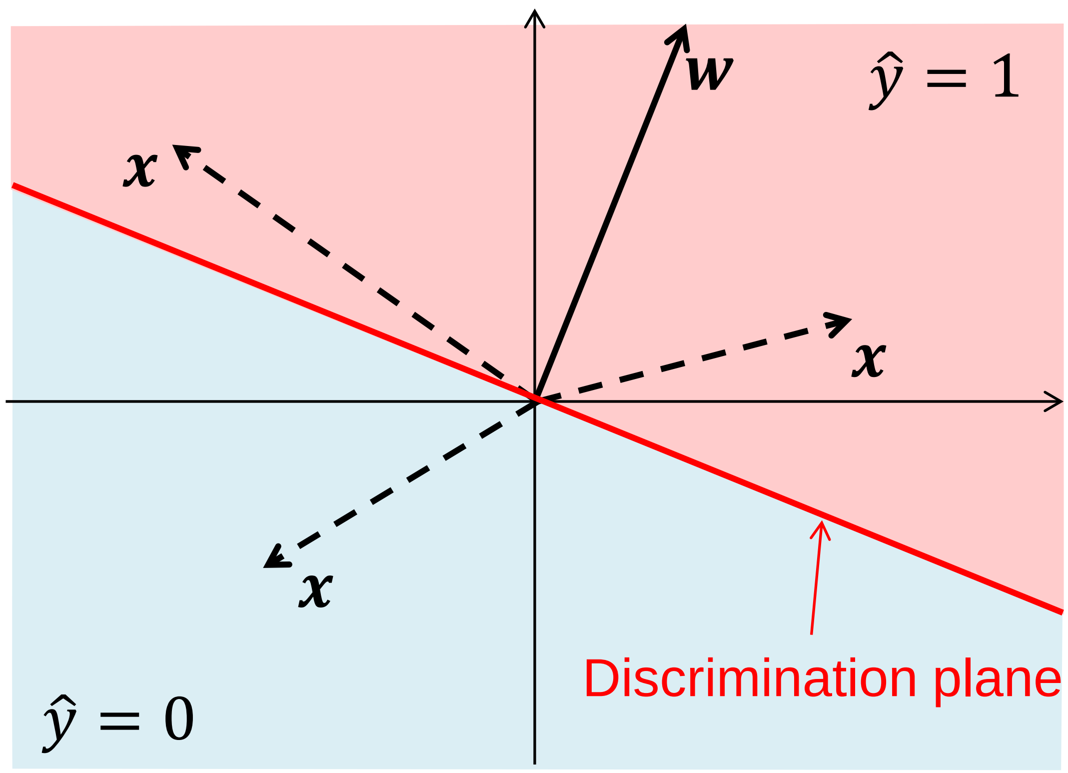
Contribution of each feature
(positive high: likely to yield $\hat{y} = 1$)

  - Output: the sign of $a(\boldsymbol{x})$
  - Training: to find $\boldsymbol{w}$ such that it *fits* well to the training data

Discrimination plane (分離平面)

- Draw the region of $\boldsymbol{x}$ such that $\hat{y} = 1$

$$\boldsymbol{w} \cdot \boldsymbol{x} > 0 \iff (|\text{angle between } \boldsymbol{w} \text{ and } \boldsymbol{x}| < \frac{\pi}{2})$$



Two important questions

- *Feature extraction* (feature engineering)
  - Represent an input text with a feature vector
  - Using a number of NLP techniques: tokenization, stop-word, stemming, part-of-speech tagging, parsing, dictionary lookup, etc.
  - Important: *a linear model can see a text only through features!*

- *Training*
  - Find a weight vector such that it fits well to the training data
    - The classifier is expected to re-predict all training instances correctly
  - Fitness (formalizations; loss functions)
    - Logistic Regression, Support Vector Machine (SVM), …
  - Finding (training algorithms)
    - Perceptron, Gradient Descent, Stochastic Gradient Descent, …

Feature Extraction

Representing a text with an input vector $\boldsymbol{x}$

Preparing a space of feature vectors

Tokenization

- Split a sentence into a sequence of tokens (words)
- In English, tokens are separated by whitespaces (e.g., space, punctuation characters)
    - Sophisticated methods (e.g., word segmentation, morphological analysis) are necessary for Japanese and Chinese
        - Token boundaries are not obvious in these languages
- Penn Treebank tokenization
    - http://www.cis.upenn.edu/~treebank/tokenization.html

```
Hi darling, my photo in attached file
```

⇓ Tokenize and lowercase

```
[“hi”, “darling”, “,”, “my”, “photo”, “in”, “attached”, “file”]
```

Stop words

- Remove words that are irrelevant to the processing
  - E.g., http://www.textfixer.com/resources/common-english-words.txt
    - a,able,about,across,after,all,almost,also,am,among,an,and,any,are,as,at,be,because,been,but,by,can,cannot,could,dear,did,do,does,either,else,ever,every,for,from,get,got,had,has,have,he,her,hers,him,his,how,however,i,if,in,into,is,it,its,just,least,let,like,likely,may,me,might,most,must,my,neither,no,nor,not,of,off,often,on,only,or,other,our,own,rather,said,say,says,she,should,since,so,some,than,that,the,their,them,then,there,these,they,this,tis,to,too,twas,us,wants,was,we,were,what,when,where,which,while,who,whom,why,will,with,would,yet,you,your
- Highly domain dependent
  - “my” in the phrase “my photo” may be effective to spam filtering
  - This lecture employs punctuations and prepositions as stop words

```
[“hi”, “darling”, “,”, “my”, “photo”, “in”, “attached”, “file”]
```

⇩ Remove punctuations and prepositions

```
[“hi”, “darling”, “my”, “photo”, “attached”, “file”]
```

Stemming

- Reducing inflected and derived words to their stems
  - Stems are not identical to (morphological) base forms
  - It is sufficient for an algorithm to map related words into the same string, regardless of its morphological validity

- Porter Stemming Algorithm (Porter, 1980)
  - http://tartarus.org/~martin/PorterStemmer/index.html
  - http://pypi.python.org/pypi/stemming/1.0
  - Not perfect: “viruses” – “virus”, “virus” – “viru”

```
[“hi”, “darling”, “my”, “photo”, “attached”, “file”]
```

⇓ Porter Stemming Algorithm

```
[“hi”, “darl”, “my”, “photo”, “attach”, “file”]
```

Feature extraction

- Various characteristics, depending on the target task
  - Word n-grams (unigram, bigram, tri-gram, …), prefixes/postfixes
  - Part-of-speech tags, predicate arguments, dependency edges
  - Dictionary matching (e.g., predefined 'black' words in spams)
  - Conditions (e.g., whether the email has a link to a black URL)
  - Others (e.g., the sender of the mail, the IP address of the sender)

- We may use (combinations of) multiple characteristics
  - E.g., unigram and part-of-speech tag: "photo/NN"

```
[“hi”, “darl”, “my”, “photo”, “attach”, “file”]
```

⇓ Word bigrams as features

```
[“hi_darl”, “darl_my”, “my_photo”, “photo_attach”, “attach_file”]
```

Practical considerations

- Bias term
  - Include a feature that is always 1 (without any condition)
  - The corresponding feature weight presents a threshold

$$a'(\boldsymbol{x}) = \sum_{i=1}^{m} w_i x_i + \underbrace{w_0 x_0}_{\text{Always 1}} = s(\boldsymbol{x}) + \underbrace{w_0}_{\text{Bias term}} > 0 \Leftrightarrow s(\boldsymbol{x}) > \underbrace{-w_0}_{\text{Threshold}}$$

- Feature space and number of dimensions (m)
  - We seldom determine the number of dimensions in advance
  - Instead, we do:
    - define an algorithm for extracting features from an instance
    - extract and enumerate features from all instances in the training data
    - $m = (\text{the number of distinct features extracted from the training data})$
- A feature vector extracted from an instance is sparse
  - A few non-zero elements in the m dimension
  - We often use a list of non-zero elements and their values

Example of feature extraction

- Training data (consisting of two instances)
  - +1 Hi darling, my photo in attached file
  - -1 Hi Mark, Kyoto photo in attached file

$y = 0$ is often represented by -1

- Feature representations (word bi-grams)
  - +1 hi_darl darl_my my_photo photo_attach attach_file
  - -1 hi_mark mark_kyoto kyoto_photo photo_attach attach_file
- Feature space
  - 1: 1 (bias), 2: hi_darl, 3: darl_my, 4: my_photo, 5: photo_attach,
  - 6: attach_file, 7: hi_mark, 8: mark_kyoto, 9: kyoto_photo
- Feature vectors
  - $(\boldsymbol{x}, y) = ((1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0), 1)$
  - $(\boldsymbol{x}, y) = ((1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1), 0)$

Training – Perceptron

Finding $\boldsymbol{w}$ such that it fits well to the training data

Training

- We have a training data consisting of N instances:
  - $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$

x_i: the i-th element of the vector $\boldsymbol{x}$
$\boldsymbol{x}_n$: the n-th instance in the training data

- *Generalization* (汎化): if a weight vector $\boldsymbol{w}$ predicts training instances correctly, it will work for unknown instances
  - This is an *assumption*; the weight vector $\boldsymbol{w}$ predicting training instances perfectly does not necessarily perform the best for unknown instances → over-fitting (過学習)

- Find the weight vector $\boldsymbol{w}$ such that it can predicts training instances as correctly as possible
  - Ideally, $\widehat{y_n} = y_n$ for all $n \in [1, N]$ in the training data

Perceptron (Rosenblatt, 1957)

$w_i = 0$ for all $i \in [1, m]$
Repeat:
    $(\boldsymbol{x}_n, y_n) \leftarrow$ Random sample from the training data D
    $\hat{y} \leftarrow \mathrm{predict}(\boldsymbol{w}, \boldsymbol{x}_n)$
    if $\hat{y} \neq y_n$ then:
        if $y_n = 1$ then:
            $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{x}_n$
        else:
            $\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{x}_n$
Until convergence (e.g., until no instance updates $\boldsymbol{w}$)

$$\hat{y} = \begin{cases} 1 & (\text{if } \boldsymbol{w} \cdot \boldsymbol{x}_n > 0) \\ 0 & (\text{otherwise}) \end{cases}$$

How perceptron algorithm works

- Suppose that the current model $\boldsymbol{w}$ misclassifies $(\boldsymbol{x}_n, y_n)$
  - If $y_n = 1$ then:
    - Update the weight vector $\boldsymbol{w}' \leftarrow \boldsymbol{w} + \boldsymbol{x}_n$
    - If we classify $\boldsymbol{x}_n$ again with the updated weights $\boldsymbol{w}'$:
      - $\boldsymbol{w}' \cdot \boldsymbol{x}_n = (\boldsymbol{w} + \boldsymbol{x}_n) \cdot \boldsymbol{x}_n = \boldsymbol{w} \cdot \boldsymbol{x}_n + \boldsymbol{x}_n \cdot \boldsymbol{x}_n \geq \boldsymbol{w} \cdot \boldsymbol{x}_n$
  - If $y_n = 0$ then:
    - Update the weight vector $\boldsymbol{w}' \leftarrow \boldsymbol{w} - \boldsymbol{x}_n$
    - If we classify $\boldsymbol{x}_n$ again with the updated weights $\boldsymbol{w}'$:
      - $\boldsymbol{w}' \cdot \boldsymbol{x}_n = (\boldsymbol{w} - \boldsymbol{x}_n) \cdot \boldsymbol{x}_n = \boldsymbol{w} \cdot \boldsymbol{x}_n - \boldsymbol{x}_n \cdot \boldsymbol{x}_n \leq \boldsymbol{w} \cdot \boldsymbol{x}_n$

Exercise 1: Update $\boldsymbol{w}$ using perceptron

- Training instances:
  - $(\boldsymbol{x}_1, y_1) = ((1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0), 1)$ $\leftarrow$ *Hi darling, my photo in attached file*
  - $(\boldsymbol{x}_2, y_2) = ((1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1), 0)$ $\leftarrow$ *Hi Mark, Kyoto photo in attached file*
- Initialization
  - $\boldsymbol{w} = (0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0)$
- Then?

Answer 1: Update $\boldsymbol{w}$ using perceptron

Python implementation (perceptron.py)

```
def update(w, x, y):
    a = 0.
    for i in range(len(w)):
        a += w[i] * x[i]
    if a * y <= 0:
        for i in range(len(w)):
            w[i] += y * x[i]

def classify(w, x):
    a = 0.
    for i in range(len(w)):
        a += w[i] * x[i]
    return (0. < a)

if __name__ == '__main__':
    w = [0.] * 9
    D = (
        ((1, 1, 1, 1, 1, 1, 0, 0, 0), +1),
        ((1, 0, 0, 0, 1, 1, 1, 1, 1), -1),
    )
    update(w, D[0][0], D[0][1])
    update(w, D[1][0], D[1][1])
    print classify(w, D[0][0])
    print classify(w, D[1][0])
    print w
```

Update rule is very simple if we define negative as -1 (instead of 0)

```
$ python perceptron.py
True
False
[0.0, 1.0, 1.0, 1.0, 0.0, 0.0, -1.0, -1.0, -1.0]
```

Practical considerations

- Perceptron algorithm can learn a training instance at a time (*online training*)
  - Suitable for spam filtering, which constantly receives new instances
- Perceptron algorithm does not converge if the training set is not linearly separable (no discrimination plane exists)
  - We often terminate the algorithm after:
    - A fixed number of iterations (tuned by a development set)
    - Number of misclassification instances does not decrease
- Perceptron algorithm often leads to poor generalization
  - We often use 'averaging' of weight vectors at each iteration for better generalization (Freund and Schapire, 1999)

Training –
Logistic Regression

Section 6.6.2 Logistic Regression (P231-P234)

Logistic regression (ロジスティック回帰)

- A linear binary classifier (the same as perceptron)
  - Input: feature vector $\boldsymbol{x} \in \mathcal{R}^m$
  - Output: prediction $\hat{y} \in \{0,1\}$
  - Using: weight vector $\boldsymbol{w} \in \mathcal{R}^m$

$$\hat{y} = \begin{cases} 1 & (\text{if } a(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} > 0) \\ 0 & (\text{otherwise}) \end{cases}$$

(m: number of dimension)

- In addition, conditional probability $P(y|\boldsymbol{x})$ is defined by,

$$\boxed{\begin{aligned} P(y=1|\boldsymbol{x}) &= \frac{1}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{x}}} \\ P(y=0|\boldsymbol{x}) &= 1 - P(y=1|\boldsymbol{x}) = \frac{e^{-\boldsymbol{w}\cdot\boldsymbol{x}}}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{x}}} \end{aligned}}$$

- Decision rule to classify $\boldsymbol{x}$ to positive ($y = 1$)

$$P(y=1|\boldsymbol{x}) > 0.5 \Leftrightarrow \frac{1}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{x}}} > \frac{1}{2} \Leftrightarrow \boldsymbol{w}\cdot\boldsymbol{x} > 0$$

the same

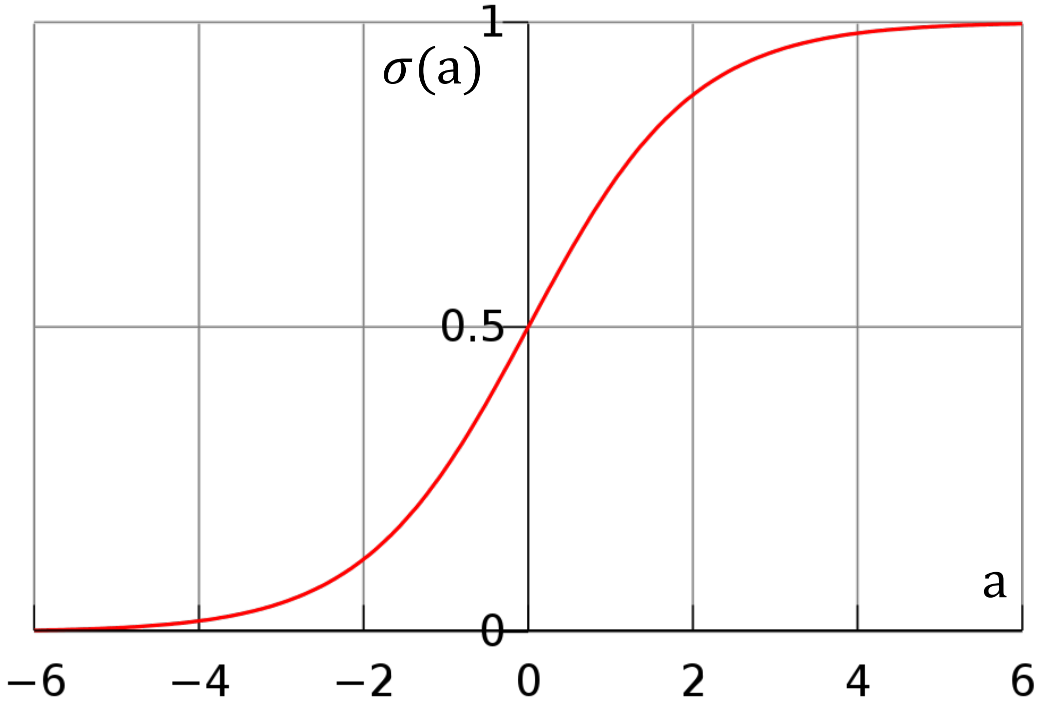
Sigmoid function (シグモイド関数)

- Sigmoid function

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

- This function maps:
  - Score $[-\infty, +\infty]$ to probability $[0, 1]$

- In an implementation, avoid the overflow problem when a is negative



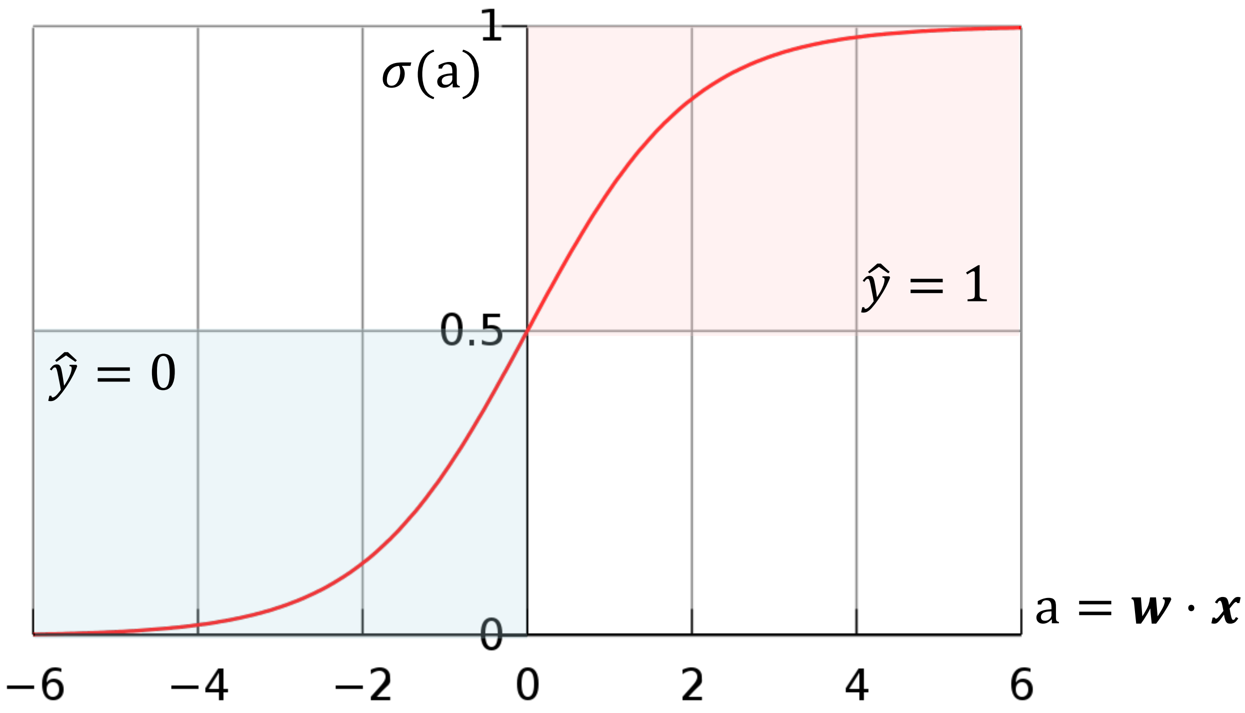
Symmetry point at $(0, 0.5)$

$$\lim_{a \to -\infty} \sigma(a) = 0 \qquad \lim_{a \to \infty} \sigma(a) = 1$$

http://en.wikipedia.org/wiki/Sigmoid_function

Interpreting logistic regression

- Compute the score (inner product) $a = \boldsymbol{w} \cdot \boldsymbol{x}$
- Apply the sigmoid function to map the score a into a probability value, $P(y = 1|\boldsymbol{x}) = \sigma(a) = \sigma(\boldsymbol{w} \cdot \boldsymbol{x})$



(Instance-wise) log-likelihood (対数尤度)

- Given a training instance $(\boldsymbol{x}_n, y_n)$, we compute the instance-wise log-likelihood ℓ_n to assess the fitness,

$$\ell_n \equiv \begin{cases} \log p_n & (\text{if } y_n = 1) \\ \log(1 - p_n) & (\text{if } y_n = 0) \end{cases} = y_n \log p_n + (1 - y_n) \log(1 - p_n),$$

$$p_n \equiv P(y = 1 | \boldsymbol{x}_n) = \frac{1}{1 + e^{-a_n}}, a_n \equiv \boldsymbol{w} \cdot \boldsymbol{x}_n$$

- *Maximum Likelihood Estimation*: we would like to find the weight vector w such that:

$$\text{maximize} \sum_{n=1}^{N} \ell_n$$

Check your understandings with example

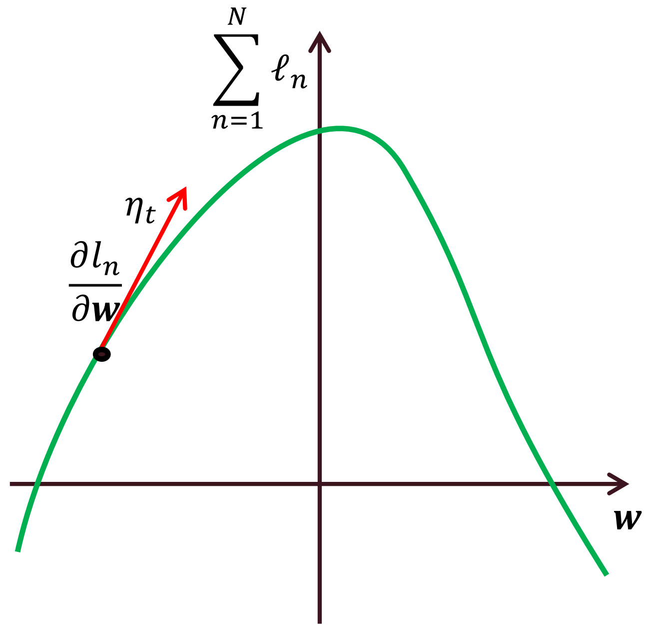
- Model: $\boldsymbol{w} = (0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad -1 \quad -1 \quad -1)$
- $\boldsymbol{x}_1 = (1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0) \leftarrow \textit{Hi darling, my photo in attached file}$
  - $a_1 = \boldsymbol{w} \cdot \boldsymbol{x}_1 = 3, p_1 = \frac{1}{1+\exp(-3)} = 0.953$
  - Suppose that this instance is annotated as positive ($y_1 = 1$)
  - $l_1 = \log p_1 = -0.0486 \longrightarrow 0$ (maximize)

- $\boldsymbol{x}_2 = (1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1) \leftarrow \textit{Hi Mark, Kyoto photo in attached file}$
  - $a_2 = \boldsymbol{w} \cdot \boldsymbol{x}_2 = -3, p_2 = \frac{1}{1+\exp(+3)} = 0.047$
  - Suppose that this instance is annotated as negative ($y_2 = 0$)
  - $l_2 = \log(1 - p_2) = -0.0486 \longrightarrow 0$ (maximize)

Stochastic Gradient ~~Descent~~ Ascent for training

- Optimization problem: *Maximum Likelihood Estimation*
  - maximize $\sum_{n=1}^{N} \ell_n$ (fortunately, this objective is concave)
- *Stochastic Gradient Descent (SGD)* (確率的勾配降下法)
  - Compute the gradient for an instance (batch size = 1): $\frac{\partial \ell_n}{\partial \boldsymbol{w}}$
  - Update parameters to the steepest direction

1. $w_i = 0$ for all $i \in [1, m]$
2. For $t \longleftarrow 1$ to T:
3. $\quad \eta_t \longleftarrow 1/t$
4. $\quad (\boldsymbol{x}_n, y_n) \longleftarrow$ Random sample from D
5. $\quad \boldsymbol{w} \longleftarrow \boldsymbol{w} + \eta_t \frac{\partial l_n}{\partial \boldsymbol{w}}$



Exercise 2: compute the gradient

- Compute the gradients $\frac{\partial \ell_n}{\partial p_n}, \frac{\partial p_n}{\partial a_n}, \frac{\partial a_n}{\partial \boldsymbol{w}}$, and prove:

$$\boxed{\frac{\partial \ell_n}{\partial \boldsymbol{w}} = \frac{\partial \ell_n}{\partial p_n} \frac{\partial p_n}{\partial a_n} \frac{\partial a_n}{\partial \boldsymbol{w}} = (y_n - p_n)\boldsymbol{x}_n}$$

  - where,

$$\ell_n = y_n \log p_n + (1 - y_n) \log(1 - p_n),$$

$$p_n = \frac{1}{1 + e^{-a}},$$

$$a = \boldsymbol{w} \cdot \boldsymbol{x}_n$$

Answer 2: compute the gradients

Interpreting the update formula

- The update formula:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta_t \frac{\partial l_n}{\partial \boldsymbol{w}} = \boldsymbol{w} + \eta_t (y_n - p_n) \boldsymbol{x}_n$$

- If $y_n = p_n$, no need for updating $\boldsymbol{w}$
- If $y_n = 1$ and $p_n < 1$, increase the weight $\boldsymbol{w}$ by the amount of the error $(y_n - p_n)$
- If $y_n = 0$ and $0 < p_n$, decrease the weight $\boldsymbol{w}$ by the amount of the error $(y_n - p_n)$

Python implementation (logress.py)

```
import math
import random

def train(w, D, T):
    for t in range(1, T+1):
        x, y = random.choice(D)
        a = sum([w[i] * x[i] for i in range(len(w))])          # a(x) = w · x
        g = y - (1. / (1. + math.exp(-a))) if -100. < a else y   # g = y − p
        eta = 1. / t
        for i in range(len(w)):                                # w ← w + ηgx
            w[i] += eta * g * x[i]                             #

def prob(x):
    a = sum([w[i] * x[i] for i in range(len(w))])              # a(x) = w · x
    return 1. / (1 + math.exp(-a)) if -100. < a else 0.        # p = 1/(1 + exp(a))

if __name__ == '__main__':
    w = [0.] * 9
    D = (
        ((1, 1, 1, 1, 1, 1, 0, 0, 0), 1),
        ((1, 0, 0, 0, 1, 1, 1, 1, 1), 0),
    )
    train(w, D, 10000)
    print prob(D[0][0]), prob(D[1][0])
    print w
```

```
$ python logpress.py
0.93786130942 0.0601874607952
[-0.0037759046835663321, 0.90852032090533386,
0.90852032090533386, 0.90852032090533386,
-0.0037759046835663321, -0.0037759046835663321,
-0.91229622558889756, -0.91229622558889756,
-0.91229622558889756]
```

Practical considerations

- MLE often leads to overfitting
  - $|\boldsymbol{w}| \to \infty$ as $\sum_{n=1}^{N} \ell_n \to 0$ when the training data is linearly separable
  - Subject to be affected by noises in the training data
- Regularization (正則化) (MAP estimation)
  - We introduce a penalty term when $\boldsymbol{w}$ becomes large
  - E.g., MAP with L2 regularization: $\text{maximize}\ \left(\sum_{n=1}^{N} \ell_n - C|\boldsymbol{w}|^2\right)$
    - C is the parameter to control the trade-off between over/under fitting
- Stopping criterion
  - Detecting the convergence of log-likelihood improvements

Other classification models

- Extensions of logistic regression
  - Maximum Entropy Modeling (MaxEnt) (最大エントロピー法)
    - Multi-class logistic regression
  - Conditional Random Fields (CRFs) (条件付き確率場)
    - Predict a sequence of labels for a given sequence

- Other formalizations
  - Support Vector Machine (SVM) (サポートベクトルマシーン)
    - SVM with linear kernel = linear classifier
  - Naïve Bayes
  - Neural Network
  - Decision Tree
  - …

Evaluation

Accuracy, precision, and recall

Contingency table

	+	-
Predicted +	a (true positive)	b (false positive)
Predicted -	c (false negative)	d (true negative)

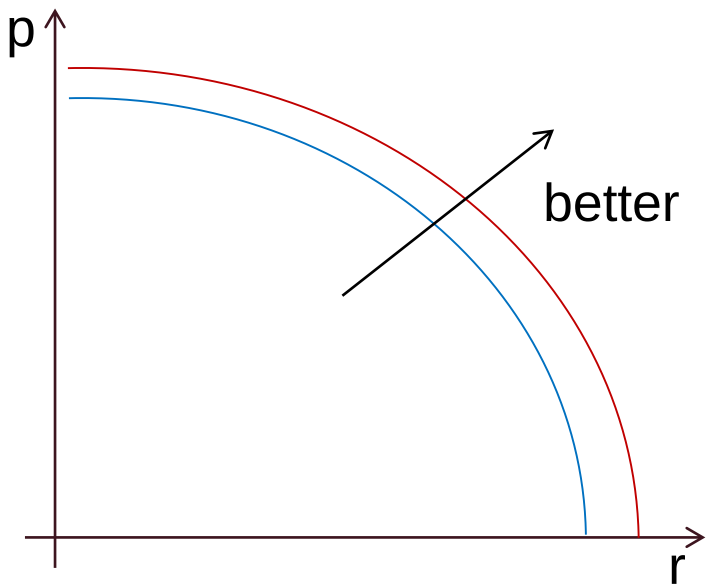
- Accuracy (精度): $\frac{a+d}{a+b+c+d}$ (correctness about positives and negatives)
- Precision (適合率): $p = \frac{a}{a+b}$ (correctness about positives)
- Recall (再現率): $r = \frac{a}{a+c}$ (coverage about positives)
- F1-score: $\frac{2pr}{p+r}$ (harmonic mean of precision and recall)

Precision – recall curve

- Precision/recall tradeoff
  - If we increase the precision of a system, its recall decreases

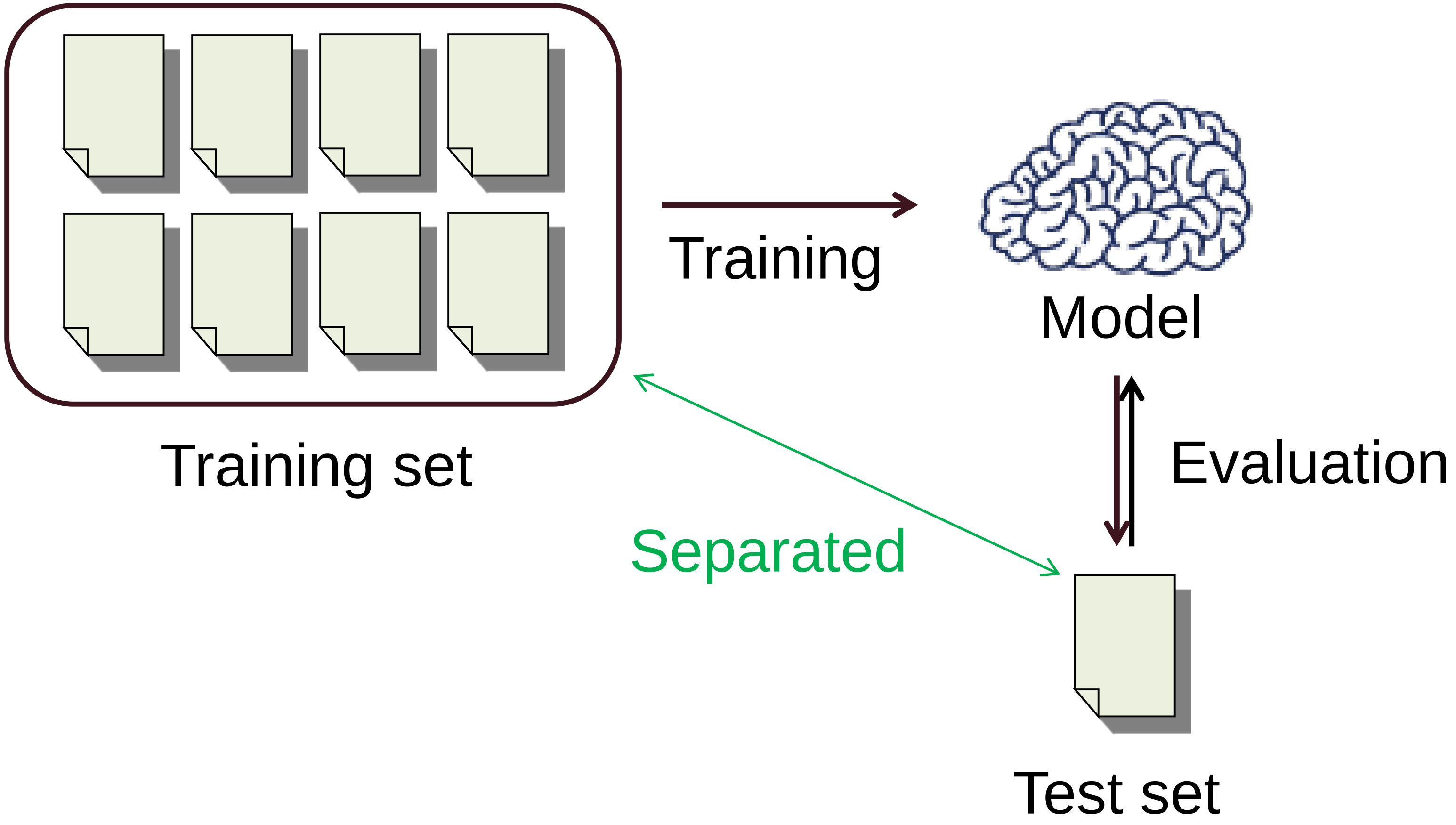
- It is not easy to improve both precision and recall
  - We sometimes prioritize either precision or recall, depending on the task
  - Spam filtering: precision is important

- How can we control the tradeoff of a linear binary classifier?
  - Increasing the threshold: improves precision
  - Decreasing the threshold: improves recall



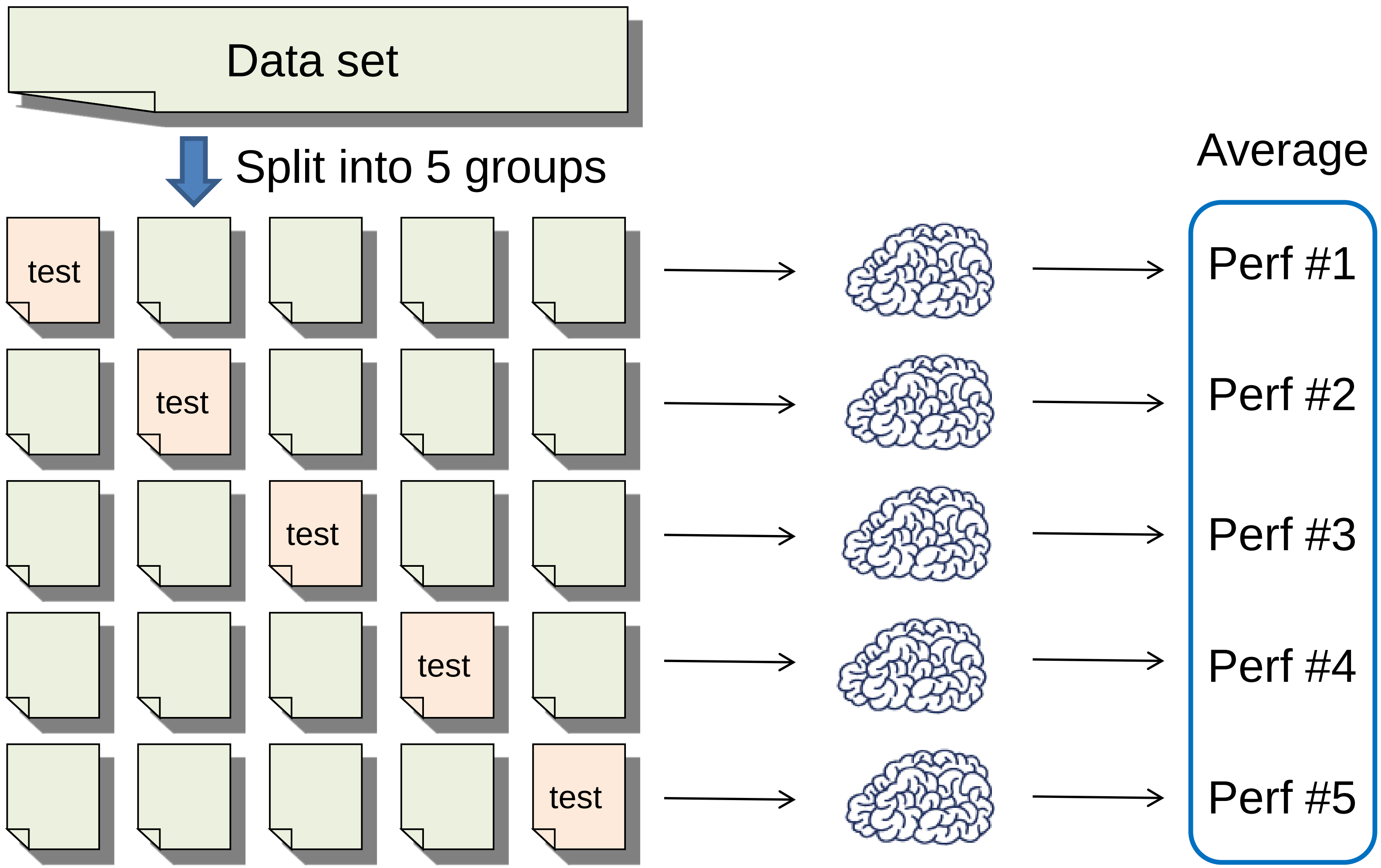
Holdout evaluation

- Use ‘heldout’ data set for evaluation



N-fold cross validation (交差検定)

- For example (5-fold cross validation)



Standard experiment procedure

- Three groups of data sets
  - *Training set*: used for training
  - *Test set*: used for evaluating the trained model
  - *Development set*: another test set for tuning parameters of training
  - This experimental setting is useful for comparing different systems

- Two groups of data sets
  - *Training set*: used for N-fold cross validation
  - *Development set*: a test set for tuning parameters of training