Introduction to Machine Learning: Naïve Bayes, Linear and Logistic Regression

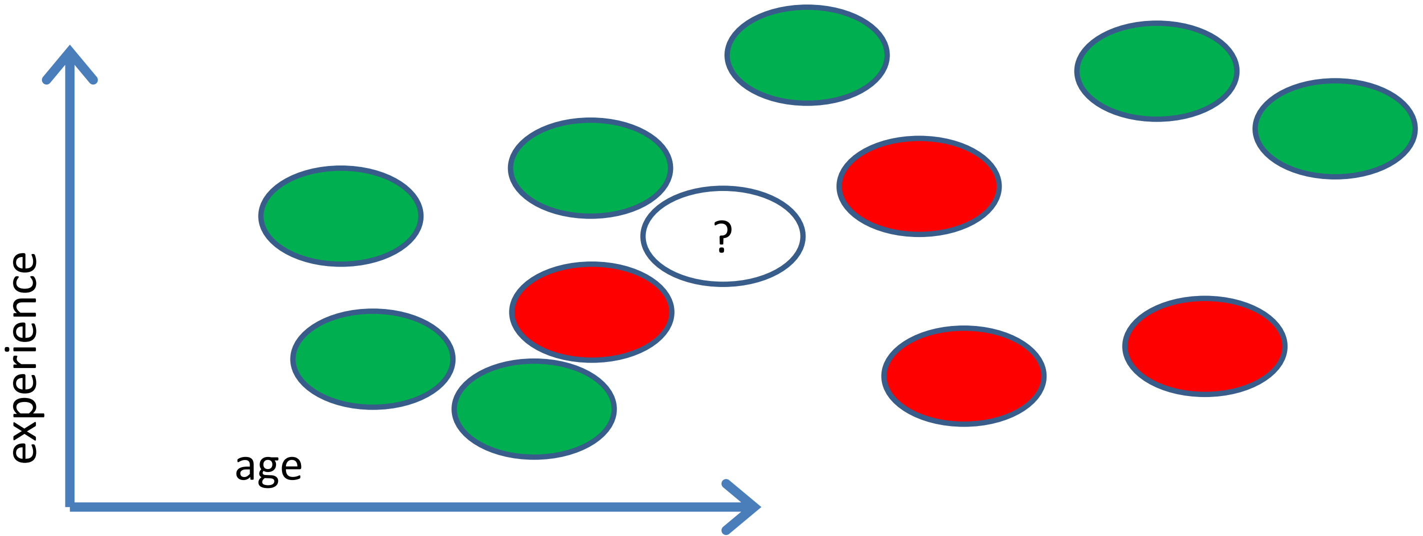
Amos Azaria

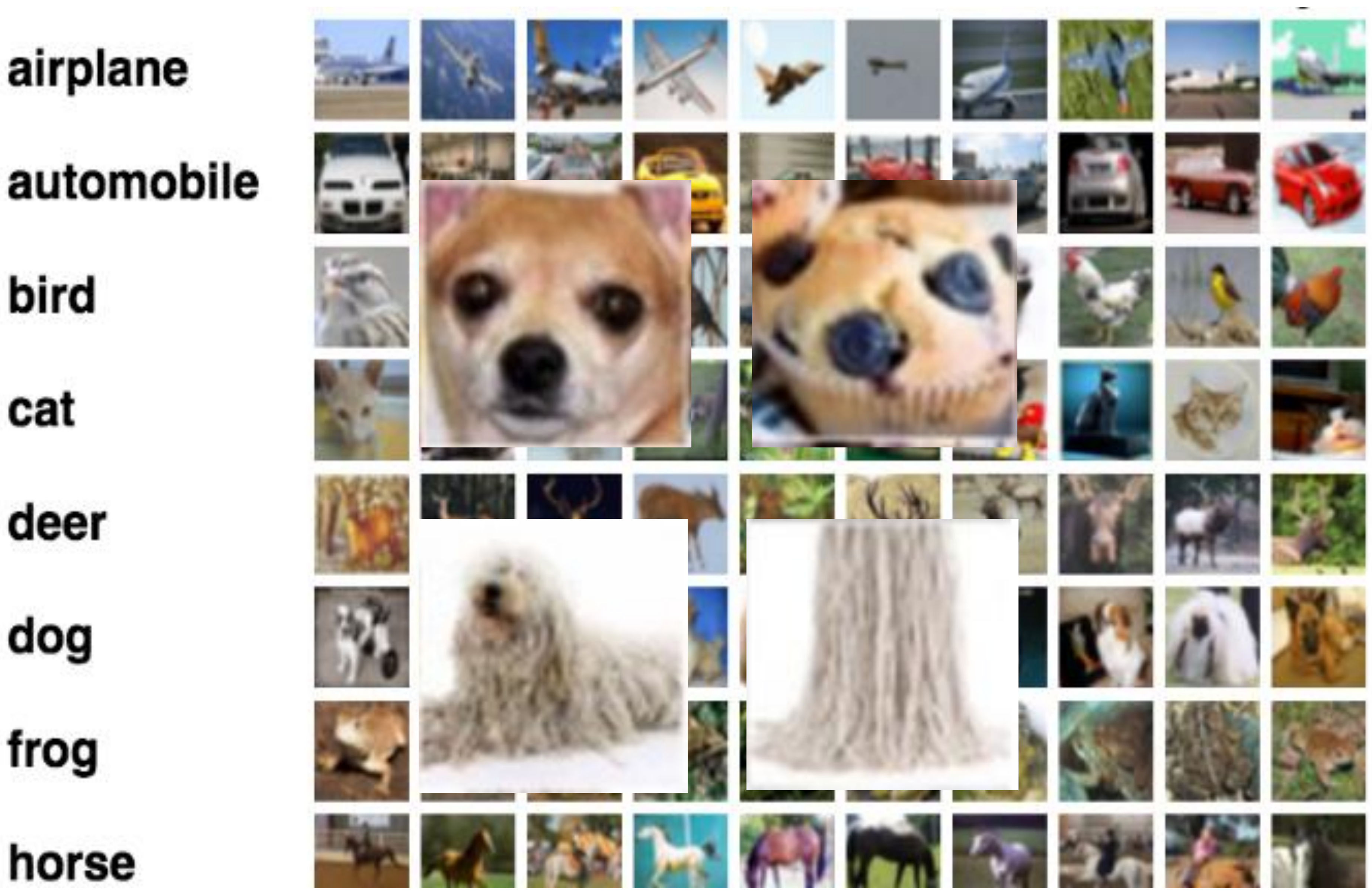
Databases and Machine Learning

- So far, we have dealt with retrieving data that was previously stored.
- Many times, we want to retrieve missing data, that is, data that we do not have. What if we want to know what will happen on new, unobserved data?
- For example, suppose some of our users are tagged as employed or unemployed, and our system needs to decide whether to show an ad which is relevant only to unemployed users.

Machine Learning (cont.)

- Assume we are given a training set (e.g. data on people and an indicator whether they are employed or not), and then we are given a new example (e.g. a new person), and we want to predict a value (e.g. employment).




airplane
automobile
bird
cat
deer
dog
frog
horse

Video…

https://www.youtube.com/watch?v=1eBxt9HUfh8

Machine Learning definition

- “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.” (Tom Mitchell)

Building a Machine Learning Classifier

- To solve the employment problem (or similar problems), we build a *machine learning model* (called a classifier) that, when given a new instance (a user), predicts a label (whether she is employed or not).
- There are many machine learning classification algorithms:
  - Naïve Bayes
  - Logistic regression
  - Neural networks
  - SVM
  - Decision Trees
  - KNN

Train / Test Split

- In order to *train* our classifier, we must have some *labeled data* (for example, we must know for some users whether they are employed or not).
- In order to evaluate the performance of our classifier, we split our data into *train and test sets*, train our classifier on the train set, and then test it on the test set.

Naïve Bayes

Amos Azaria

Naïve Bayes

- Naïve Bayes is a classifier that is based on the Bayes Rule.
- Naïve Bayes considers only the dependency of the *features* on the *category* (that is why it is called Naïve).
- Naïve Bayes can work also with missing features.

Spam Classifier

Spam:

- Buy it, pay later! Click me!
- You Won 10000 Dollars! Click here!

Real (Non-Spam):

- Will See you later.
- Will you want to meet later?
- I'm waiting for you.

Test:

- Are you paying too much? Click now!

Spam Classifier

Spam:

- Buy it, pay later! Click me!
- You Won 10000 Dollars! Click here!

Real (Non-Spam):

- Will See you later.
- Will you want to meet later?
- I'm waiting for you.

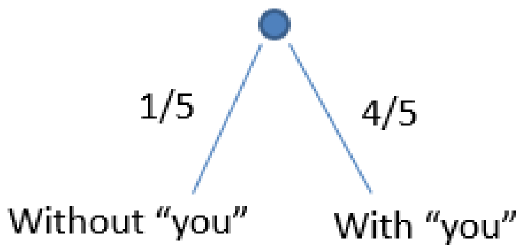
Test:

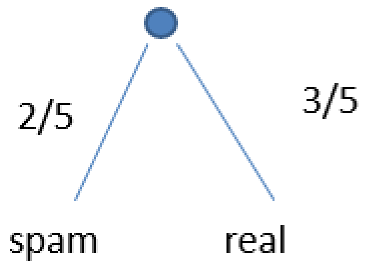
- Are you paying too much? Click now!

	Examples	are	you	paying	too	much	?	click	now	!
Spam	**2**	0	1	1	0	0	0	2	0	2
Real	**3**	0	3	0	0	0	1	0	0	0

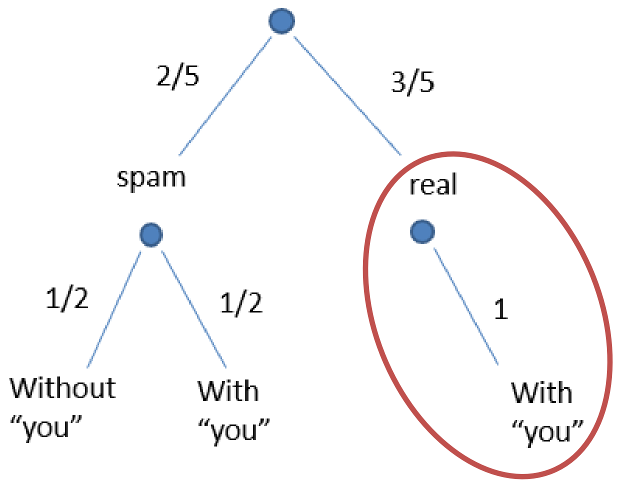
Conditional Probability

	Examples	are	you	paying	too	much	?	click	now	!
Spam	**2**	0	1	1	0	0	0	2	0	2
Real	**3**	0	3	0	0	0	1	0	0	0



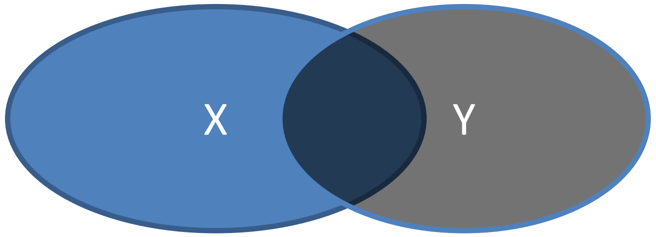


$$P(with\ "you"|real) = ?$$



Bayes Rule

$$p(X, Y) = p(X)p(Y \mid X)$$



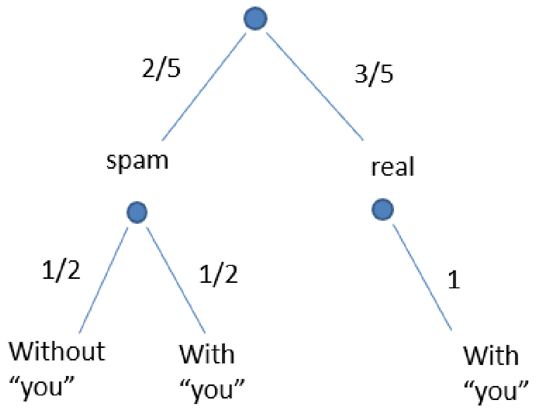
$$p(X)p(Y \mid X) = p(X, Y) = p(Y)p(X \mid Y)$$

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

This is an important formula which is always true!

Bayes Rule (one feature demo)

	Examples	are	you	paying	too	much	?	click	now	!
Spam	**2**	0	1	1	0	0	0	2	0	2
Real	**3**	0	3	0	0	0	1	0	0	0



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

- p(spam|with "you") $= \dfrac{P(sapm)\times P(with\ "you"|sapm)}{P(with\ "you")} = \dfrac{\frac{2}{5}\times\frac{1}{2}}{\frac{4}{5}} = \dfrac{1}{4}$
- p(real|with "you") $= \dfrac{P(real)\times P(with\ "you"\ |real)}{P(with\ "you")} = \dfrac{\frac{3}{5}\times\frac{1}{1}}{\frac{4}{5}} = \dfrac{3}{4}$

Bayes Rule (multi features demo)

	Examples	are	you	paying	too	much	?	click	now	!
Spam	**2**	0	1	1	0	0	0	2	0	2
Real	**3**	0	3	0	0	0	1	0	0	0

- p(spam | with “are” +“you” +”paying“ +”too”+”much”) = $\frac{P(sapm)\times P(full\ sentence\ |sapm)}{P(full\ sentence)} = ?$
- p(real | with “are” +“you” +”paying“ +”too”+”much”) = $\frac{P(real)\times P(full\ sentence\ |real)}{P(full\ sentence)} = ?$

Counting Messages / Documents

$$y^* = \underset{k \in \{1,..,K\}}{\operatorname{argmax}} \; p(y=k) \prod_{i=1}^{n} p(x_i \mid y=k)$$

If we perform stemming (or lemmatization): paying=pay

	Examples	are	you	paying	too	much	?	click	now	!
Spam	**2**	0	1	1	0	0	0	2	0	2
Real	**3**	0	3	0	0	0	1	0	0	0

K - number of classes (2 in our demo: {spam,real})
n - represents the number of words in our query
Xi - the word i in our query {are,you etc.}

x_2="you"

$p(x_2 \mid y = real) = \frac{3}{3} = 1$

x_1="are"

$p(x_1 \mid y = real) = \frac{0}{3} = 0$

?

Spam:
- Buy it, pay later! Click me!
- You Won 10000 Dollars! Click here!

Real (Non-Spam):
- Will See you later.
- Will you want to meet later?
- I'm waiting for you.

Test:
- Are you paying too much? Click now!

Laplace smoothing

$$y^* = \underset{k \in \{1,..,K\}}{\operatorname{argmax}} \; p(y=k) \prod_{i=1}^{n} p(x_i \mid y=k)$$

Most of the time, alpha = 1 is being used to remove the problem of zero probability.

$$p_{i,empirical} = \frac{X_i}{N} \quad \Rightarrow \quad p_{i,\alpha-smoothed} = \frac{X_i + \alpha}{N + \alpha K}$$

x_2="you"

$$p(x_2 \mid y = real) = \frac{3}{3} = 1 \quad \Rightarrow \quad p(x_2 \mid y = real) = \frac{4}{5}$$

x_2="you"

x_1="are"

$$p(x_1 \mid y = real) = \frac{0}{3} = 0 \quad \Rightarrow \quad p(x_1 \mid y = real) = \frac{1}{5}$$

x_1="are"

Counting Messages / Documents

$$y^* = \underset{k\in\{1,..,K\}}{\operatorname{argmax}} \; p(y=k) \prod_{i=1}^{n} p(x_i \mid y=k)$$

	Examples	are	you	paying	too	much	?	click	now	!
Spam	**3**	1	2	2	1	1	1	3	1	3
Real	**4**	1	4	1	1	1	2	1	1	1

- Laplace's Smoothing

$$p(y=0|x) = \frac{3}{7} \cdot \frac{1}{4}\frac{2}{4}\frac{2}{4}\frac{1}{4}\frac{1}{4}\frac{1}{4}\frac{3}{4}\frac{1}{4}\frac{3}{4} = 5.87 \cdot 10^{-5}$$

$$p(y=1|x) = \frac{4}{7} \cdot \frac{1}{5}\frac{4}{5}\frac{1}{5}\frac{1}{5}\frac{1}{5}\frac{2}{5}\frac{1}{5}\frac{1}{5}\frac{1}{5} = 2.34 \cdot 10^{-6}$$

- For smoothing, we may consider the vocabulary length and length of every message.
- In practice, we many times maximize the log (so we don't need to deal with such small numbers).
- Note that when using the log, we sum rather than multiply.

Naïve Bayes (Why? How?)

- Notation:
  - The instances (messages in our case) are denoted by: $x_1, x_2, ..., x_m$
    - m in our case is 5
    - (so x_1= Buy it, pay later! Click me!)
    - Note that each of the xs is a vector: $x_t = x_{t1}, x_{t2}, ..., x_{tn}$
  - The classes (Spam/Real in our case) are denoted by: $y_1, y_2, ..., y_m$
    - (so y_1=spam)

Naïve Bayes (Cont.)

- Suppose we have a new message x_t (a vector) and we want to find the most likely class (y), that is, we want to know if x_t is more likely to present class 0 (e.g. Spam) or class 1 (not-spam), etc.
- For that we want to compute for every k: $p(y=k|x_t)$
- We then pick the *k* that maximizes the above.
- Denote that *k* as y^*.

Naïve Bayes

- Naïve Bayes makes the conditionally independent assumption:

Always true $$p(x_t, y_t) = p(y_t)p(x_{t1} \mid y_t)p(x_{t2} \mid y_t, x_{t1})p(x_{t3} \mid y_t, x_{t1}, x_{t2}) \cdots$$

Naïve Bayes assumption $$p(x_t, y_t) = p(y_t)p(x_{t1} \mid y_t)p(x_{t2} \mid y_t)p(x_{t3} \mid y_t) \cdots$$

$$p(x_t, y_t) = p(y_t) \prod_{i=1}^{n} p(x_{ti} \mid y_t)$$

Bayes Rule $$p(y = k \mid x_t) = \frac{p(y=k) \prod_{i=1}^{n} p(x_{ti} | y=k)}{p(x_t)}$$

Independent of K (so we can ignore when looking for the maximum)

$$y^* = \underset{k \in \{1,..,K\}}{\operatorname{argmax}} \, p(y = k) \prod_{i=1}^{n} p(x_{ti} \mid y = k)$$

Naïve Bayes

- Naïve Bayes makes the conditionally independent assumption:

Always true

$$p(x_t, y_t) = p(y_t)p(x_{t1} \mid y_t)p(x_{t2} \mid y_t, \cancel{x_{t1}})p(x_{t3} \mid y_t, \cancel{x_{t1}}, \cancel{x_{t2}}) \cdots$$

$$\Downarrow$$

Naïve Bayes assumption

$$p(x_t, y_t) = p(y_t)p(x_{t1} \mid y_t)p(x_{t2} \mid y_t)p(x_{t3} \mid y_t) \cdots$$

$$p(x_t, y_t) = p(y_t) \prod_{i=1}^{n} p(x_{ti} \mid y_t)$$

Bayes Rule

$$p(y = k \mid x_t) = \frac{p(y=k) \prod_{i=1}^{n} p(x_{ti}|y=k)}{p(x_t)}$$

Independent of K (so we can ignore when looking for the maximum)

$$\boxed{y^* = \operatorname*{argmax}_{k \in \{1,..,K\}} p(y = k) \prod_{i=1}^{n} p(x_{ti} \mid y = k)}$$

Naïve Bayes (Cont.)

- Note that the calculations shown previously (the fractions of the appearances of every word) aren't an immediate corollary of the formula we derived.
- In general, we try to maximize the likelihood of the data: $\prod_{i=0}^{N} p(x_i, y_i)$
- It turns out (and easy to show), that if we use the fraction of messages each word appears in (as we have done), we in-fact maximize the likelihood of the data.
- We end up with:

$$y^* = \underset{k \in \{1,..,K\}}{\text{argmax}}\ p(y=k) \prod_{i=1}^{n} \frac{\text{num of messages in class k with word i}}{\text{num of messages in class k}}$$

Multiple predictions

- How would we prepare the data for multiple queries?
  - Determine how many documents we have in total. Denote this value by p_{tot}.
  - We first count how many documents every class has. Denote these values by: p_k.
  - [We then find all words that appear in all documents and form a dictionary.]
  - For each word in the dictionary (i) and for every class (k), we need to count how many documents it appears in. Denote these values by p_{ki}.
- For every new query we simply compute:

$$y^* = \operatorname*{argmax}_{k \in \{1,..,K\}} \frac{p_k}{p_{tot}} \prod_{i=1}^{n} \frac{p_{ki}}{p_k}$$

Online Naïve Bayes: Parallel Execution (In Spark)

- Suppose the data is stored in an RDD as (message, class) tuples. E.g. (greeting / valediction):

```
input_data = sc.parallelize([("hello there", 0), ("hi there", 0), ("go home", 1),
("see you",1), ("goodbye to you", 1), (“bye bye”, 1)])
```

- We need to find p_{tot}, p_k and all p_{ki}:

```
>>> pk = input_data.map(lambda tup: (tup[1], 1)) \
.reduceByKey(lambda a,b: a+b).collectAsMap()
```

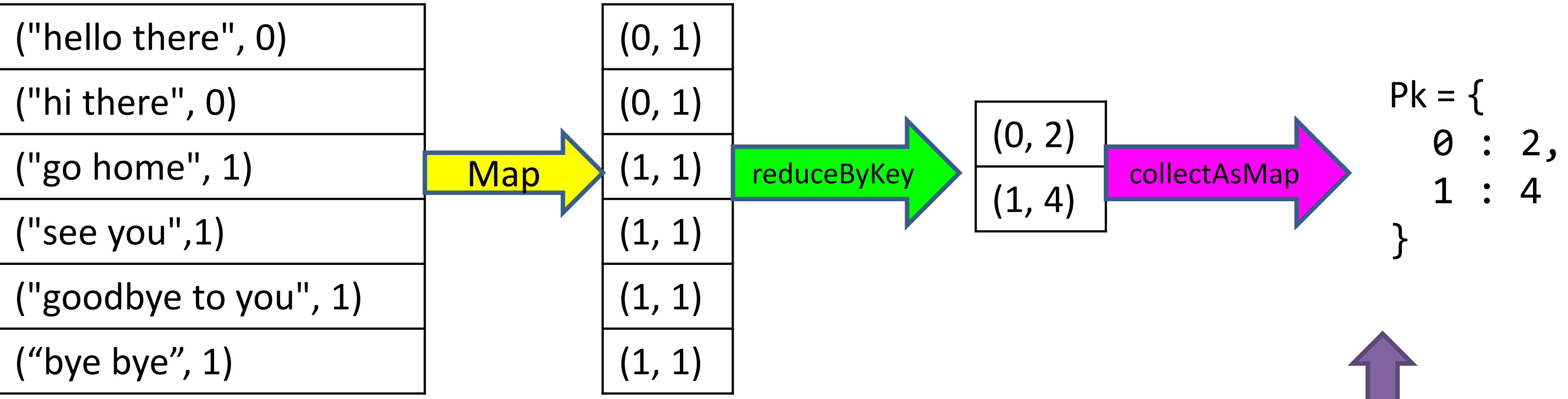
```
>>> ptot = sum(pk.values())
```

```
>>> pki = input_data \
.flatMap(lambda tup: list(set([(tup[1],w) for w in tup[0].split()]))) \
.map(lambda tup: (tup, 1)) \
.reduceByKey(lambda a,b: a+b).collectAsMap()
```

Pk example

```
input_data = sc.parallelize([("hello there", 0), ("hi there", 0), ("go
home", 1), ("see you",1), ("goodbye to you", 1), (“bye bye”, 1)])
```

```
>>> pk = input_data.map(lambda tup: (tup[1], 1)) \
         .reduceByKey(lambda a,b: a+b).collectAsMap()
```



back

Ptot example

```
>>> ptot = sum(pk.values())
```

```
Pk = {
   0 : 2,
   1 : 4
}
```

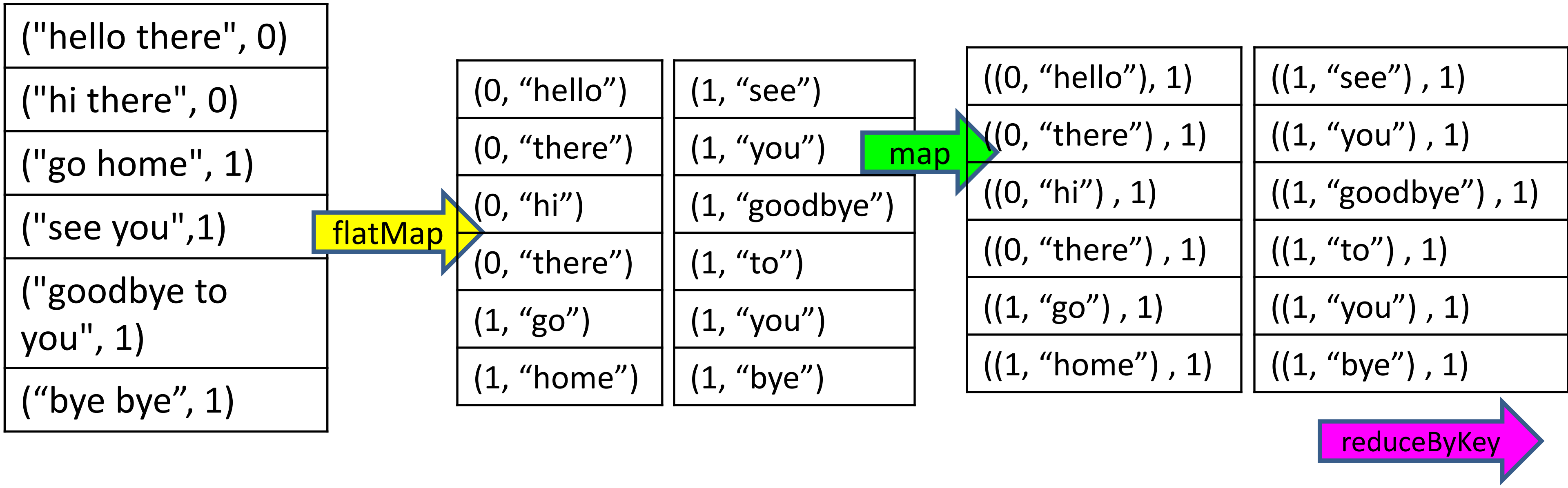
$\xrightarrow{\text{sum}}$ Ptot = 6

back

Pki example

```
>>> pki = input_data\
    .flatMap(lambda tup: list(set([(tup[1],w) for w in tup[0].split()]))) \
    .map(lambda tup: (tup, 1)) \
    .reduceByKey(lambda a,b: a+b).collectAsMap()
```

```
input_data = sc.parallelize([("hello there", 0), ("hi there", 0), ("go
home", 1), ("see you",1), ("goodbye to you", 1), (“bye bye”)])
```



```
("hello there", 0)
("hi there", 0)
("go home", 1)
("see you",1)
("goodbye to
you", 1)
(“bye bye”, 1)
```

```
(0, “hello”)      (1, “see”)
(0, “there”)      (1, “you”)
(0, “hi”)         (1, “goodbye”)
(0, “there”)      (1, “to”)
(1, “go”)         (1, “you”)
(1, “home”)       (1, “bye”)
```

```
((0, “hello”), 1)       ((1, “see”) , 1)
((0, “there”) , 1)      ((1, “you”) , 1)
((0, “hi”) , 1)         ((1, “goodbye”) , 1)
((0, “there”) , 1)      ((1, “to”) , 1)
((1, “go”) , 1)         ((1, “you”) , 1)
((1, “home”) , 1)       ((1, “bye”) , 1)
```

Pki example - cont.

```
>>> pki = input_data \
    .flatMap(lambda tup: list(set([(tup[1],w) for w in tup[0].split()]))) \
    .map(lambda tup: (tup, 1)) \
    .reduceByKey(lambda a,b: a+b).collectAsMap()
```

```
input_data = sc.parallelize([("hello there", 0), ("hi there", 0), ("go
home", 1), ("see you",1), ("goodbye to you", 1), (“bye bye”)])
```

reduceByKey

((0, “hello”), 1)	((1, “see”) , 1)
((0, “there”) , 2)	((1, “you”) , 2)
((0, “hi”) , 1)	((1, “goodbye”) , 1)
((1, “go”) , 1)	((1, “to”) , 1)
((1, “home”) , 1)	((1, “bye”) , 1)

collectAsMap

```
Pki = {
    (0, “hello”)  :  1,
    (0, “there”)  :  2,
    (0, “hi”) : 1,
    (1, “go”) : 1,
    (1, “home”) : 1,
    (1, “see”) : 1,
    (1, “you”) : 2,
    (1, “goodbye” : 1,
    (1, “to”) : 1,
    (1, “bye”) : 1
}
```

back

Answering a new query

$$y^* = \underset{k \in \{1,..,K\}}{\operatorname{argmax}} \frac{p_k}{p_{tot}} \prod_{i=1}^{n} \frac{p_{ki}}{p_k}$$

```
>>> import numpy as np
```

Cast not required in Python 3

0 as default

Number of documents with word i in class k

Number of documents in class k

```
>>> query = "hello hi"

>>> class_probs = [pk[k]/float(ptot)*np.prod (np.array([pki.get((k,i),0)/float(pk[k])
for i in query.split()])) for k in range(0,2)]

>>> print(class_probs)
[0.10000000000000001, 0.0]
```

No Laplass smoothing, so we get zeros.

```
>>> y_star = np.argmax(np.array(class_probs))

>>> print(y_star)
0
```

It is indeed a greeting!

What is a good classifier?

- Accuracy? $\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Number Of Examples}}$
- What if we classify cancer with a 1% chance?
  - If we classify all as healthy, we will have 99% accuracy.
  - What would you think about a classifier that:
    - catches 98% of the cancers,
    - but has only 30% precision (if you are told that you might have cancer, there is only 30% that you actually have).

Recall and Precision

Confusion Matrix	Classified as Positive	Classified as Negative
Really Positive	True Positive	False Negative
Really Negative	False Positive	True Negative

- Accuracy?
  - Trues / All

$$\text{Accuracy} = \frac{\textit{True Positive} + \textit{True Negative}}{\textit{True Positive} + \textit{True Negative} + \textit{False Negative} + \textit{False Positive}}$$

- Recall
  - What fraction of positives did we actually find?
  - True Positive / Really Positive

$$\text{Recall} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Negative}}$$

- Precision:
  - If we say positive, how precise are we?
  - True Positive / Classified as Positive

$$\text{Precision} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Positive}}$$

F-Measure (F_1-Score, F-Score)

- A combination of precision and recall:
- 2 (Precision * Recall) / (Precision + Recall)
- E.g.:
  - Precision = 1, recall = 0,
    - F-Measure = 0
  - Precision = recall = 0.5
    - F-Measure = 0.5
  - Precision = recall = x
    - F-Measure = x