Text classification

IN4080
Natural Language Processing

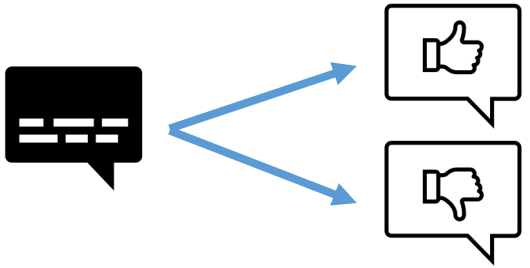
Yves Scherrer

Main NLP tasks

Natural language generation



Annotation (natural language understanding)



These are (in general) **text classification** or **document classification** tasks:
one label per text / document.

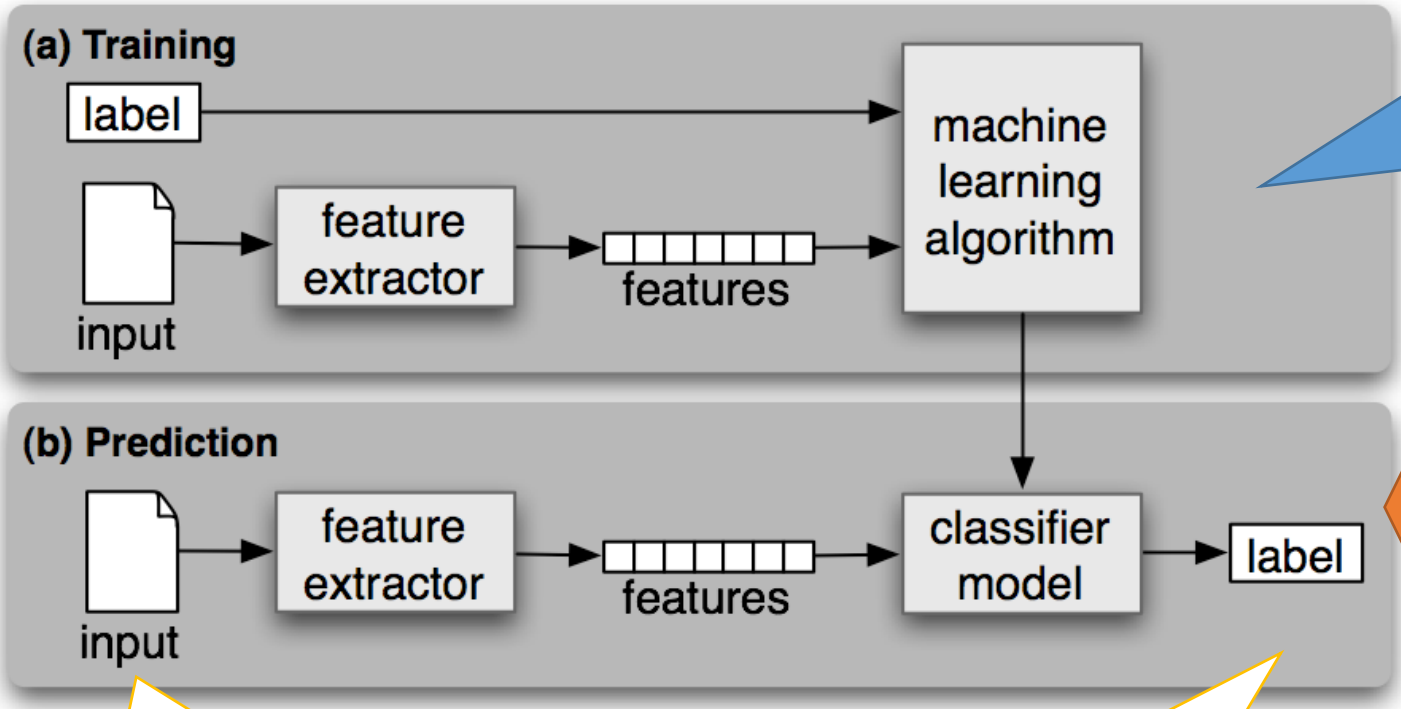
- Hate speech detection
- Sentiment analysis
- Language identification
- Syntactic analysis
- Part-of-speech tagging

A sample of a sentiment analysis dataset

Text / Document	Label
Shanghai is also really exciting (precisely -- skyscrapers galore). Good tweeps in China:	positive
Recession hit Veronique Branquinho, she has to quit her company, such a shame!	negative
happy bday!	positive
that`s great!! weee!! visitors!	positive
I THINK EVERYONE HATES ME ON HERE lol	negative
soooooo wish i could, but im in school and myspace is completely blocked	negative
and within a short time of the last clue all of them	neutral
What did you get? My day is alright.. Haven't done anything yet. leaving soon to my stepsister tho...	neutral
My bike was put on hold...should have known that.... argh total bummer	negative
I checked. We didn't win	neutral
.. and you're on twitter! Did the tavern bore you that much?	neutral
Its coming out the socket I feel like my phones hole is not a virgin. That's how loose it is... :'(	negative

https://www.kaggle.com/datasets/abhi8923shriv/sentiment-analysis-dataset

Supervised classification



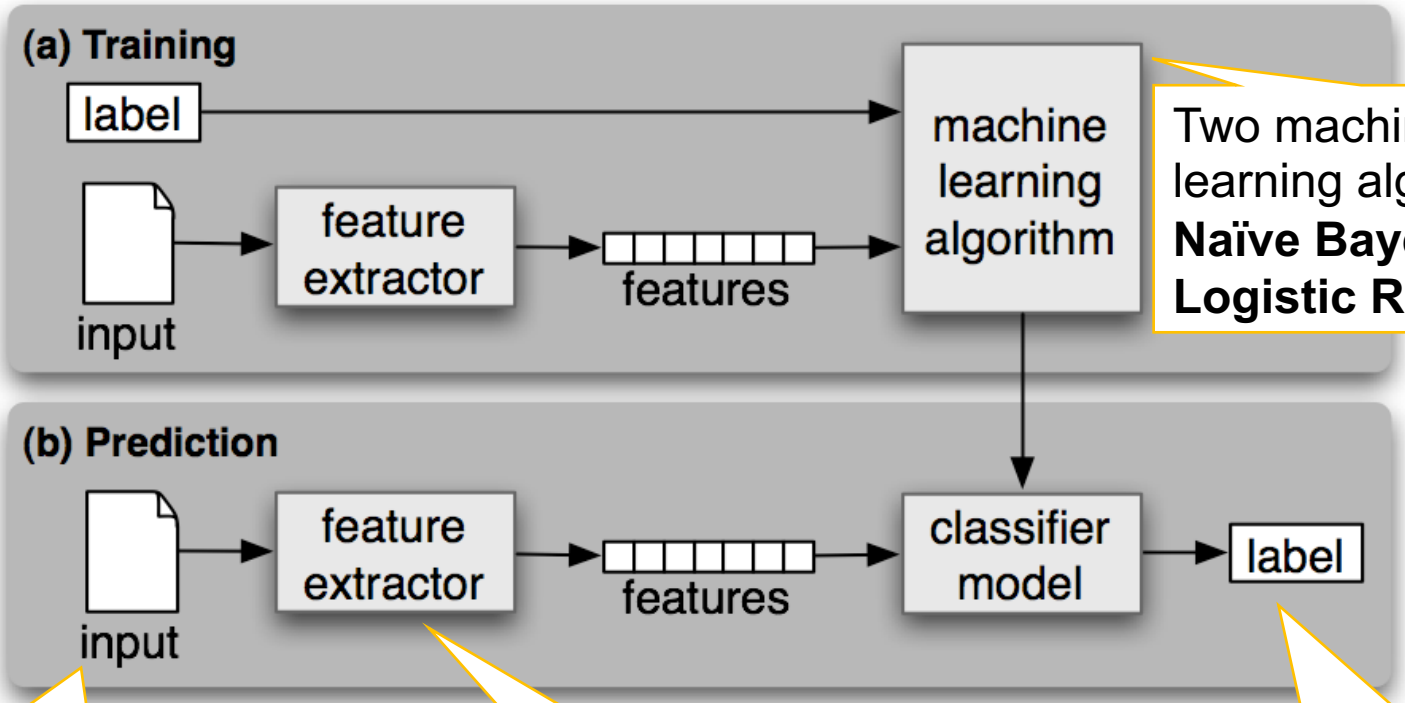
Training phase: both input text and label are given

Test / prediction phase: only input text is given, the model predicts the label

x, input, instance, observation, document

y, label, output, class

Supervised classification



Two machine learning algorithms: **Naïve Bayes** and **Logistic Regression**

How does the data look like, and how do we split it between the training and prediction phases?

One approach to feature extraction: **bag of words**

How can we evaluate and compare classifiers?

Task definition

Input:

- An instance x from dataset X
  - Also called document d
- A fixed set of labels Y
  - Also called classes C

Output:

- A predicted label $\hat{y} \in Y$
  - A predicted class $\hat{c} \in C$

Supervised classification

Take an instance $x \in X$ and associate it with a label $y \in Y$.

- General prediction function:

$$\hat{y} = \arg\max_{y \in Y} S(x, y)$$

  - $\hat{y}$: predicted class according to a model
  - S is some scoring function and depends on the type of classification algorithm

- Prediction function for probabilistic models:

$$\hat{y} = \arg\max_{y \in Y} P(y|x)$$

Naive Bayes

Bayesian inference

- Prediction: $\hat{y} = \arg\max_{y \in Y} P(y|x)$
- Bayes' theorem: $P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$
- Thus: $\hat{y} = \arg\max_{y \in Y} \frac{P(x|y) \cdot P(y)}{P(x)}$
- $P(x)$ does not affect the argmax computation
- Thus: $\hat{y} = \arg\max_{y \in Y} (P(x|y) \cdot P(y))$

Bayesian inference

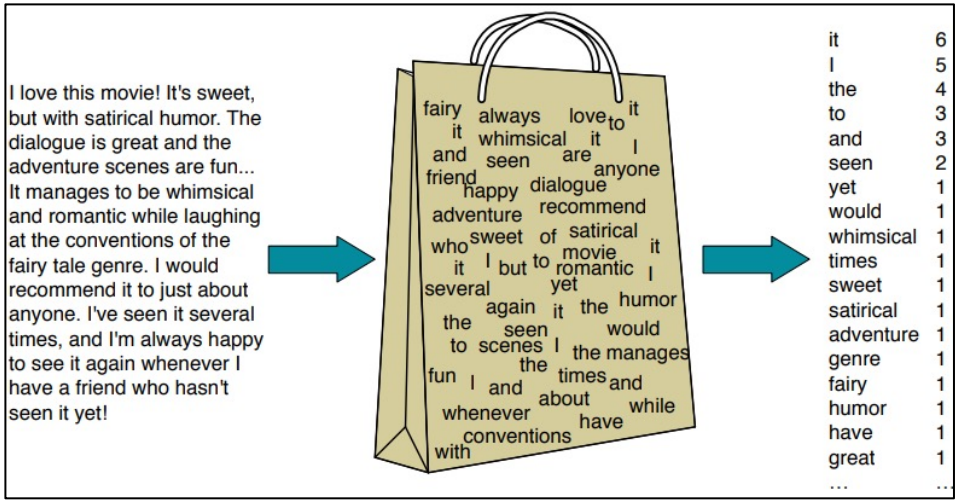
$$\hat{y} = \arg\max_{y \in Y} (P(x|y) \cdot P(y))$$

likelihood prior probability

- The likelihood part feels backwards: we compute the probability of a document given a label, not the probability of a label given a document!

Bag of words

- In the NB prediction function, x represents a document, consisting of dozens to hundreds of words: $\hat{y} = \arg\max_{y \in Y}(P(x|y) \cdot P(y))$
- We need to decompose the document.
- The **bag of words representation** reduces a document to the frequency distribution of its words.



The Naive Bayes assumption

Bag of words representation:

$$P(x|y) = P(x_1, x_2, \ldots, x_n|y)$$

- Each x_i corresponds to a word.
- Simplifying assumption: a word is equally likely in all positions.

Naïve Bayes assumption:

- All x_i are conditionally independent given the class.
- Their probabilities can thus be 'naively' multiplied:

$$P(x_1, x_2, \ldots, x_n|y) = P(x_1|y) \cdot P(x_2|y) \cdot \cdots \cdot P(x_n|y)$$

- How realistic is this assumption?

The Naive Bayes assumption

Full prediction function:

$$\hat{y} = \arg\max_{y \in Y} \left(P(y) \cdot \prod_{i=1}^{n} P(x_i | y) \right)$$

Next steps:

- An example with fictive probabilities
- How to train the probability distributions from data
- How to evaluate text classification models

An example

Likelihoods $P(x_i|y)$:

	positive	negative
I	0.09	0.15
always	0.07	0.08
like	0.29	0.06
foreign	0.06	0.15
films	0.08	0.12

Priors $P(y)$:

positive	negative
0.5	0.5

$$\hat{y} = \arg\max_{y \in Y} \left(P(y) \cdot \prod_{i=1}^{n} P(x_i|y) \right)$$

- What is the prediction score of the sentence "I always like foreign films" for class "positive"?
- What is the score for class "negative"?

Experimental setup

- We need **training data** to fill the likelihoods and priors tables
  - Labeled instances
  - The model learns to associate instances with labels
- We need **test data** to apply the argmax function and to evaluate the quality of the predictions
  - In most cases: labeled instances
  - Hide the labels, let the model predict new labels, and compare the two to compute performance

Training

Training a Naive Bayes model amounts to estimating the probabilities $\hat{P}(y)$ and $\hat{P}(x_i|y)$

Maximum likelihood estimation: maximize the probability of the training set (= compute relative counts)

- Learning the priors: $$\hat{P}(y) = \frac{N(y)}{\sum_{y' \in Y} N(y')}$$

$N(y)$ = number of documents in class y

- Learning the likelihoods: $\hat{P}(x_i|y) = \frac{\text{Count}(x_i, y)}{\sum_{x' \in V} \text{Count}(x', y)}$

V = union of all words in all classes

An example

Consider the following movie reviews associated with genres “comedy” or “action”:

fun couple love love	comedy
fast furious shoot	action
couple fly fast fun fun	comedy
furious shoot shoot fun	action
fly fast shoot love	action

- What is the value of $P(y = \text{comedy})$?
- What is the value of $P(x = \text{fast} \mid y = \text{comedy})$?
- What is the value of $P(x = \text{fast} \mid y = \text{action})$?
- Which is the predicted class for “fast couple shoot fly”?

Smoothing

It is generally not the case that all words $x_1, x_2, \ldots, x_n$ are observed in all classes

- When they are not, $P(x_i|y) = 0$
- By multiplication, the entire $P(x|y) = 0$

Solution: Apply (additive) smoothing like for language modeling

$$\hat{P}(x_i|y) = \frac{\text{Count}(x_i, y) + \alpha}{\sum_{x' \in V} (\text{Count}(x', y) + \alpha)}$$

- Generally, smoothing is only necessary for the likelihoods, but it can be applied to the priors too

An example

Let us apply **add-one smoothing** to the likelihoods.

fun couple love love	comedy
fast furious shoot	action
couple fly fast fun fun	comedy
furious shoot shoot fun	action
fly fast shoot love	action

- What is the value of $P(y = \text{comedy})$?
- What is the value of $P(x = \text{fast} \mid y = \text{comedy})$?
- What is the value of $P(x = \text{fast} \mid y = \text{action})$?
- Which is the predicted class for “fast couple shoot fly”?

Prediction

We have learned the $P(x_i|y)$ for all x_i and y.

We have learned the $P(y)$ for all y.

Prediction: For each x, compute

$$\hat{y} = \arg\max_{y \in Y} \left(P(y) \cdot \prod_{i=1}^{n} P(x_i|y) \right)$$

- Prediction scores are probabilities ($[0..1]$) and typically become very small.
- To prevent this, one typically converts probabilities to log probabilities.

Computing in log space

What do logarithms do?

- Logarithms transform big positive numbers (> 1) into smaller positive numbers
- Logarithms transform numbers between 0 and 1 into big negative numbers
  - $\log(0.0005) = -7.60$
  - $\log(0.0004) = -7.82$
- The order is preserved: if $a > b$ then $\log(a) > \log(b)$
- The log of a product is the sum of the logs of its factors

Computing in log space

Prediction function:

$$\hat{y} = \arg\max_{y \in Y} \left(P(y) \cdot \prod_{i=1}^{n} P(x_i|y) \right)$$

In log space:

$$\hat{y} = \arg\max_{y \in Y} \left(\log P(y) + \sum_{i=1}^{n} \log P(x_i|y) \right)$$

- Avoids underflow and increases speed
- You may assume that all logarithms are base e, but any other base is fine (if used consistently)

Algorithm: Training

```
procedure train_naive_bayes (D, alpha):

    for each document x with label y in D:
        doc_counts[y] += 1
        doc_counts_total += 1

        for each word xi in x:
            feat_counts[xi, y] += count(xi)
            feat_counts_total[xi] += count(xi)
        end for
    end for

    for each y in doc_counts:
        logpriors[y] = log(doc_counts[y] / doc_counts_total)
    end for

    for each xi, y in feat_counts:
        loglikelihoods[xi, y] = log( (feat_counts[xi, y] + alpha) /
                                     (feat_counts_total[xi] + (len(feat_counts_total) * alpha)) )
    end for

return logpriors, loglikelihoods
```

This procedure uses simple dictionaries instead of vectors/arrays. The latter may be more efficient.

Algorithm: Testing

```
procedure test_naive_bayes (x, logpriors, loglikelihoods):
                                          # x is one test document
    for each class y in logpriors:
        logsum[y] = logpriors[y]

        for each word xi in x:
            if xi, y in loglikelihoods:  # skip words not seen in training
                logsum[y] += loglikelihoods[xi,y]
            end if
        end for
    end for

    y_hat = argmax_y(logsum[y])

return y_hat
```

Variants

Multinomial Naive Bayes uses count-based bags of words (or weighted count-based BOWs)

- Multinomial distribution = probability distribution over vectors of non-negative counts

Binary Naive Bayes uses binary counts

- 1 if word is present, 0 if word is absent

Four original documents:

− it was pathetic the worst part was the boxing scenes
− no plot twists or great scenes
+ and satire and great plot twists
+ great scenes great film

After per-document binarization:

− it was pathetic the worst part boxing scenes
− no plot twists or great scenes
+ and satire great plot twists
+ great scenes film

	NB Counts		Binary Counts	
	+	−	+	−
and	2	0	1	0
boxing	0	1	0	1
film	1	0	1	0
great	3	1	2	1
it	0	1	0	1
no	0	1	0	1
or	0	1	0	1
part	0	1	0	1
pathetic	0	1	0	1
plot	1	1	1	1
satire	1	0	1	0
scenes	1	2	1	2
the	0	2	0	1
twists	1	1	1	1
was	0	2	0	1
worst	0	1	0	1

Figure 4.3 An example of binarization for the binary naive Bayes algorithm.

Properties of Naïve Bayes

- A probabilistic classifier
  - There are also non-probabilistic ones
- A multi-class classifier
  - Can handle more than two classes
  - This is the default case for NLP problems
- Training is deterministic:
  - Each training instance is seen exactly once
  - The order in which the training instances are seen does not matter
  - Probabilities can be computed exactly (“closed form”), there is no random element in the computation

Properties of Naïve Bayes

- The independence assumption is in general wrong!
  - $P(x_1, x_2, \ldots, x_n|y)$ is far from $P(x_1|y) \cdot P(x_2|y) \cdot \cdots \cdot P(x_n|y)$
- Nevertheless, NB works reasonably well as a classifier:
  - Fast to train, low storage requirements
  - Not prone to overfitting
  - Irrelevant features just cancel each other out
  - Good in domains with many equally important features

Evaluation measures

Automatic evaluation

- Use a test set that is already labeled
- Hide the existing labels, let the model predict new labels, and compare the two
- For each test instance, we have a **gold label** and a **predicted label**

Accuracy

Accuracy simply refers to the proportion of matching labels (gold label = predicted label):

$$\frac{\text{number of instances with matching labels}}{\text{total number of instances}}$$

Accuracy

- Example: A model that predicts whether a given message is spam or not.
- 1685 test instances, distributed as follows:

	Gold label YES	Gold label NO
System prediction YES	83	22
System prediction NO	41	1539

- Accuracy: $\frac{83+1539}{83+22+41+1539} = \frac{1622}{1685} = 96.26\%$

- What would be the accuracy of a stupid model that always returns NO?

Contingency tables

- This is a **contingency table** (or **confusion matrix**):

	Gold label YES	Gold label NO
System prediction YES	true positive	false positive
System prediction NO	false negative	true negative

- These names are indeed a bit confusing…
  - positive = obtained a YES prediction
  - negative = obtained a NO prediction
  - true = prediction is true (predicted = gold label)
  - false = prediction is false (predicted ≠ gold label)

Precision and recall

	Gold label YES	Gold label NO
System prediction YES	true positive	false positive
System prediction NO	false negative	true negative

- $\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$
- $\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$
- Intuition: We want our model to report "all the truth and nothing but the truth"

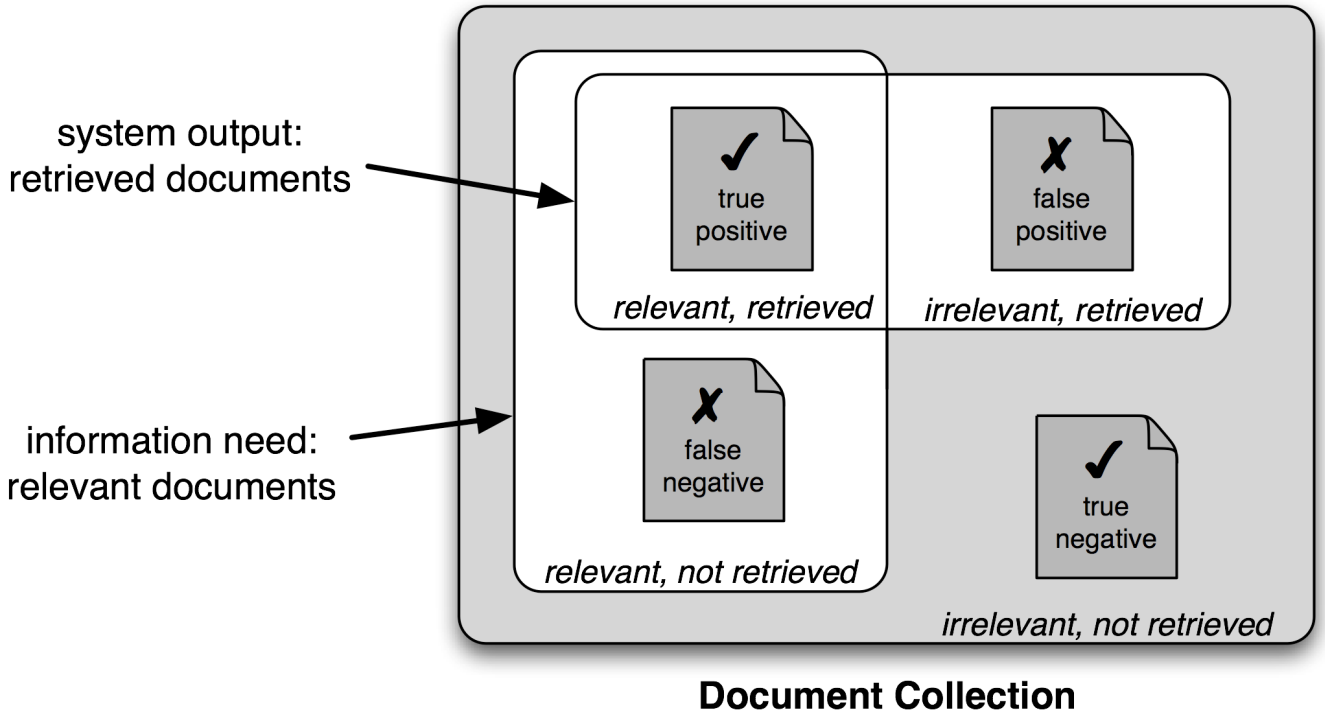
recall

precision

Intuition: Information retrieval

- Goal: Find all the documents on a particular topic out of 100 000 documents
  - Say there are 5
  - The system delivers 10 documents: 2 relevant and 8 irrelevant
  - What is the accuracy?
- For information retrieval tasks, we focus on:
  - The relevant documents
  - The documents returned by the system
- We do not care about:
  - The irrelevant documents which are not returned

Intuition: Information retrieval



Precision and recall

What about our two spam classification models?

- Intelligent:
  - Precision: $\frac{83}{83+22} = 79.0\%$
  - Recall: $\frac{83}{83+41} = 66.9\%$
- Stupid:
  - Precision: $\frac{0}{0+0} = 0\%$ (undefined $\rightarrow$ 0)
  - Recall: $\frac{0}{0+124} = 0\%$

F-measure

- It is a bit inconvenient to have to report two figures, precision and recall...
- Can we combine them into one, take some "average"?
- **F-measure** is the harmonic mean of precision and recall
- **F_1-measure** is the (most popular) variant that weighs precision and recall equally:

$$F_1 = \frac{2 * P * R}{P + R}$$

Precision and recall

- Precision and recall are usually defined on the YES class, but they can also be defined on any other class

	Gold label YES	Gold label NO
System prediction YES	83	22
System prediction NO	41	1539

- Precision of NO: $\frac{1539}{41+1539}$
- Recall of NO: $\frac{1539}{22+1539}$

Multi-class settings

system output \ gold labels	urgent	normal	spam	
urgent	8	10	1	$\text{precision}_u = \frac{8}{8+10+1}$
normal	5	60	50	$\text{precision}_n = \frac{60}{5+60+50}$
spam	3	30	200	$\text{precision}_s = \frac{200}{3+30+200}$
	$\text{recall}_u = \frac{8}{8+5+3}$	$\text{recall}_n = \frac{60}{10+60+30}$	$\text{recall}_s = \frac{200}{1+50+200}$	

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$\text{precision} = \frac{8}{8+11} = .42$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$\text{precision} = \frac{60}{60+55} = .52$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$\text{precision} = \frac{200}{200+33} = .86$

$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = \mathbf{.60}$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$\text{microaverage precision} = \frac{268}{268+99} = \mathbf{.73}$

Readings

- Jurafsky & Martin 3rd ed., chapter 4

Logistic regression

Supervised classification

Take an instance $x \in X$ and associate it with a label $y \in Y$.

- General prediction function:

$$\hat{y} = \arg\max_{y \in Y} S(x, y)$$

  - $\hat{y}$: predicted class according to a model
  - S is some scoring function and depends on the type of classification algorithm

- Prediction function for probabilistic models:

$$\hat{y} = \arg\max_{y \in Y} P(y|x)$$

Can we estimate this probability directly, without applying Bayes' theorem?

Generative vs discriminative models

Generative models

- Ask about $P(x|y)$ or $P(x,y)$
- "What is the likelihood of document x if we know that its label is y?" and "What is the overall probability of label y?"

$$\hat{y} = \arg\max_{y \in Y} (\underbrace{P(y) \cdot P(x|y)}_{P(x,y)})$$

Discriminative models

- Ask about $P(y|x)$ directly
- "What is the likelihood that we get label y if we know that the document is x?"

$$\hat{y} = \arg\max_{y \in Y} P(y|x)$$

Logistic regression

Agenda:

- Describe the instances as feature vectors
- Define the prediction function
- Define the training process