CS447: Natural Language Processing

http://courses.engr.illinois.edu/cs447

Lecture 4: Introduction to Classification for NLP

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

Today's lecture

Very brief intro to classification

Naive Bayes Classifiers for text classification

How to run and evaluate classification experiments

What is Classification?

Spam Detection

Spam detection is a **binary classification** task:
Assign one of two labels (e.g. {SPAM, NOSPAM})
to the input (here, an email message)

Spam Detection

A classifier is a **function** that maps inputs to a predefined (finite) set of class **labels**:

$$\text{Spam Detector: Email} \longmapsto \{\textsc{Spam}, \textsc{NoSpam}\}$$

$$\text{Classifier: Input} \longmapsto \{\textsc{Label}_1, \ldots, \textsc{Label}_K\}$$

The importance of **generalization**

Mail thinks this message is junk mail.

We need to be able to **classify items our classifier has never seen before**.

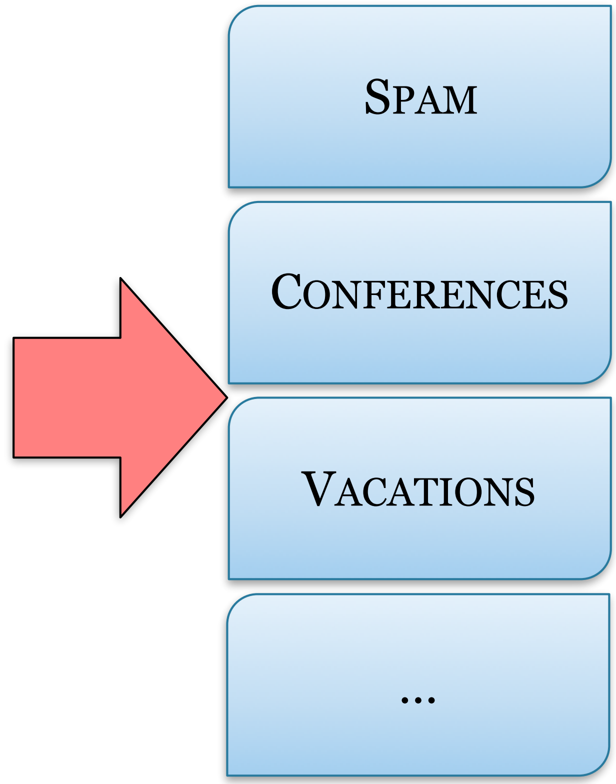
The importance of **adaptation**

Mail thinks this message is junk mail.

Not junk

The classifier needs to adapt/change based on the feedback (**supervision**) it receives

Text classification more generally



This is a **multiclass** classification task:
Assign one of K labels to the input
{SPAM, CONFERENCES, VACATIONS,…}

Classification tasks

Classification tasks: Map inputs to a fixed set of class labels

- Binary classification: each input has exactly one of two classes
- Multi-class classification: each input has exactly one of K classes ($K > 2$)
- Multi-label classification: each input has N of K classes ($N \geq 1$, varies per input)

What are "inputs"?

To talk about machine learning mathematically, we often assume each input item is represented as a vector $\mathbf{x} = (x_1 \dots x_N)$

(The number of elements N is fixed, and may be very large)

In NLP, inputs are documents, sentences, words, ….
⇒ How do we represent these as vectors?

Later today we'll assume that each element x_i in $(x_1 \dots x_N)$ corresponds to one word type (v_i) in the vocabulary $V = \{v_1, \dots, v_N\}$

- If $x_i \in \{0,1\}$: Does word v_i occur in the input document?
- If $x_i \in \{0, 1, 2, \dots\}$: How often does word v_i occur in the input document?

Classification as supervised machine learning

Classification tasks: Map inputs to a fixed set of class labels

Underlying assumption: Each input *really* has one (or N) correct labels
Corollary: The correct mapping is a function (aka the 'target function')

How do we obtain a classifier (model) for a given task?

— If the target function is very simple (and known), implement it directly
— Otherwise, if we have enough correctly labeled data, estimate (aka. learn/train) a classifier based on that labeled data.

Supervised machine learning:

Given (correctly) labeled training data, obtain a classifier that predicts these labels as accurately as possible.

Learning is supervised because the learning algorithm can get feedback about how accurate its predictions are from the labels in the training data.

Supervised machine learning

The supervised learning task (for classification):

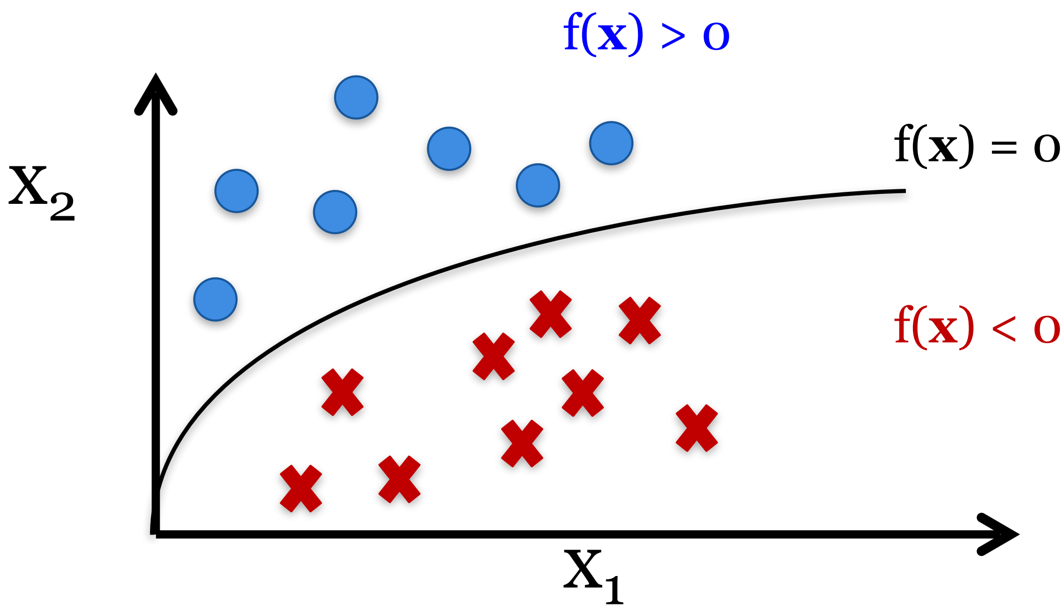
Given (correctly) labeled data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$,
where each item $\mathbf{x}_i$ is a vector $(x_1....x_N)$ with label y_i
(which we assume is given by the target function $f(\mathbf{x}_i) = y_i$),
return a classifier $g(\mathbf{x}_i)$ that predicts these labels as accurately as possible (i.e. such that $g(\mathbf{x}_i) = y_i = f(\mathbf{x}_i)$)

To make this more concrete, we need to specify:

— what class of functions $g(\mathbf{x}_i)$ to consider
(many classifiers assume $g(\mathbf{x}_i)$ is a linear function)

— what learning algorithm we will use to learn $g(\mathbf{x}_i)$
(many learning algorithms assume a particular class of functions)

Classifiers in vector spaces



Binary classification:

Learn a function g that best *separates* the positive and negative examples:

- Assign y = 1 to all **x** where g(**x**) > 0
- Assign y = 0 to all **x** where g(**x**) < 0

Probabilistic classifiers

Return the most likely class *y* for the input **x**:

$$y^* = \textbf{argmax}_y P(Y = y \mid \mathbf{X} = \mathbf{x})$$

We can either model $P(Y = y \mid \mathbf{X} = \mathbf{x})$ directly [*next class*]
or use **Bayes' Rule** ("*the posterior probability P(A|B) is proportional to prior (P(A)) times likelihood P(B|A)*")

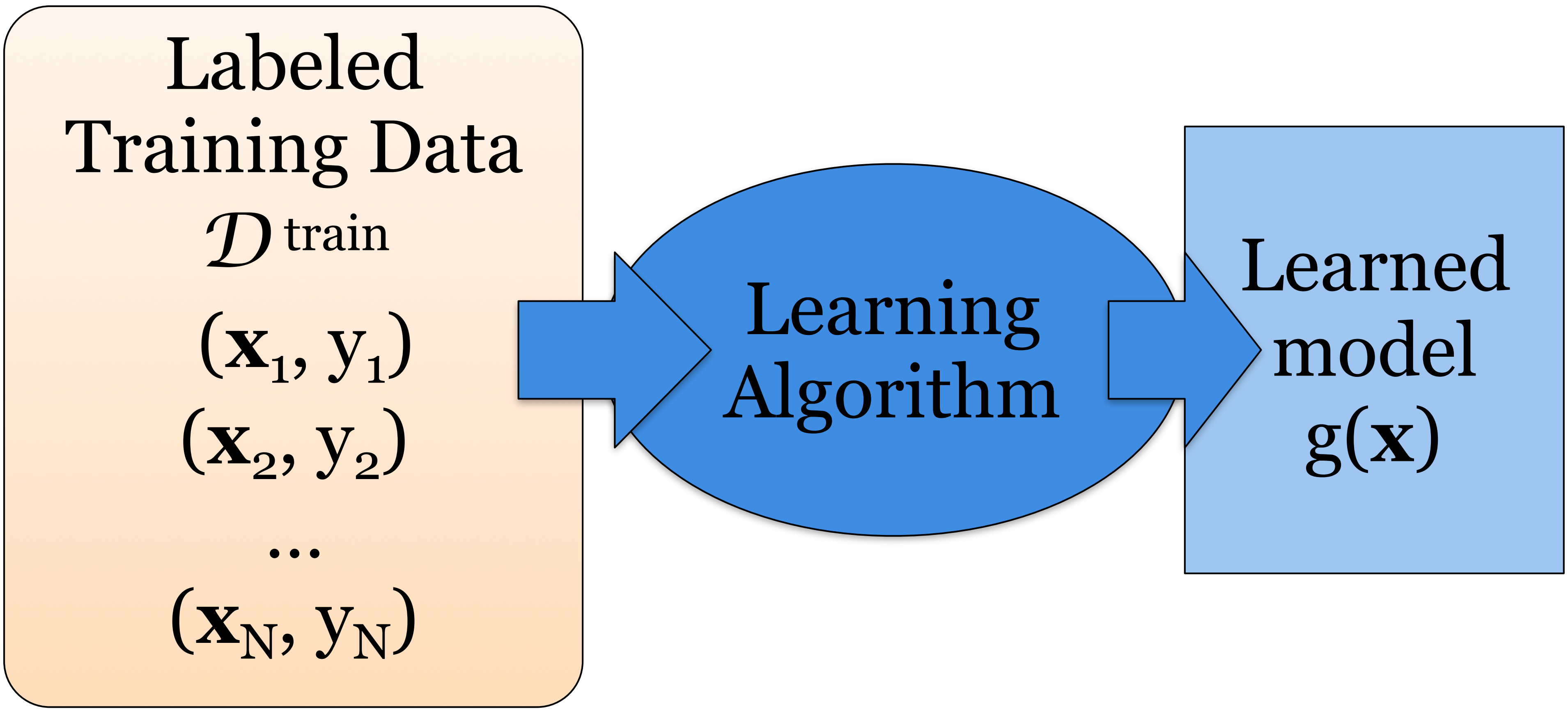
$$P(A \mid B) = \frac{P(A,B)}{P(B)} = \frac{P(B \mid A)P(A)}{P(B)} \propto P(B \mid A)P(A)$$

$$y^* = \textbf{argmax}_y P(Y = y \mid \mathbf{X} = \mathbf{x})$$

$$= \textbf{argmax}_y \frac{P(\mathbf{X} = \mathbf{x} \mid Y = y)P(Y = y)}{P(\mathbf{X} = \mathbf{x})} \quad \text{[Bayes' Rule]}$$

$$= \textbf{argmax}_y P(\mathbf{X} = \mathbf{x} \mid Y = y)P(Y = y) \quad \text{[P(X) doesn't change argmax}_y\text{]}$$

Supervised learning: Training



Give the learning algorithm examples in D^{train}

The learning algorithm returns a model $g(\mathbf{x})$

Supervised learning: Testing

Labeled
Test Data
$\mathcal{D}^{\text{test}}$

$(\mathbf{x}'_1, y'_1)$
$(\mathbf{x}'_2, y'_2)$
…
$(\mathbf{x}'_M, y'_M)$

Reserve some labeled data for testing

Supervised learning: Testing

Raw Test Data $\mathcal{X}^{\text{test}}$

$\mathbf{x}'_1$

$\mathbf{x}'_2$

....

$\mathbf{x}'_M$

Labeled Test Data $\mathcal{D}^{\text{test}}$

$(\mathbf{x}'_1, y'_1)$

$(\mathbf{x}'_2, y'_2)$

…

$(\mathbf{x}'_M, y'_M)$

Test Labels $\mathcal{Y}^{\text{test}}$

y'_1

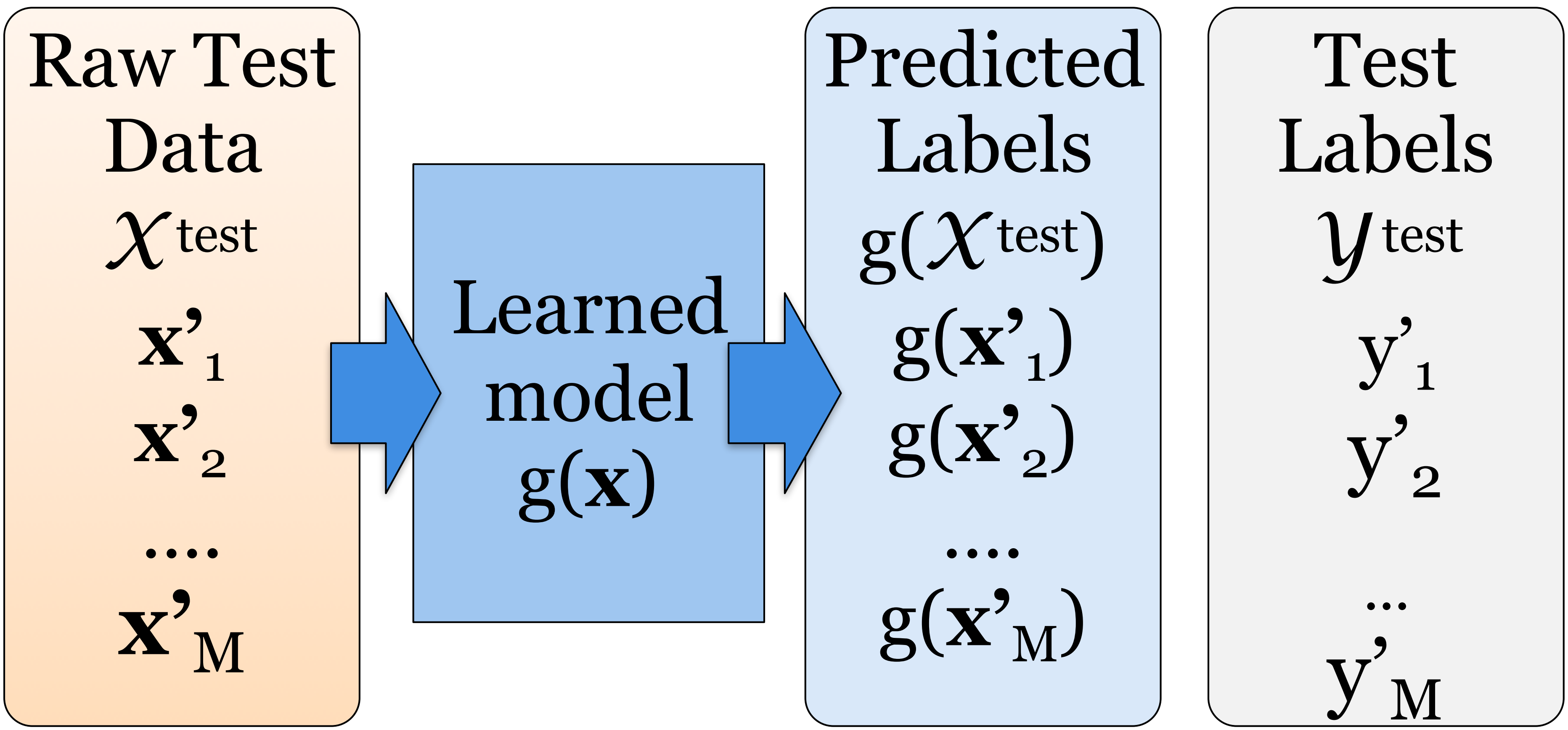
y'_2

…

y'_M

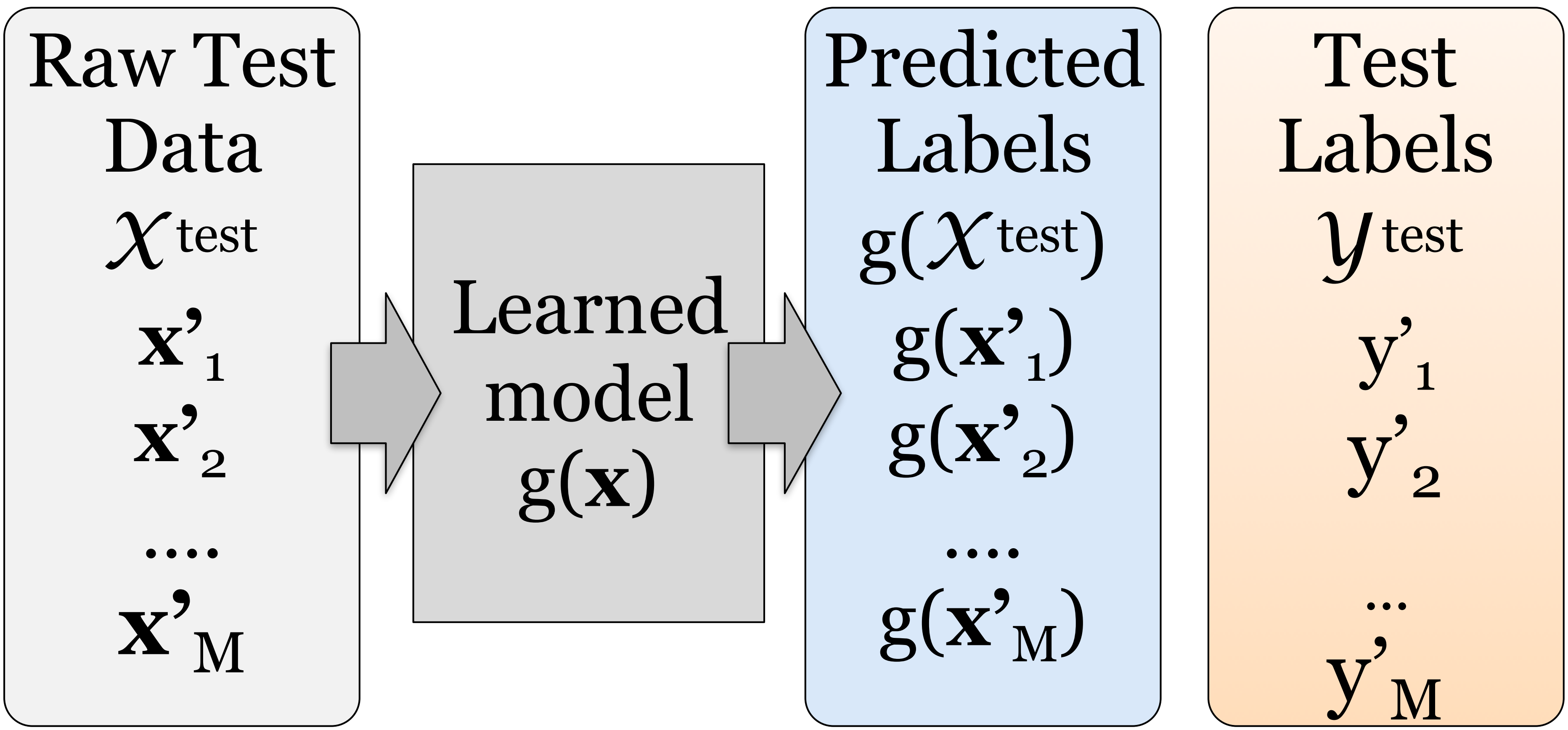
Supervised learning: Testing

Apply the learned model to the raw test data to obtain predicted labels for the test data



Supervised learning: Testing

Evaluate the learned model by comparing the predicted labels against the (correct) test labels



The Naive Bayes Classifier

Probabilistic classifiers

Return the most likely class *y* for the input **x**:

$$y^* = \textbf{argmax}_y P(Y = y \,|\, \mathbf{X} = \mathbf{x})$$

We can either model $P(Y = y \,|\, \mathbf{X} = \mathbf{x})$ directly [*next class*]
or use **Bayes' Rule** ("*the posterior probability P(A|B) is proportional to prior (P(A)) times likelihood P(B|A)*")

$$P(A \,|\, B) = \frac{P(A,B)}{P(B)} = \frac{P(B \,|\, A)P(A)}{P(B)} \propto P(B \,|\, A)P(A)$$

$$\begin{aligned} y^* &= \textbf{argmax}_y P(Y = y \,|\, \mathbf{X} = \mathbf{x}) \\ &= \textbf{argmax}_y \frac{P(\mathbf{X} = \mathbf{x} \,|\, Y = y)P(Y = y)}{P(\mathbf{X} = \mathbf{x})} \quad \text{[Bayes' Rule]} \\ &= \textbf{argmax}_y P(\mathbf{X} = \mathbf{x} \,|\, Y = y)P(Y = y) \quad \text{[P(X) doesn't change argmax}_y\text{]} \end{aligned}$$

Modeling $P(\mathbf{X} = \mathbf{x} \mid Y = y)P(Y = y)$

$P(Y = y)$ is the "prior" class probability

We can estimate this as the fraction of documents in the training data that have class *y:*

$$\hat{P}(Y = y) = \frac{\text{\#documents } \langle \mathbf{x}_i, y_i \rangle \in D_{train} \text{with } y_i = y}{\text{\#documents } \langle \mathbf{x}_i, y_i \rangle \in D_{train}}$$

$P(\mathbf{X} = \mathbf{x} \mid Y = y)$ is the "likelihood" of the input **x**

x = ($x_1 \ldots x_n$) is a vector; each $x_i \approx$ a word in our vocabulary

Let's make a (naive) independence assumption:

$$P(\mathbf{X} = \langle x_1, \ldots, x_n \rangle \mid Y = y) := \prod_{i=1..n} P(X_i = x_i \mid Y = y)$$

Now we need to multiply together all $P(X_i = x_i \mid Y = y)$

The Naive Bayes Classifier

Assign class y* to input **x** = ($x_1 \ldots x_n$) if

$$y^* = \textbf{argmax}_y P(Y = y) \prod_{i=1..n} P(X_i = x_i \mid Y = y)$$

$P(Y = y)$ is the prior class probability (estimated as the fraction of items in the training data with class y)

$P(X_i = x_i \mid Y = y)$ is the (class-conditional) likelihood of the feature x_i.

There are different ways to model this probability

$P(X_i = x_i \mid Y = y)$ as Bernoulli

$P(X_i = x_i \mid Y = y)$ is a **Bernoulli** distribution ($x_i \in \{0,1\}$)

$P(X_i = 1 \mid Y = y)$ is the probability that word v_i occurs in a document of class *y*.

$P(X_i = 0 \mid Y = y)$ is the probability that word v_i does not occur in a document of class *y*

Estimation:

$$\hat{P}(X_i = 1 \mid Y = y) = \frac{\textbf{\#docs } \langle \mathbf{x}_i, y_i \rangle \in D_{train} \textbf{ with } y_i = y \textbf{ in which } x_i \textbf{ occurs}}{\textbf{\#docs } \langle \mathbf{x}_i, y_i \rangle \in D_{train} \textbf{ with } y_i = y}$$

$$\hat{P}(X_i = 0 \mid Y = y) = \frac{\textbf{\#docs } \langle \mathbf{x}_i, y_i \rangle \in D_{train} \textbf{ with } y_i = y \textbf{ in which } x_i \textbf{ does not occur}}{\textbf{\#docs } \langle \mathbf{x}_i, y_i \rangle \in D_{train} \textbf{ with } y_i = y}$$

$P(\mathbf{X}_i = \mathbf{x}_i \mid Y = y)$ as Multinomial

$P(\mathbf{X}_i = \mathbf{x}_i \mid Y = y)$ is a **Multinomial:** $(x_i \in \{0,1,2,...\})$

$P(X_i = x_i \mid Y = y)$ is the probability that word v_i occurs with frequency x_i (= 0, 1, 2, …) in a document of class y.

Recall: a multinomial computes the probability of, say, getting three 6s and two 5s if you roll a die five times:

$$P(\langle 0,0,0,0,2,3 \rangle) = \frac{5!}{0!0!0!0!2!3!}(1/6)^2(1/6)^3$$

#of sequences of three 6s and two 5s: 5!/(0!0!0!0!2!3!)

Prob. of getting a 5 (or a 6) when you roll a die once = 1/6

Prob. of any one sequence of three 6s and two 5s: $(1/6)^2(1/6)^3$

Note that we can now ignore the probabilities of any sides (1, 2, 3, 4) that didn't come up in our trial

$P(\mathbf{X}_i = \mathbf{x}_i \mid Y = y)$ as Multinomial

We want to know P(**x** = (0,0,0,0,2,3) | y = SPAM) for a given **x**:

— Words do not have uniform probability (language ≠ dice)
— We need to know the class-conditional unigram probability $P(v_i \mid Y = y)$ of word v_i in all documents of class y
– We also do not need to worry about the probability of the particular *sequence* of words in our document

So, for us:

$$P(\langle 0,0,0,0,2,3\rangle \mid Y = y) = P(v_5 \mid Y = y)^2 P(v_6 \mid Y = y)^3$$

Unigram probabilities $P(v_i \mid Y = y)$

We can estimate the unigram probability $P(v_i \mid Y = y)$ of word v_i in all documents of class y as

$$\hat{P}(v_i \mid Y = y) = \frac{\#v_i \text{ in all docs} \in D_{\text{train}} \text{of class } y}{\#\text{words in all docs} \in D_{\text{train}} \text{of class } y}$$

or with add-one smoothing (with *N* words in vocab V):

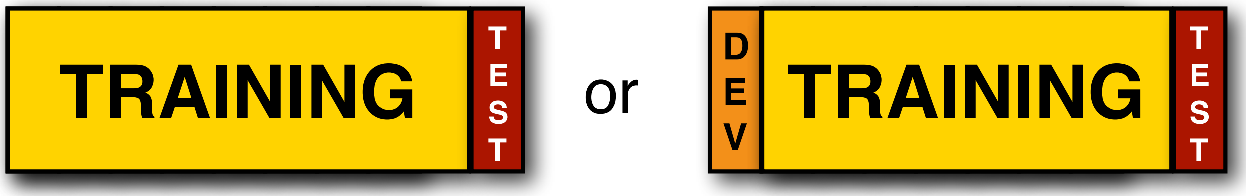
$$\hat{P}(v_i \mid Y = y) = \frac{(\#v_i \text{ in all docs} \in D_{\text{train}} \text{of class } y) + 1}{(\#\text{words in all docs} \in D_{\text{train}} \text{of class } y) + N}$$

Running and Evaluating Classification Experiments

Evaluating Classifiers

Evaluation setup:

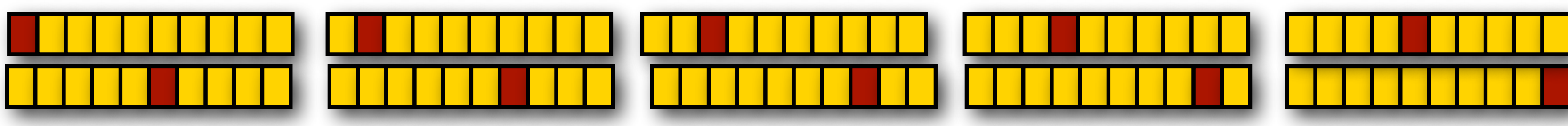
Split data into separate **training,** (**dev**elopment) and **test** sets.



Better setup: **n-fold cross validation**:

Split data into *n* sets of equal size

Run *n* experiments, using set *i* to test and remainder to train



This gives average, maximal and minimal accuracies

When **comparing two classifiers**:

Use the **same** test and training data with the same classes

Evaluation Metrics

Accuracy: How many documents in the test data did you classify correctly?

It's easy to get high accuracy if one class is very common (just label everything as that class)

But that would be a pretty useless classifier

Precision and recall

Precision and recall were originally developed as evaluation metrics for information retrieval:

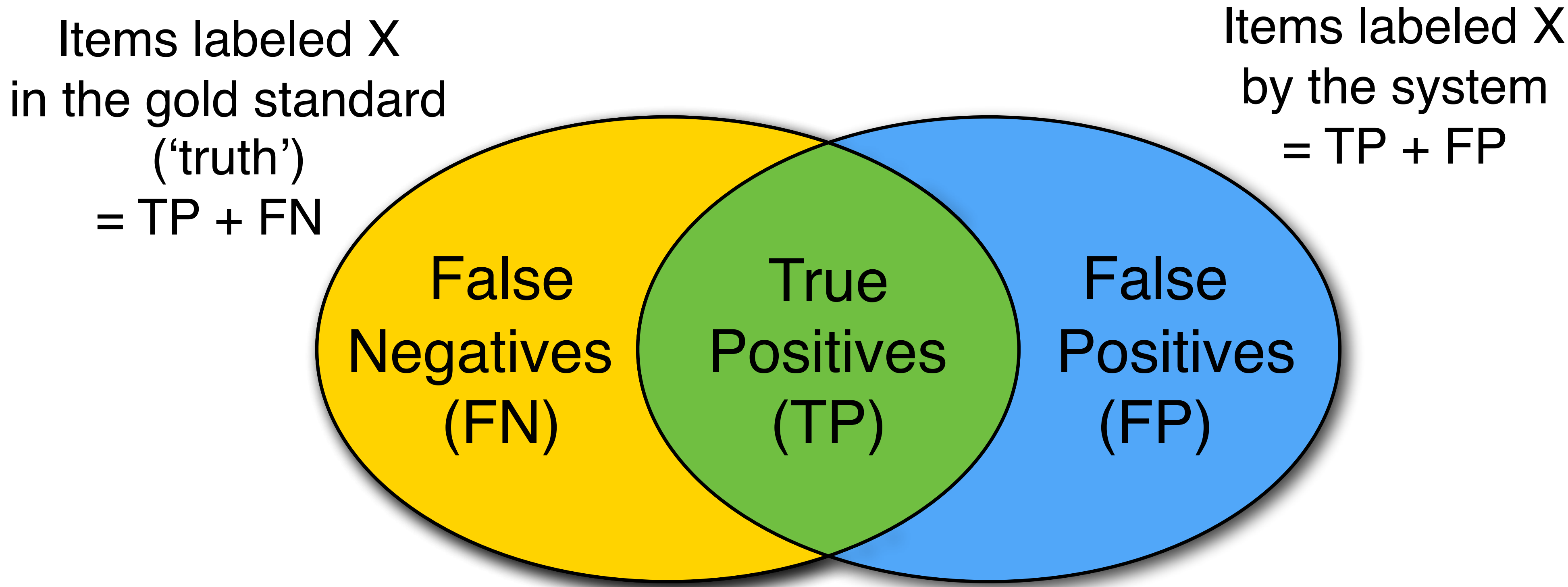
- **Precision:** What percentage of retrieved documents are relevant to the query?
- **Recall**: What percentage of relevant documents were retrieved?

In NLP, they are often used in addition to accuracy:

- **Precision:** What percentage of items that were assigned label X do actually have label X in the test data?
- **Recall:** What percentage of items that have label X in the test data were assigned label X by the system?

Particularly useful when there are more than two labels.

True vs. false positives, false negatives

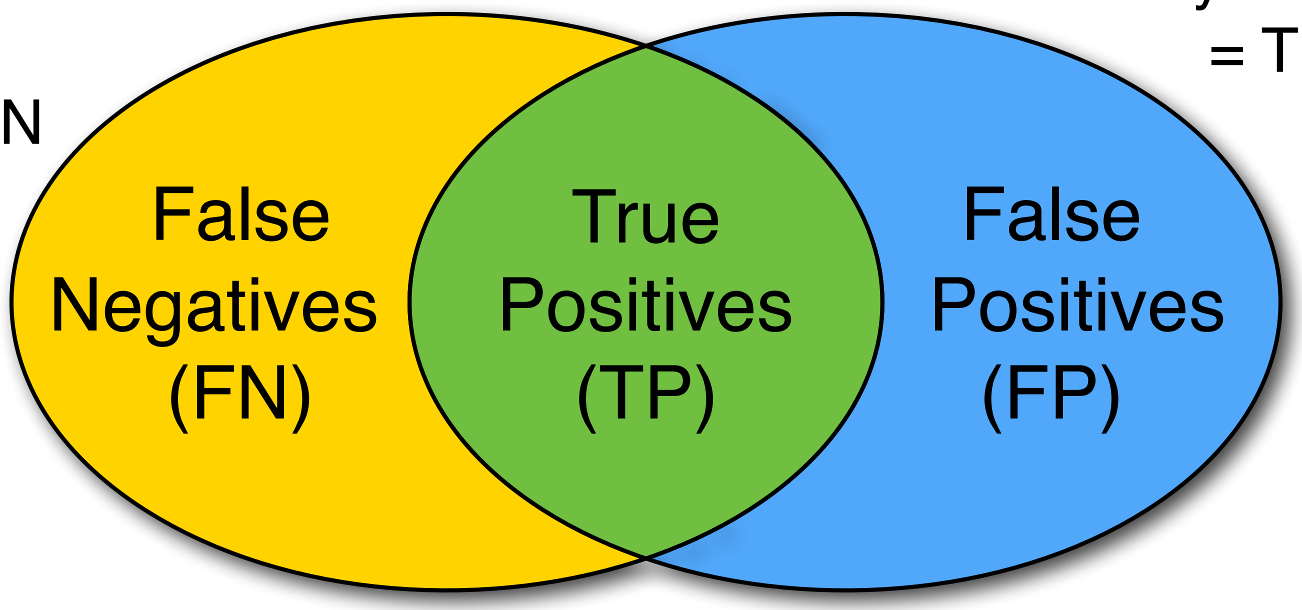


- True positives: Items that were labeled X by the system, and should be labeled X.
- False positives: Items that were labeled X by the system, but should not be labeled X.
- False negatives: Items that were not labeled X by the system, but should be labeled X,

Precision, recall, f-measure

Items labeled X
in the gold standard
('truth')
= TP + FN

Items labeled X
by the system
= TP + FP



Precision: P = TP / (TP + FP)

Recall: R = TP / (TP + FN)

F-measure: harmonic mean of precision and recall

$$F = (2 \cdot P \cdot R) / (P + R)$$

Confusion matrices

		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 = $\frac{200}{1+50+200}$	

Figure 4.5 Confusion matrix for a three-class categorization task, showing for each pair of classes (c_1, c_2), how many documents from c_1 were (in)correctly assigned to c_2

Confusion matrices

gold labels

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

Figure 4.5 Confusion matrix for a three-class categorization task, showing for each pair of classes (c_1, c_2), how many documents from c_1 were (in)correctly assigned to c_2

Macro-average vs Micro-average

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

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

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

Figure 4.6 Separate contingency tables for the 3 classes from the previous figure, showing the pooled contingency table and the microaveraged and macroaveraged precision.

Macro-average: average the precision over all classes
(regardless of how common each class is)

Micro-average vs Macro-average

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

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

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

Figure 4.6 Separate contingency tables for the 3 classes from the previous figure, showing the pooled contingency table and the microaveraged and macroaveraged precision.

Macro-average: average the precision over all classes
(regardless of how common each class is)

Micro-average: average the precision over all items
(regardless of which class they have)