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

Lecture 04, Part 1: Review and Overview

Review: Lecture 03

Language models define a probability distribution over all strings $\mathbf{w}=w^{(1)}...w^{(K)}$ in a language:

$$\sum_{\mathbf{w}\in L} P(\mathbf{w}) = 1$$

N-gram language models define the probability of a string $\mathbf{w}=w^{(1)}...w^{(K)}$ as the product of the probabilities of each word $w^{(i)}$, conditioned on the n–1 preceding words:

$$P_{n-gram}(w^{(1)}\ldots w^{(K)}) = \prod_{i=1..K} P(w^{(i)} \,|\, w^{(i-1)}, \ldots, w^{(i-n+1)})$$

Unigram: $P_{unigram}(w^{(1)}\ldots w^{(K)}) = \prod_{i=1..K} P(w^{(i)})$

Bigram: $P_{bigram}(w^{(1)}\ldots w^{(K)}) = \prod_{i=1..K} P(w^{(i)} \,|\, w^{(i-1)})$

Trigram: $P_{trigram}(w^{(1)}\ldots w^{(K)}) = \prod_{i=1..K} P(w^{(i)} \,|\, w^{(i-1)}, w^{(i-2)})$

Review: Lecture 03

How do we…

…estimate the parameters of a language model?

Relative frequency estimation (aka Maximum Likelihood estimation)

… compute the probability of the first n–1 words?

By padding the start of the sentence with n–1 BOS tokens

… obtain *one* distribution over strings of *any* length?

By adding an EOS token to the end of each sentence.

… handle unknown words?

By replacing rare words in training and unknown words with an UNK tokens

… evaluate language models?

Intrinsically with perplexity of test data, extrinsically e.g. with word error rate

Overview: Lecture 04

Part 1: Review and Overview
Part 2: What is classification?
Part 3: The Naive Bayes classifier
Part 4: Running&evaluating classification experiments
~~Part 5: Features for Sentiment analysis~~

Reading:
Chapter 4, 3rd edition of Jurafsky and Martin

Lecture 04's questions

What is classification?

What is binary/multiclass/multilabel classification?

What is supervised learning?

And why do we want to learn classifiers
(instead of writing down some rules, say)?

Feature engineering: from data to vectors

How is the Naive Bayes Classifier defined?

How do you evaluate a classifier?

Lecture 04, Part 2: 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**:

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

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

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.**

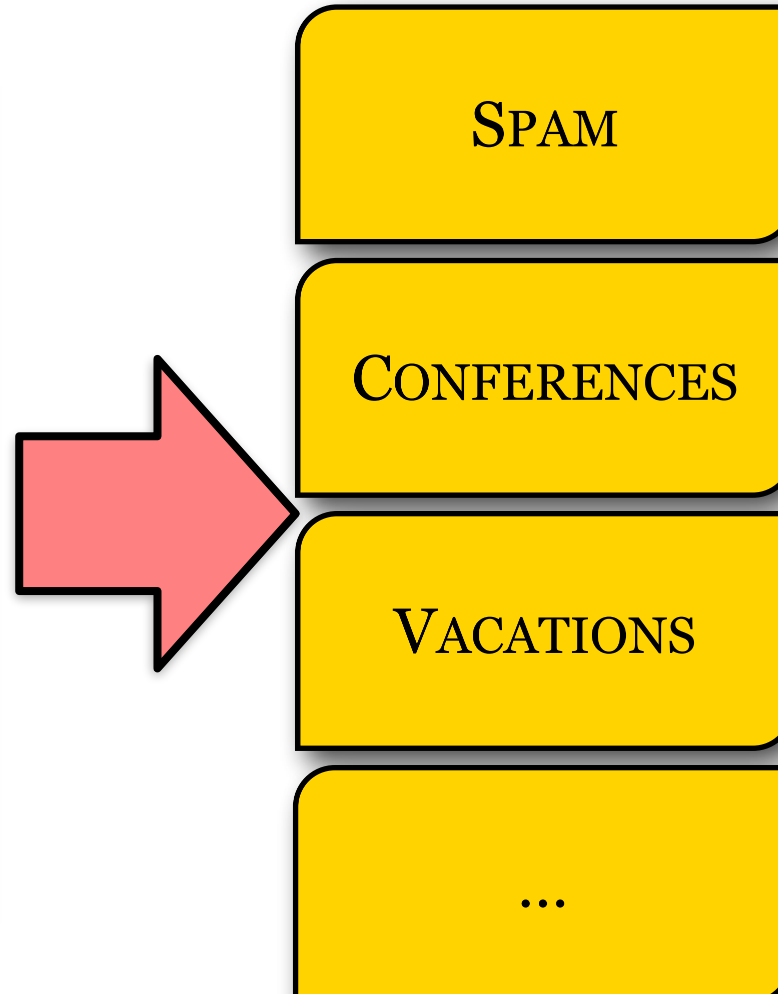
The importance of **adaptation**

Mail thinks this message is junk mail.

Not junk

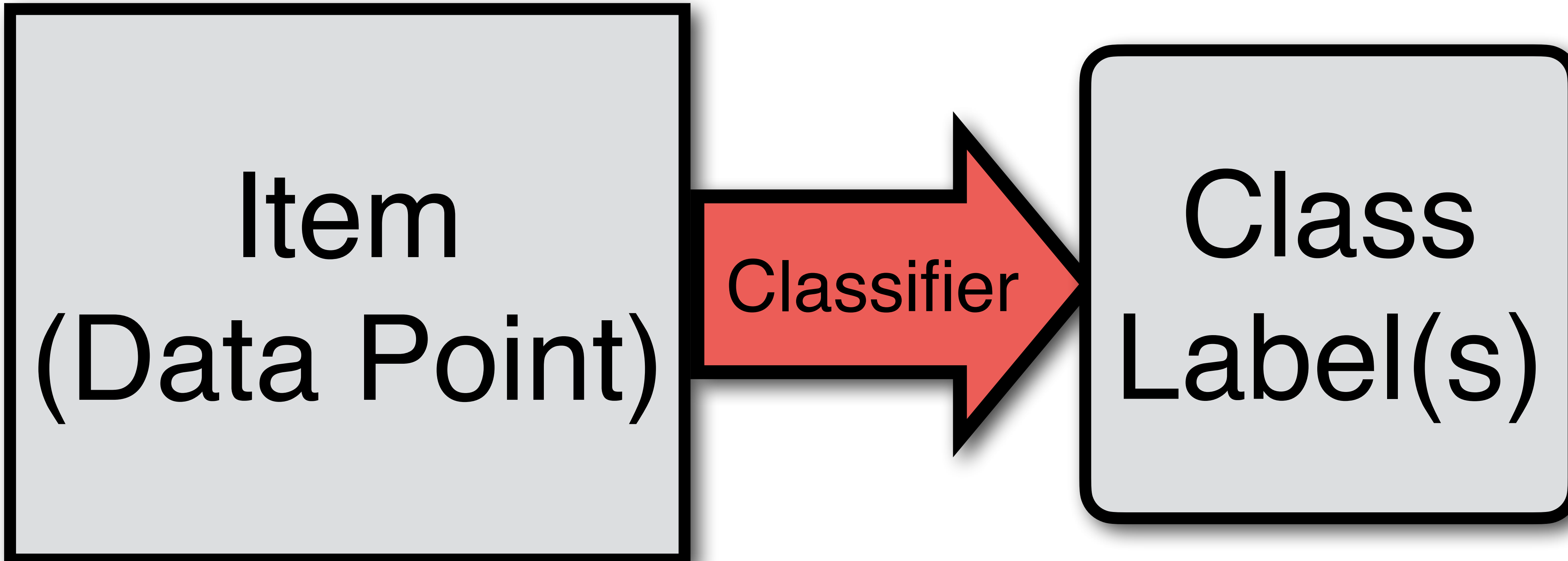
The classifier needs to adapt/change based on the feedback (**supervision**) it receives

Text classification more generally



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

Classification more generally

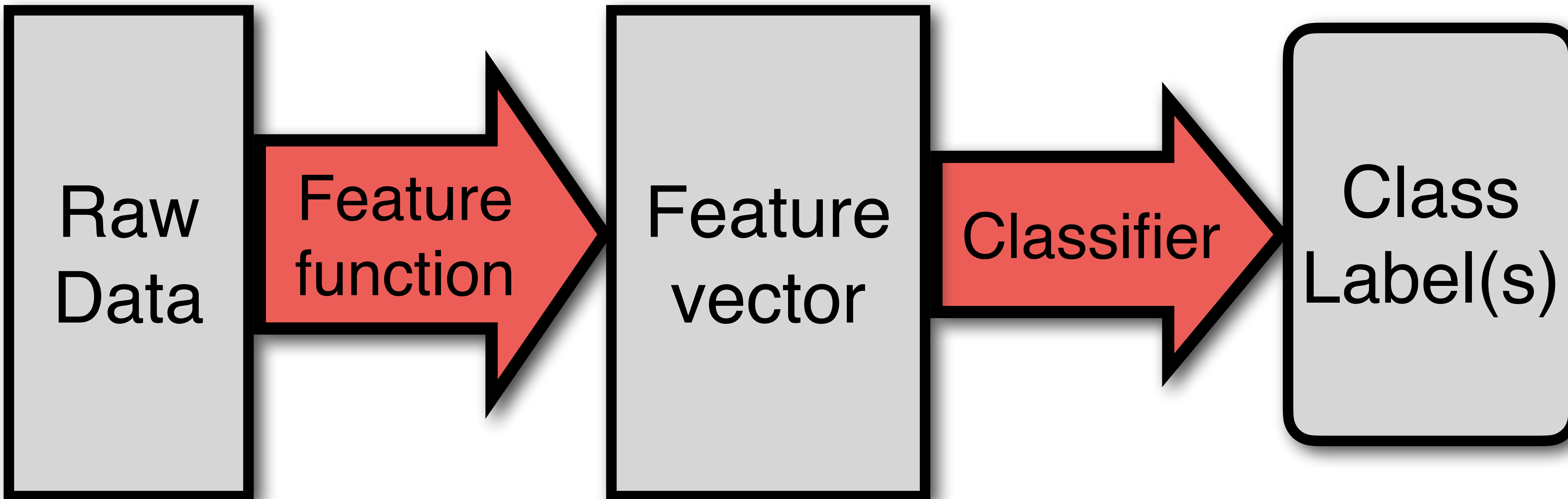


But: The data we want to classify could be *anything*:

Emails, words, sentences, images, image regions, sounds, database entries, sets of measurements, ….

We assume that *any* data point can be represented as a **vector**

Classification more generally



Before we can use a classifier on our data, we have to map the data to **"feature" vectors**

Feature engineering as a prerequisite for classification

To talk about classification mathematically, we assume each input item is represented as a **'feature' vector** $\mathbf{x} = (x_1 \ldots . x_N)$

- Each element in **x** is one feature.
- The number of elements/features N is fixed, and may be very large.
- **x** has to capture *all* the information about the item that the classifier needs.

But the raw data points (e.g. documents to classify) are typically not in vector form.

Before we can train a classifier, we therefore have to first define a suitable **feature function** that maps raw data points to vectors.

In practice, **feature engineering** (designing suitable feature functions) is very important for accurate classification.

From texts to vectors

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

Bag-of-Words representation: (this ignores word order)
Assume that each element x_i in $(x_1 \ldots . x_N)$ corresponds to one word type (v_i) in the vocabulary $V = \{v_1, \ldots, v_N\}$

There are many different ways to represent a piece of text as a vector over the vocabulary, e.g.:

- If $x_i \in \{0,1\}$: Does word v_i occur (yes: $x_i = 1$, no: $x_i = 0$) in the input document?
- If $x_i \in \{0, 1, 2, \ldots\}$: How often does word v_i occur in the input document?

[We will see many other ways to map text to vectors this semester]

Now, back to classification…:

A **classifier** is a function $f(\mathbf{x})$ that maps
input items $\mathbf{x} \in X$ to class labels $y \in Y$
(X is a vector space, Y is a finite set)

Binary classification:
Each input item is mapped to exactly one of 2 classes

Multi-class classification:
Each input item is mapped to exactly one of K classes (K > 2)

Multi-label classification:
Each input item is mapped to N of K classes
(N ≥1, varies per input item)

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?

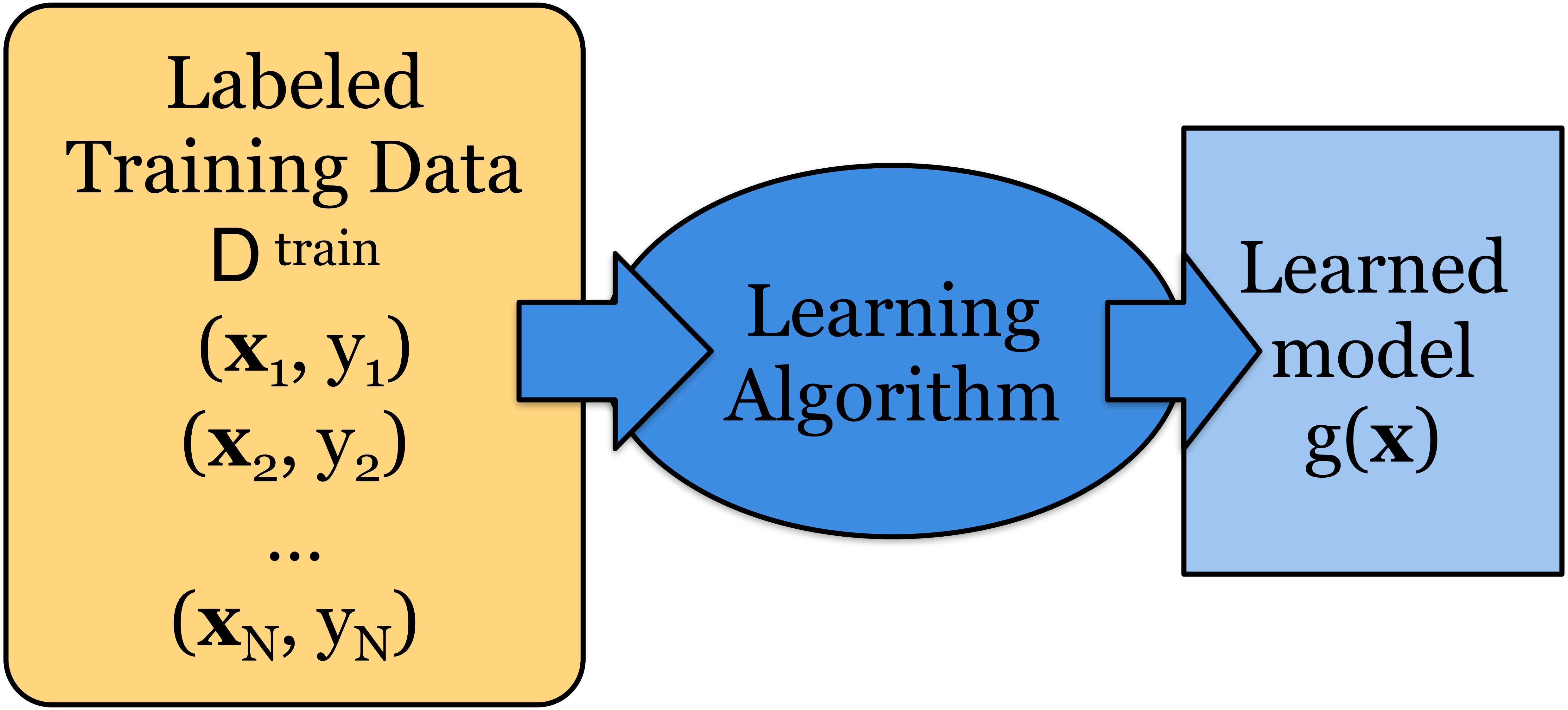
— If the target function is very simple (and known), implement it directly
— Otherwise, if we have enough correctly labeled data, estimate (aka. learn/train) a classifier based on that labeled data.

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 learning: **Training**



Give the learning algorithm examples in D^{train}

The learning algorithm returns a model g(**x**)

Supervised learning: **Testing**

Labeled
Test Data
D^{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

X_{test}

$\mathbf{x}'_1$

$\mathbf{x}'_2$

....

$\mathbf{x}'_M$

Labeled Test Data

D_{test}

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

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

…

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

Test Labels

Y_{test}

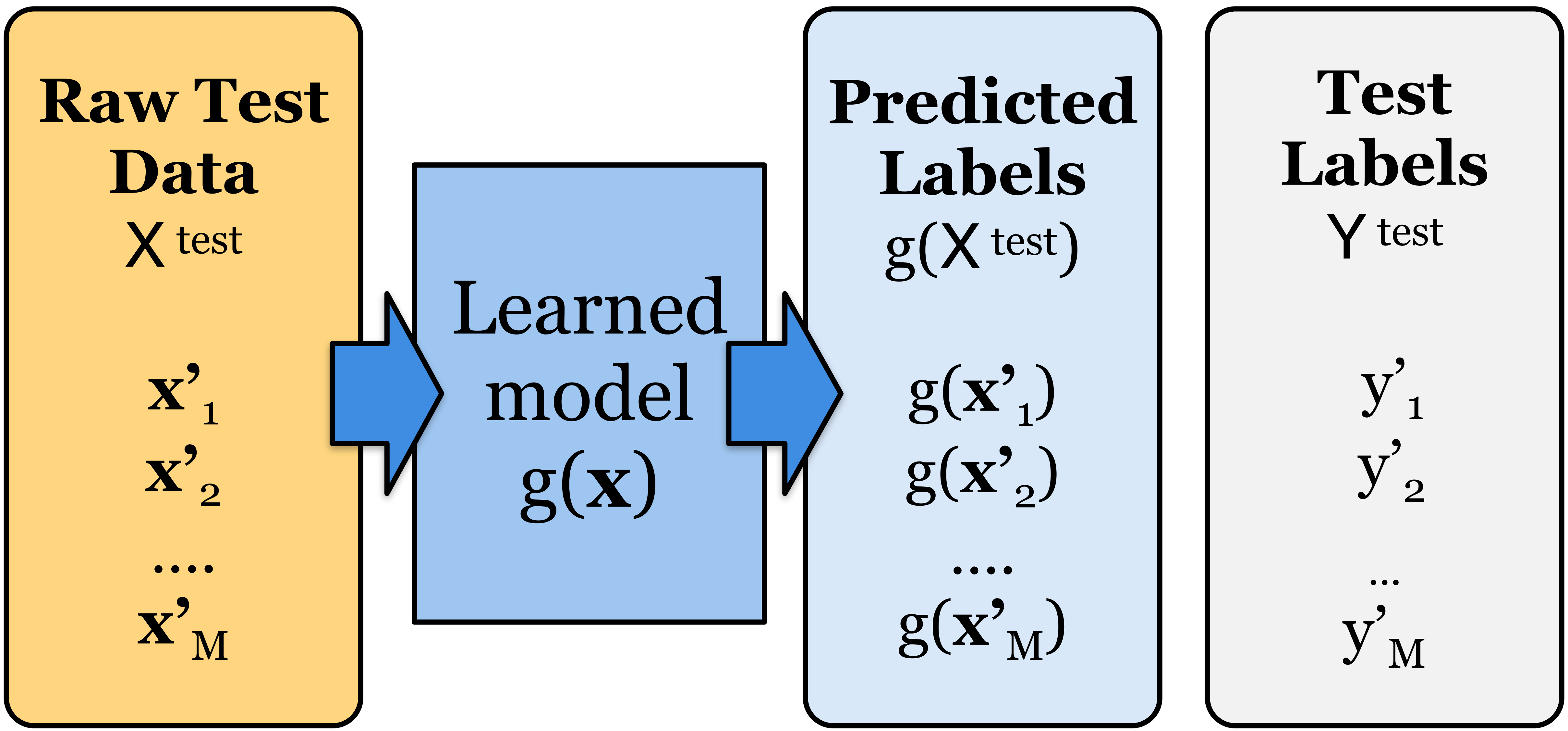
y'_1

y'_2

…

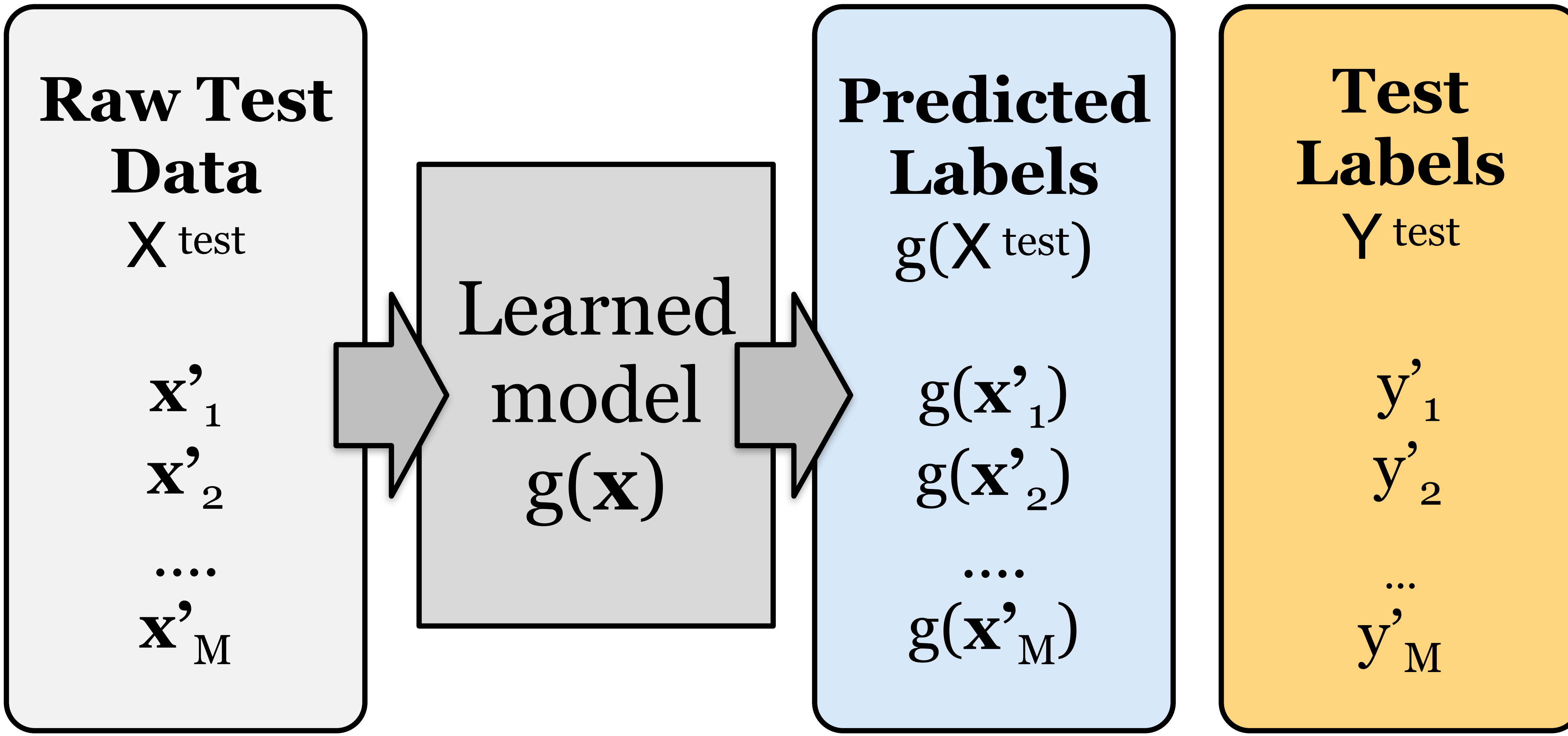
Supervised learning: **Testing**

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



Supervised machine learning

The supervised learning task (for classification):

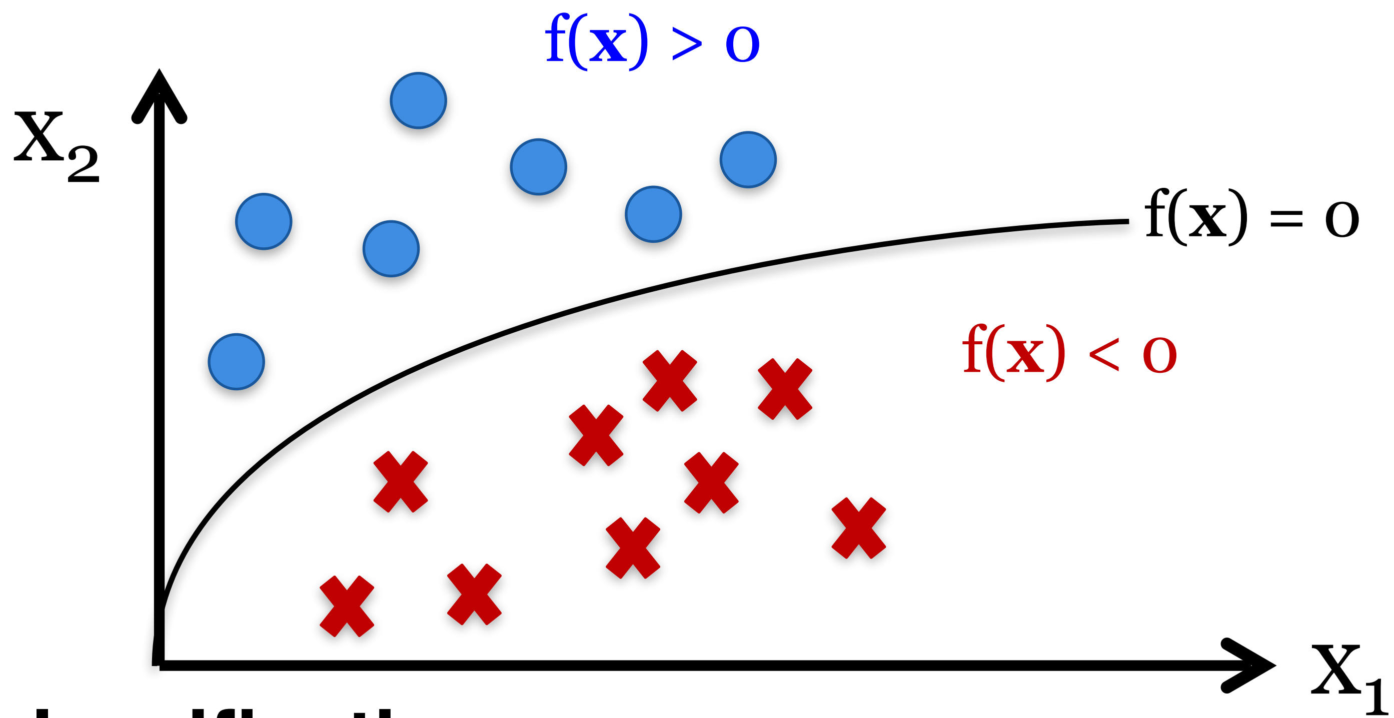
Given (correctly) labeled data $D = \{(\mathbf{x}_i, y_i)\}$,
where each item $\mathbf{x}_i$ is a vector $(x_1 \ldots x_N)$ with label y_i
(which we assume is given by the target function $f(\mathbf{x}_i) = y_i$),
return a classifier $g(\mathbf{x}_i)$ that predicts these labels as accurately as possible (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 f that best *separates* the positive and negative examples:

— Assign y = 1 to all $\mathbf{x}$ where $f(\mathbf{x}) > 0$

— Assign y = 0 to all $\mathbf{x}$ where $f(\mathbf{x}) < 0$

Linear classifier: $f(\mathbf{x}) = \mathbf{wx} + b$ is a **linear** function of $\mathbf{x}$

Lecture 04, Part 3: The Naive Bayes Classifier

Probabilistic classifiers

We want to find the *most likely* class y for the input $\mathbf{x}$:

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

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

The probability that the class label is y when the input feature vector is $\mathbf{x}$

$y^* = \text{argmax}_y \; f(y)$

Let y^* be the y that maximizes $f(y)$

Modeling $P(Y|X)$ with Bayes Rule

Bayes Rule relates $P(Y|X)$ to $P(X|Y)$ and $P(Y)$:

$$\underbrace{P(Y|X)}_{\text{Posterior}} = \frac{P(Y,X)}{P(X)}$$

$$= \frac{P(X|Y)P(Y)}{P(X)}$$

$$\propto \overbrace{P(X|Y)}^{\text{Likelihood}}\overbrace{P(Y)}^{\text{Prior}}$$

Bayes rule: The **posterior** $P(Y \mid X)$ is proportional to the **prior** $P(Y)$ times the **likelihood** $P(X|Y)$

Using Bayes Rule for our classifier

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

$$= \text{argmax}_y \frac{P(\mathbf{X} \mid Y)P(Y)}{P(\mathbf{X})}$$

[Bayes Rule]

$$= \text{argmax}_y P(\mathbf{X} \mid Y)P(Y)$$

[$P(X)$ doesn't change argmax_y]

Modeling $P(Y = y)$

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

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

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

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

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

$\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ is a vector

Each x_i represents a word (type) in our vocabulary

Let's make a (naive) **independence assumption**:

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

With this independence assumption, we now need to define (and multiply together) all $P(X_i = x_i \mid Y = y)$

The Naive Bayes Classifier

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

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

$P(Y=y)$ 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 conditioned on the class y.
There are different ways to model this probability.

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

Capture **whether a word occurs** in a document or not:
$P(X_i = x_i \mid Y = y)$ is a **Bernoulli** distribution $(x_i \in \{0,1\})$

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

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

Estimation:

Compute the fraction of documents of class y with/without x_i:

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

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

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

What if we want to capture ***how often*** a word appears in a document?

Let's represent each document as a vector of **word frequencies** $x_i = C(v_i)$:

Vocabulary $V = \{$apple, banana, coffee, drink, eat, fish$\}$

A document: *"fish fish eat eat fish"*

Vector representation of this document: $\mathbf{x} = \langle 0,0,0,0,2,3 \rangle$

$P(X_i = x_i \mid Y = y)$: probability that word v_i occurs with frequency $x_i = C(v_i)$ in a document of class y.

We can model this by treating $P(\mathbf{X} \mid Y)$ as a **Multinomial distribution**

Multinomial Distribution: Rolling Dice

Before we look at language, let's assume we're **rolling dice**, where the **probability of getting any one side** (e.g. a 4) when rolling the die once is **equal** to that of any other side (e.g. a 6).

A **multinomial** computes the probability of, say, getting two 5s and three 6s 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

#Occurrences of 5 and 3: 2 and 3

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

NB: Note that we can ignore the probabilities of any sides (i.e. 1, 2, 3, 4) that didn't come up in our trial (unlike in the Bernoulli model)

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

We want to know $P(\mathbf{X} = \langle 0,0,0,0,2,3 \rangle \mid Y = y)$
where $\langle 0,0,0,0,2,3 \rangle = \langle C(apple), \ldots, C(eat), C(fish) \rangle$

Unlike the sides of a dice, words don't have uniform probability (cf. Zipf's Law)

So we need to **estimate the class-conditional unigram probability** $P(apple \mid Y = y)$ **of each word** v_i {apple,…, fish} in documents of class y…

… **and multiply that probability** x_i **times**
(x$_i$ = frequency of v$_i$ in our document):

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

Or more generally: $P(\mathbf{X} = \mathbf{x} \mid Y = y) = \prod P(v_i \mid Y = y)^{x_i}$

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):

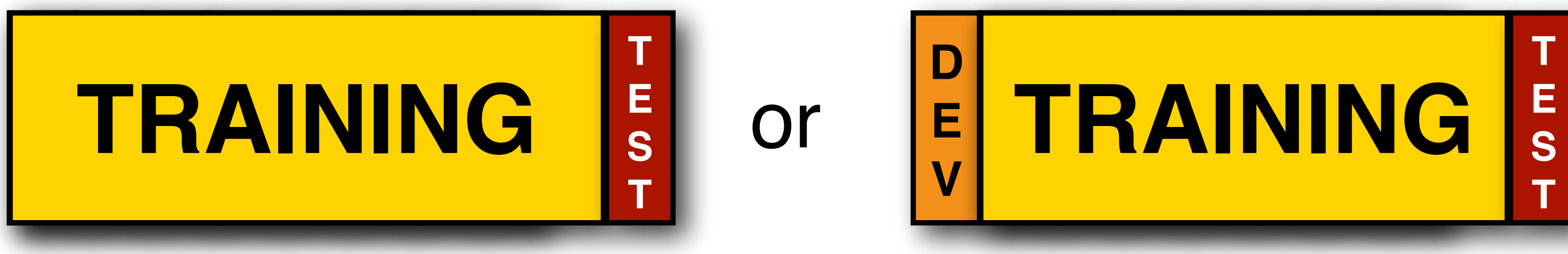
$$\hat{P}(v_i \mid Y = y) = \frac{(\#v_i \text{ in all docs} \in D_{\text{train}} \text{of class } y) + 1}{(\#\text{words in all docs} \in D_{\text{train}} \text{of class } y) + N}$$

Lecture 04, Part 4:
Running and Evaluating
Classification
Experiments

Evaluating Classifiers

Evaluation setup:

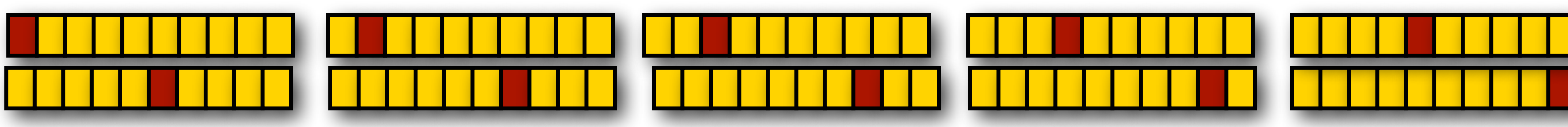
Split data into separate **training,** (**dev**elopment) and **test** sets.



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: What fraction of items in the test data were classified 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:

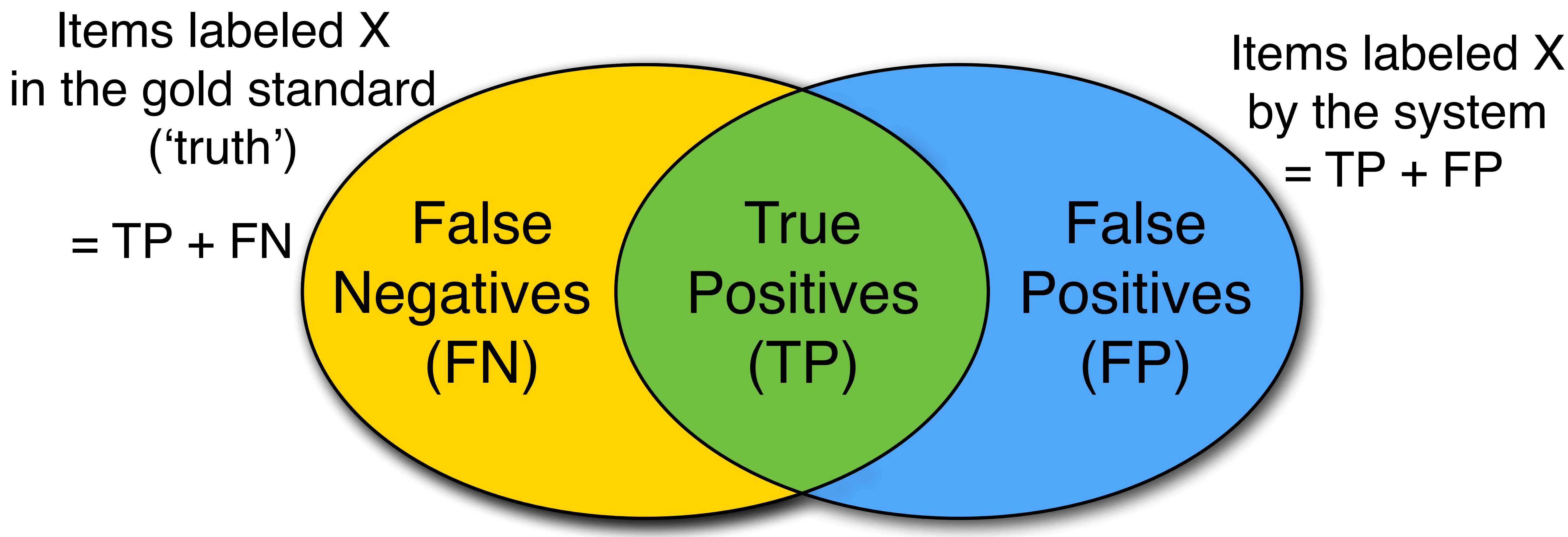
- **Precision**: What percentage of retrieved documents are relevant to the query?
- **Recall**: What percentage of relevant documents were retrieved?

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?

Precision and Recall are particularly useful when there are more than two labels.

True vs. false positives, false negatives

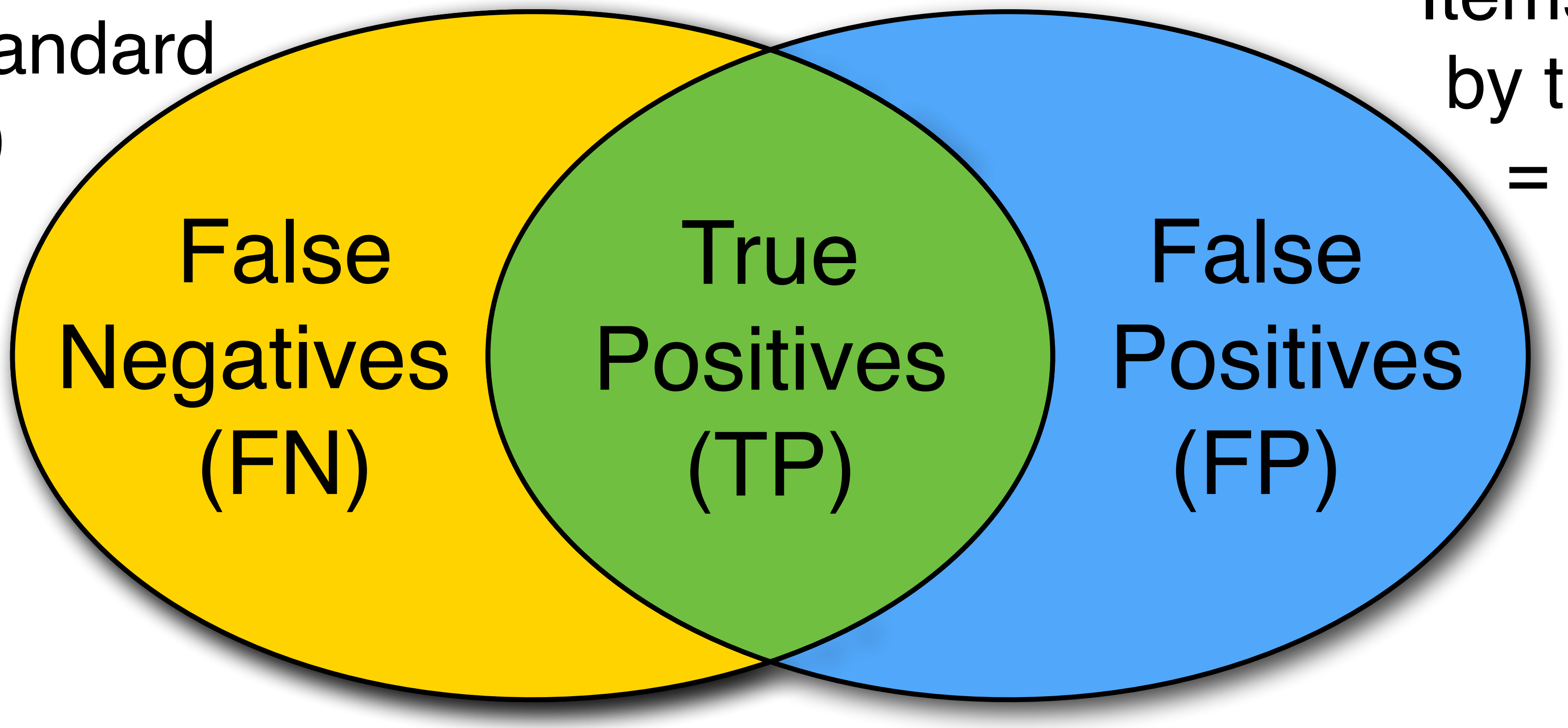


- **True positives**: Items that were labeled X by the system, and should be labeled X.
- **False positives**: Items that were labeled X by the system, but should *not* be labeled X.
- **False negatives**: Items that were *not* labeled X by the system, but should be labeled X,

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

A confusion matrix tabulates how many items that are labeled with class y in the gold data are labeled with class y' by the classifier.

		gold labels		
		urgent	normal	spam
system output	urgent	8	10	1
	normal	5	60	50
	spam	3	30	200

Confusion Matrices

This can be useful for understanding what kinds of mistakes a (multi-class) classifier makes

		gold labels		
		urgent	normal	spam
system output	urgent	8	10	1
	normal	5	60	50
	spam	3	30	200

Confusion Matrices

This can be useful for understanding what kinds of mistakes a (multi-class) classifier makes

system output		*gold labels* urgent	normal	spam
	urgent	8	10	1
	normal	5	60	50
	spam	3	30	200

Only 8/16 ‘urgent’ messages are classified correctly.

Confusion Matrices

This can be useful for understanding what kinds of mistakes a (multi-class) classifier makes

system output \ *gold labels*	urgent	normal	spam
urgent	8	10	1
normal	5	60	50
spam	3	30	200

Only 8/16 ‘urgent’ messages are classified correctly.
But 200/251 ’spam’ messages are classified correctly.

Confusion Matrices

This can be useful for understanding what kinds of mistakes a (multi-class) classifier makes

gold labels

		urgent	normal	spam
system output	urgent	8	10	1
	normal	5	60	50
	spam	3	30	200

Only 8/16 ‘urgent’ messages are classified correctly.
But 200/251 ’spam’ messages are classified correctly.
And only 8/19 messages labeled ‘urgent’ are actually urgent

Reading off Precision and Recall

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

Reading off Precision and Recall

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

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

Macro-average vs Micro-average

How do we aggregate precision and recall across classes?

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

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

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

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

Macro-average vs **Micro-average**

How do we aggregate precision and recall across classes?

Class 1: Urgent

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

Class 2: Normal

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

Class 3: Spam

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

Pooled

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

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

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

Macro-average vs. Micro-average

Which average should you report?

Macro-average (average P/R of all classes):
Useful if performance on all *classes*
is equally important.

Micro-average (average P/R of all items):
Useful if performance on all *items*
is equally important.

The End

Lecture 04, Part 5: Features for Sentiment Analysis