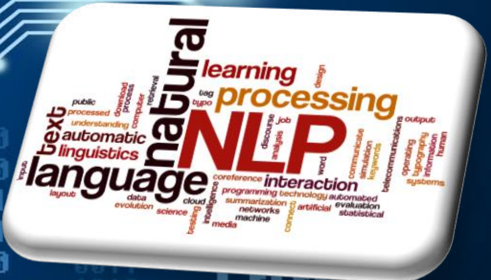
Cairo University
Faculty of Engineering
Computer Engineering Department

Cairo University

Natural Language Processing

Dr. Sandra Wahid



Text Categorization/Classification

Text Categorization: is the task of assigning a label or category to an entire text or document.

- **Sentiment Analysis**: is one common text categorization task, concerned with the extraction of sentiment, the positive or negative orientation that a writer expresses toward some object.
  - Examples: a review of a movie, book, or product on the web expresses the author's sentiment toward the product, while an editorial or political text expresses sentiment toward a candidate or political action.
  - Extracting consumer or public sentiment is thus relevant for fields from marketing to politics.

- **Spam detection**: is the binary classification task of assigning an email to one of the two classes *spam* or *not-spam*.

- **Language id**: is the task concerned with knowing the language a certain text is written in.

- **Authorship attribution**: is a task related to text classification tasks concerned with determining a text's author.

- **Subject category classification**: is the task concerned with assigning a library subject category or topic label to a text.

Text Categorization/Classification

- Classification is essential for tasks below the level of the document as well.
  - Period disambiguation: deciding if a period is the end of a sentence or part of a word.
  - Word tokenization: deciding if a character should be a word boundary.
  - Language modeling: can be viewed as classification, each word can be thought of as a class, and so predicting the next word is classifying the context-so-far into a class for each next word.
  - A part-of-speech tagger: classifies each occurrence of a word in a sentence as, e.g., a noun or a verb.

- The goal of classification is to take a single observation, extract some useful features, and thereby classify the observation into one of a set of discrete classes.

ba5ud word, atl3 menha m3lomat, w b3den a7aded hya
arbotha b class mn el classes elly 3andy

Classifiers

- **Supervised classification**: we have a training set of *N* documents that have each been hand-labeled with a class: *(d1,c1),…, (dN,cN)*. Our goal is to learn a classifier that is capable of mapping from a new document *d* to its correct class *c*ϵ*C*.

- **Probabilistic classifier**: additionally will tell us the probability of the observation being in the class. This full distribution over the classes can be useful information for downstream decisions; avoiding making discrete decisions early which can be useful when combining systems.

lama bnst5dm el probability, bn5aly el model ya5ud el model bta3o mt25r, fa ye2dr ye3ml processing l more details, 3ks el discrete, momken el model yegy 3nd layer fl nos masln, w tla2y eno 5las hwa 3aml el prediction, fa ay propagation odam fl layers, msh htfr2 m3ak f 7aga, hwa keda keda khad kraro 5las.

Classifiers

- Many kinds of machine learning algorithms are used to build classifiers:
    - **Generative classifiers**: build a model of how a class could generate some input data. Given an observation, they return the class most likely to have generated the observation, e.g. Naïve Bayes.
    - **Discriminative classifiers**: learn what features from the input are most useful to discriminate between the different possible classes, e.g. Logistic Regression.

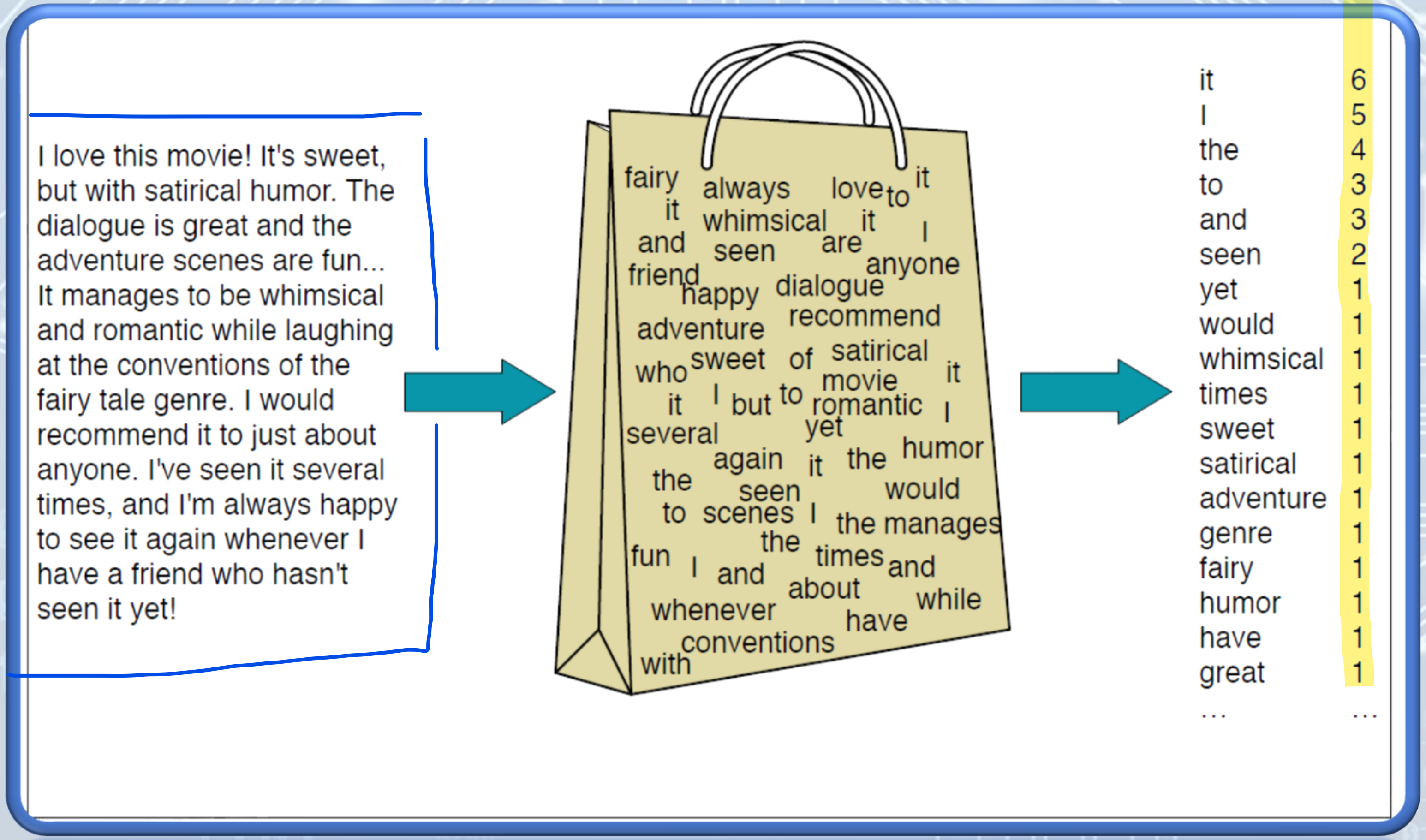
Imagine we're trying to distinguish dog images from cat images:

- A generative model would have the goal of understanding what dogs look like and what cats look like.
- A discriminative model, by contrast, is only trying to learn to distinguish the classes (perhaps without learning much about them). So maybe all the dogs in the training data are wearing collars and the cats aren't.
- A generative model like naive Bayes makes use of the likelihood term ***P(d|c)***, which expresses how to generate the features of a document if we knew it was of class c.
- By contrast a discriminative model in this text categorization scenario attempts to directly compute ***P(c|d)***. It will learn to assign a high weight to document features that directly improve its ability to discriminate between possible classes, even if it couldn't generate an example of one of the classes.

- While discriminative systems are often more accurate and hence more commonly used, generative classifiers still have a role.

Naive Bayes Classifiers

- We will introduce the ***multinomial*** *naive Bayes* classifier.
- We represent a text document as if it were a **bag-of-words**, that is, an ***unordered*** set of words with their ***position ignored***, keeping only their ***frequency*** in the document.



I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

e7na 3auzen ne3ml a hena aslun?

a probability bta3t en el document de tkon related le class mo3ayn given el document.

example: sentiment analysis, m3ak post, w 3auz te3rf hwa eh el probability en el post da ykon el classification bta3to eno happy post masln.

Naive Bayes Classifiers

- Naive Bayes is a **probabilistic classifier**, meaning that for a document *d*, out of all classes *c∈C*, the classifier returns the class $\hat{c}$ which has the **maximum posterior probability** given the document.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d)$$

max index

vec

- Uses Bayes' rule:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

- Substituting:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)}$$

- Simplifying:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \overbrace{P(d|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

This is possible because we will be computing $\frac{P(d|c)P(c)}{P(d)}$ for each possible class. But *P(d)* doesn't change for each class →we are always asking about the most likely class for the same document *d*, which must have the same probability *P(d)*.

Naive Bayes Classifiers

- We can represent a document d as a set of features *f1, f2, …,fn*:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \overbrace{P(f_1, f_2, ..., f_n|c)}^{\text{likelihood}} \; \overbrace{P(c)}^{\text{prior}}$$

- This is still **too hard to compute directly**: without some simplifying assumptions, estimating the probability of every possible combination of features (for example, every possible set of words and positions) would require huge numbers of parameters and impossibly large training sets.

- Naive Bayes classifiers therefore make two simplifying assumptions:
  - **bag of words**: we assume position doesn't matter, and that the word "happy" has the same effect on classification whether it occurs as the 1st, 30th, or last word in the document. Thus, we assume that the features *f1, f2, …, fn* only **encode word identity** and not position.
  - **naive Bayes assumption**: this is the conditional independence assumption that the probabilities *P(fi|c)* are independent given the class *c* and hence can be '*naively*' multiplied as follows:

$$P(f_1, f_2, ..., f_n|c) \; = \; P(f_1|c) \cdot P(f_2|c) \cdot ... \cdot P(f_n|c)$$

Naive Bayes Classifiers

- The final equation for the class chosen by a naive Bayes classifier is thus:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c)$$

es2l el doctor 3la 7war el word positions da.

- To apply the naive Bayes classifier to text, we need to consider word positions, by simply walking an index through every word position in the document:

$$\text{positions} \leftarrow \text{all } \underbrace{\text{word positions}} \text{ in test document}$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i \in positions} P(w_i|c)$$

- Naive Bayes calculations are done in **log space**, to avoid underflow and increase speed.

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

linear 34an el features bta3tna ba2t hya el log fa ehna hena bn3ml summation, fa keda el space bta3na baa linear.

By considering features in log space: the predicted class is computed as a ***linear function of input features***. Naive Bayes and also Logistic regression are **linear classifiers**.

Training the Naive Bayes Classifier

- For the class prior *P(c),* we ask what percentage of the documents in our training set are in each class c.

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

***N*$_c$**: the number of documents in our training data with class *c*
***N*$_{doc}$**: the total number of documents

- To learn the probability *P(fi|c),* we'll assume a feature is just the existence of a word in the document's bag of words, and so *P(wi|c)* is computed as the fraction of times the word *wi* appears among all words in all documents of topic *c*.

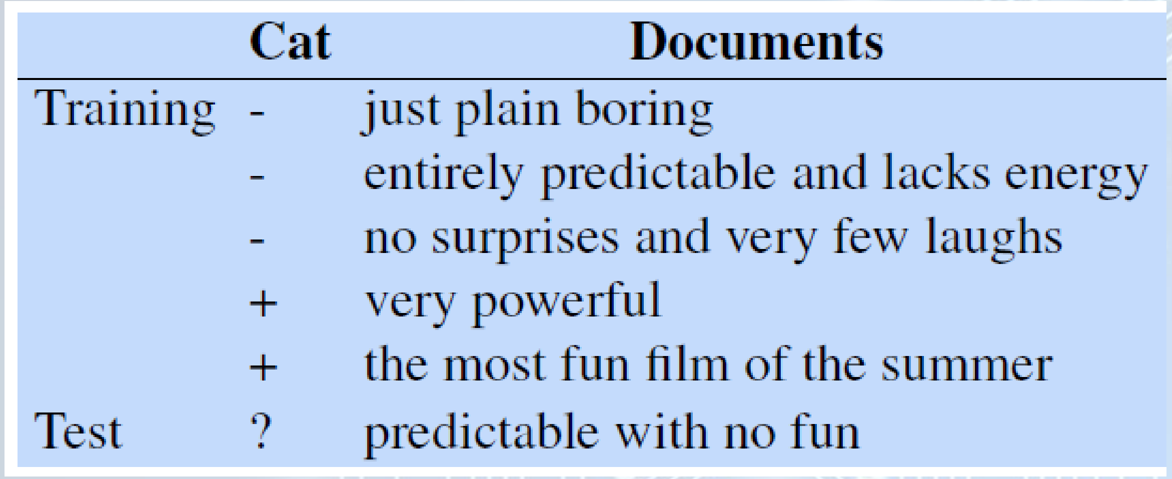
$$\hat{P}(w_i|c) = \frac{count(w_i,c)}{\sum_{w\in V} count(w,c)}$$

- Laplace smoothing is commonly used in naive Bayes text categorization:

$$\hat{P}(w_i|c) = \frac{count(w_i,c)+1}{\sum_{w\in V}(count(w,c)+1)} = \frac{count(w_i,c)+1}{\left(\sum_{w\in V} count(w,c)\right)+|V|}$$

V is the total number of unique words in all the documents of the training set

Naive Bayes Classifier Example



	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

$$P(-) = \frac{3}{5} \qquad P(+) = \frac{2}{5}$$

$$P(\text{“predictable”}|-) = \frac{1+1}{14+20} \qquad P(\text{“predictable”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“no”}|-) = \frac{1+1}{14+20} \qquad P(\text{“no”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“fun”}|-) = \frac{0+1}{14+20} \qquad P(\text{“fun”}|+) = \frac{1+1}{9+20}$$

- For the test sentence S = “predictable with no fun”, after removing the unknown word ‘with’, the chosen class is therefore computed as follows:

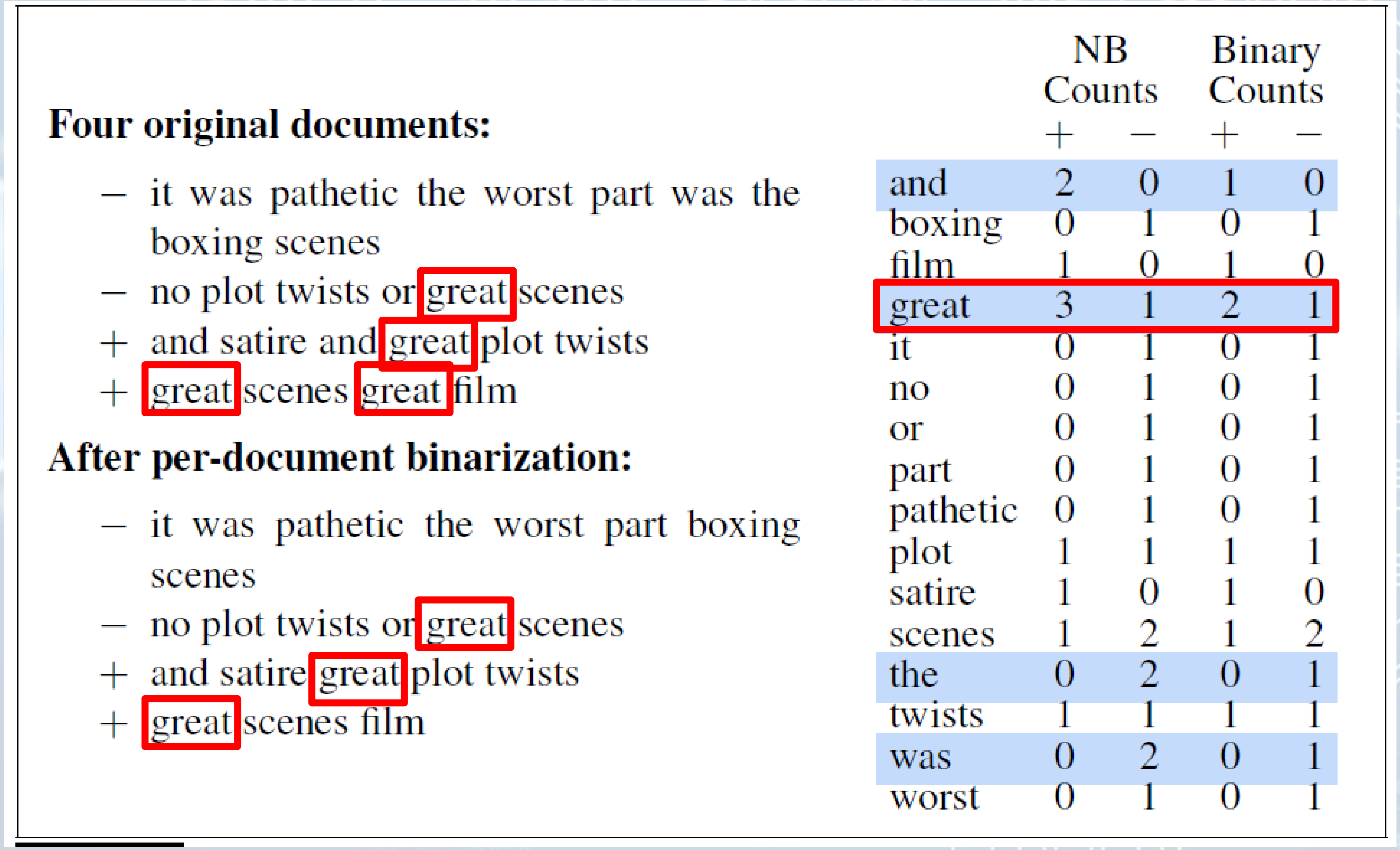
$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$

$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

The model thus predicts the class negative for the test sentence

Optimizing for Sentiment Analysis

Binary Naive Bayes: for sentiment classification whether a word *occurs or not seems to matter more than its frequency*. Thus, it often improves performance to clip the word counts in each document at 1.



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

Optimizing for Sentiment Analysis

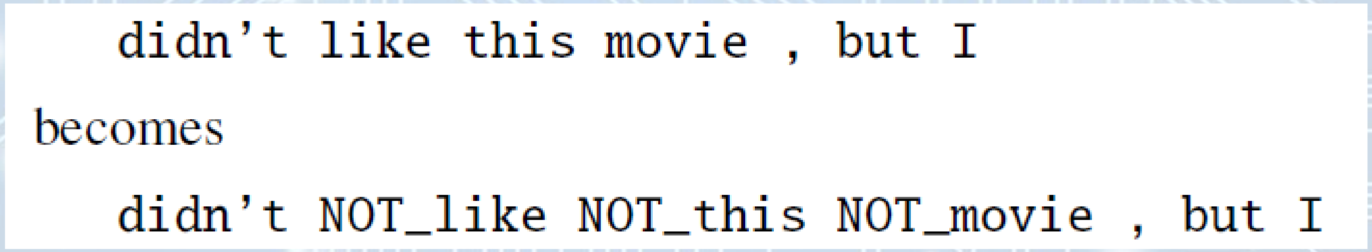
Dealing with Negation:

- Consider the difference between *I really like this movie (positive)* and *I didn't like this movie (negative).*

  → The negation expressed alters the inferences we draw from the predicate *like*.

- Similarly, negation can modify a negative word to produce a positive review (*don't dismiss this film, doesn't let us get bored*).

- During text normalization, prepend the prefix **NOT** to every word after a token of logical negation *(n't, not, no, never)* until the next punctuation mark.

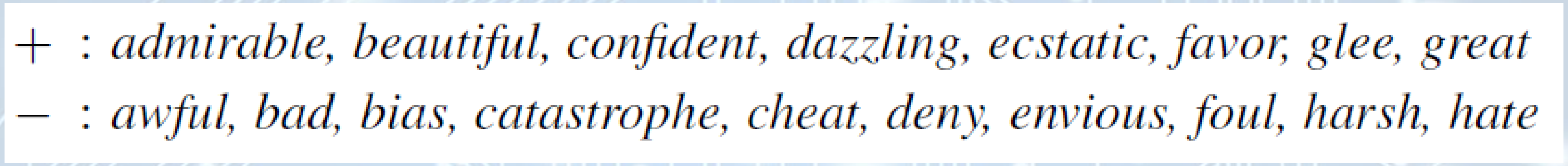


```
    didn't like this movie , but I
becomes
    didn't NOT_like NOT_this NOT_movie , but I
```

- Newly formed 'words' like *NOT like*, *NOT recommend* will thus occur more often in negative document and act as cues for negative sentiment, while words like *NOT bored*, *NOT dismiss* will acquire positive associations.

Optimizing for Sentiment Analysis

Sentiment Lexicons: lists of words that are pre-annotated with positive or negative sentiment.



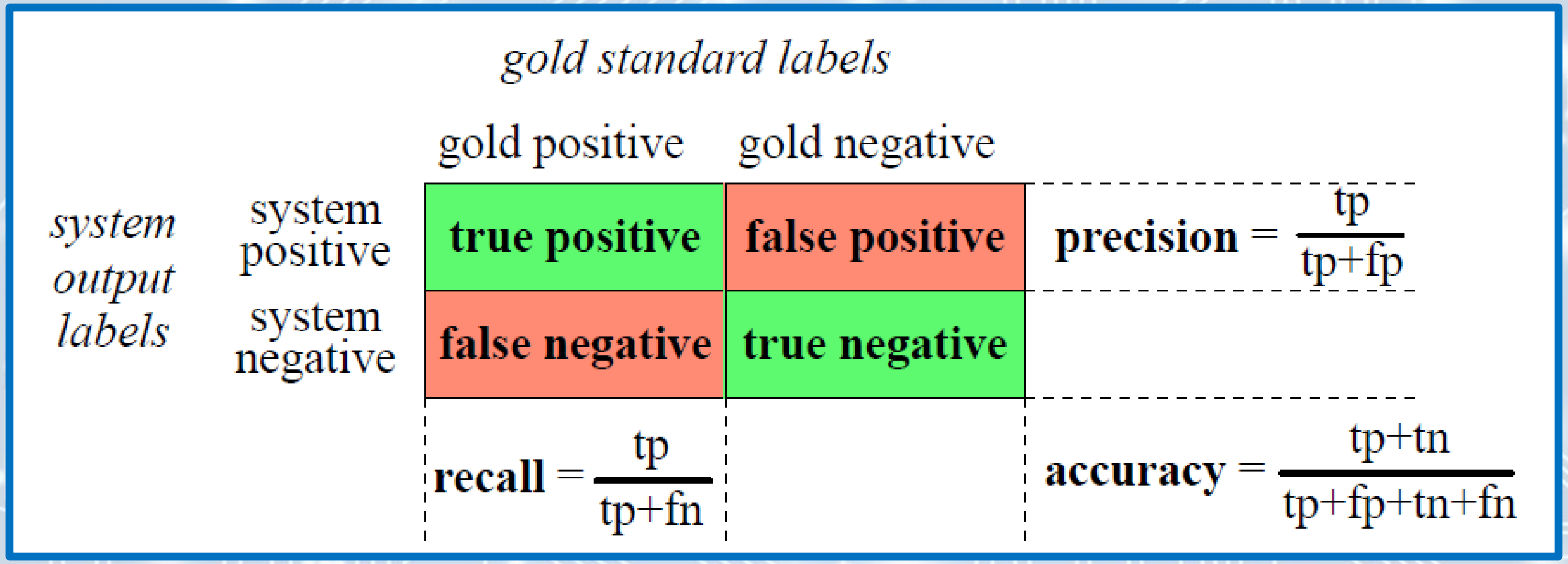
$+$: *admirable, beautiful, confident, dazzling, ecstatic, favor, glee, great*

$-$: *awful, bad, bias, catastrophe, cheat, deny, envious, foul, harsh, hate*

A common way to use lexicons in a naive Bayes classifier is to add a feature that is counted whenever a word from that lexicon occurs.

Evaluation

- A **confusion matrix** for visualizing how well a binary classification system performs against **gold standard** labels.



gold standard labels

system output labels		gold positive	gold negative	
	system positive	**true positive**	**false positive**	**precision** $= \frac{\text{tp}}{\text{tp+fp}}$
	system negative	**false negative**	**true negative**	
		recall $= \frac{\text{tp}}{\text{tp+fn}}$		**accuracy** $= \frac{\text{tp+tn}}{\text{tp+fp+tn+fn}}$

- **F1-score**: is the most frequently used metric that incorporates aspects of both precision and recall in a single metric. (F **"1"** since P and R are treated as **equally important**)

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

Evaluation

- Although accuracy might seem a natural metric, we generally don't use it for text classification tasks

→That's because accuracy doesn't work well when the classes are unbalanced **e.g., we are discovering something that is rare**.

<u>Example</u>**:**

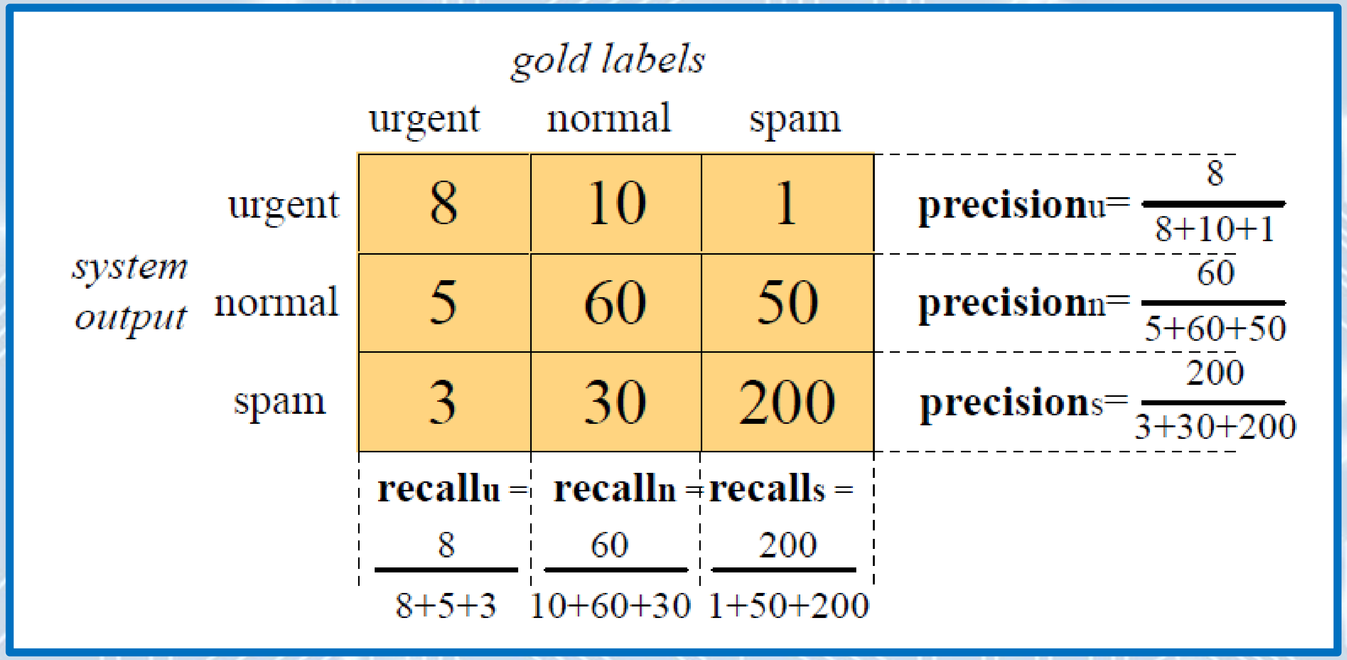
We have a million tweet, only 100 of them are discussing their love (or hatred) for our pie "about pie", while the other 999,900 are about something completely unrelated "not about pie".

- Imagine a simple classifier that stupidly classified every tweet as "not about pie". This classifier would have **999,900 true negatives** and only **100 false negatives**. The accuracy for this model is very high, at 999,900/1,000,000= **99.9%.→** But of course this fabulous 'no pie' classifier would be completely useless, since it wouldn't find a single one of the customer comments we are looking for.
- Solution: other metrics "recall" and "precision". This classifier, despite having a fabulous accuracy of 99.99%, has a terrible recall/precision of 0 (since there are no true positives).

Precision is a good measure, when the cost of False Positive is high.
Recall is a good measure, when the cost of False Negative is high.

Evaluation

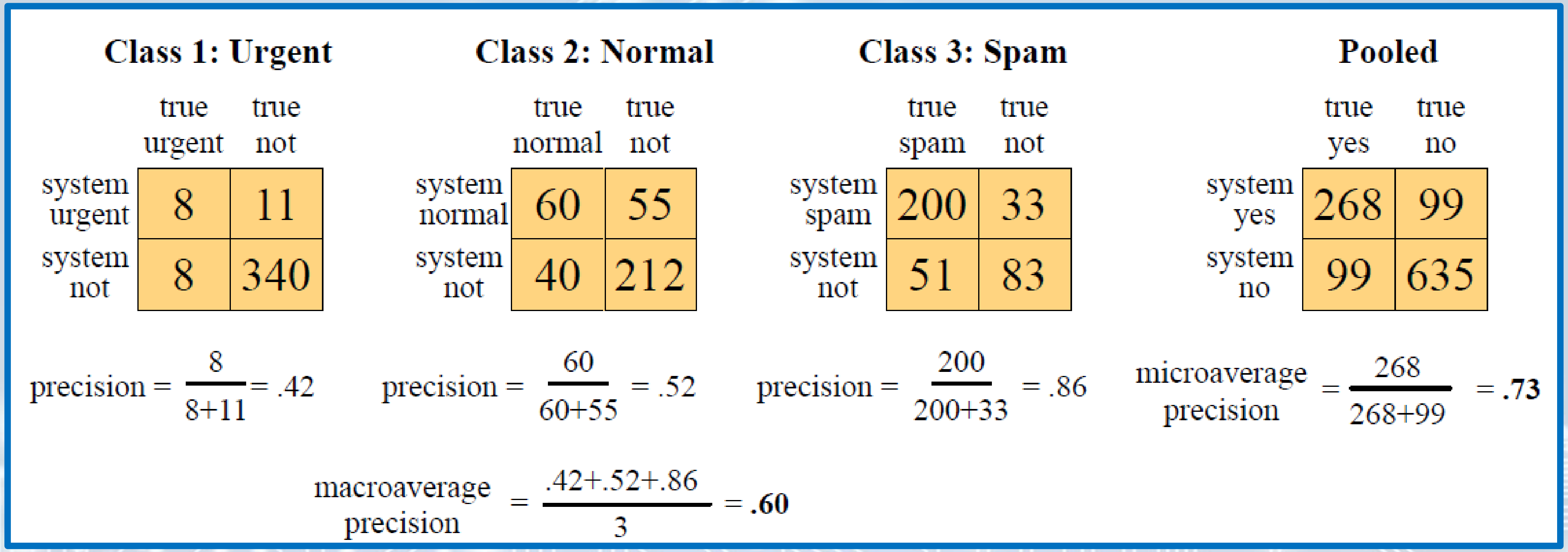
- Confusion matrix for a **three-class** classification task.



		gold labels			
		urgent	normal	spam	
system output	urgent	8	10	1	**precision**u= $\frac{8}{8+10+1}$
	normal	5	60	50	**precision**n= $\frac{60}{5+60+50}$
	spam	3	30	200	**precision**s= $\frac{200}{3+30+200}$
		recallu = $\frac{8}{8+5+3}$	**recall**n = $\frac{60}{10+60+30}$	**recall**s = $\frac{200}{1+50+200}$	

- In the figure, it shown how to compute a **distinct** precision and recall value for each class.
- In order to derive a single metric that tells us how well the system is doing, we can combine these values in two ways:
  - **Macroaveraging**, we compute the performance for each class, and then average over classes.
  - **Microaveraging**, we collect the decisions for all classes into a single confusion matrix, and then compute precision and recall from that table.

Evaluation



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

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

Pooled

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

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

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

- *Microaverage* is dominated by the *more frequent* class (in this case spam), since the counts are pooled.
- *Macroaverage* better reflects the statistics of the *smaller* classes, and so is more appropriate when performance on all the classes is *equally* important.

Thank You