TFIDF and NGrams

Sameer Singh and **Conal Sathi**

BANA 290: ADVANCED DATA ANALYTICS

MACHINE LEARNING FOR TEXT

SPRING 2018

April 10, 2018

Upcoming...

Homework

- Homework 1 is due tonight!
- Make sure to focus on error analysis!
- Homework 2 will be out later this week

Recap from Previous Lecture

TEXT CLASSIFICATION

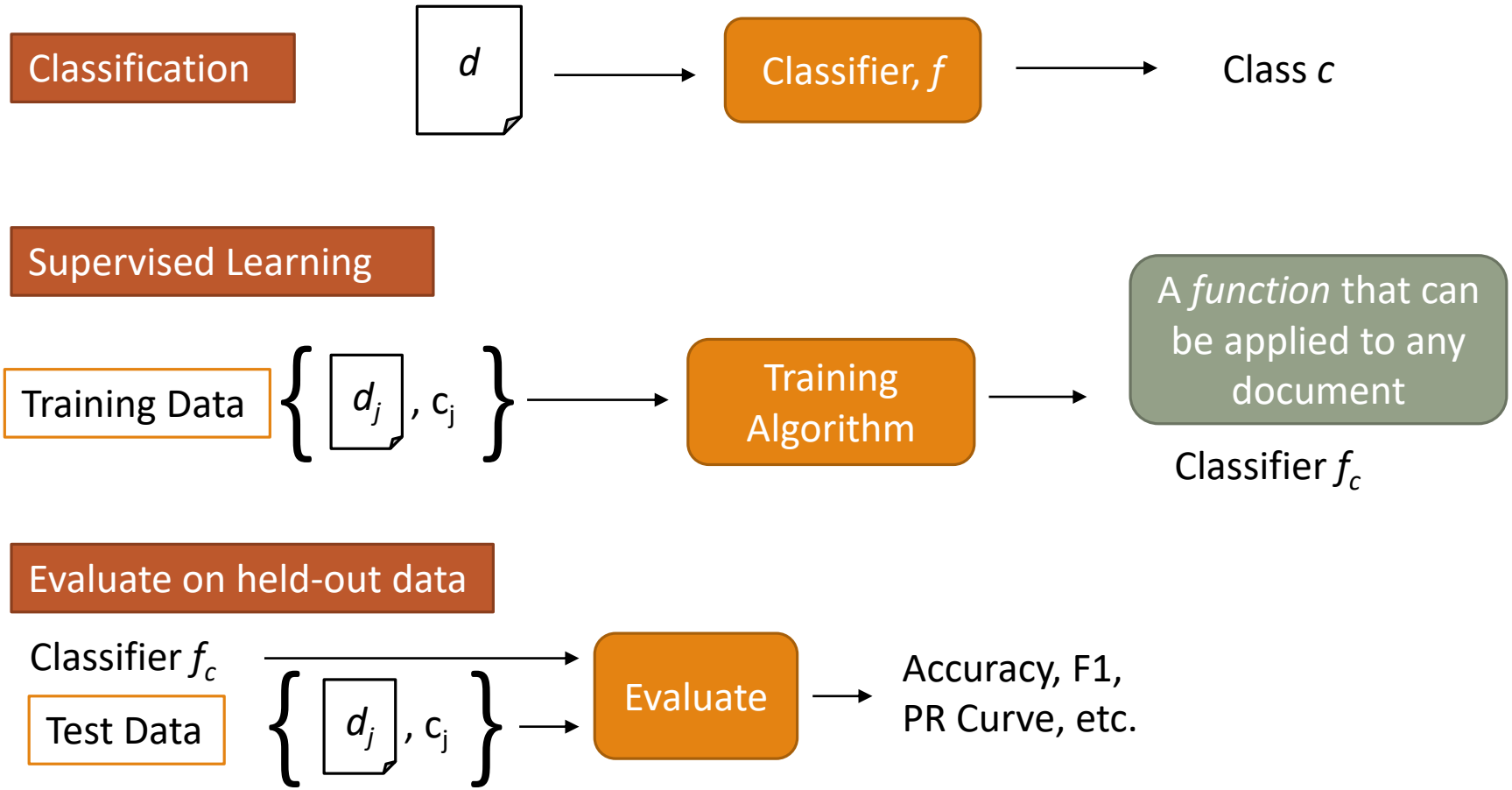
Text Classification: definition

Input:

- a document d
- a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$

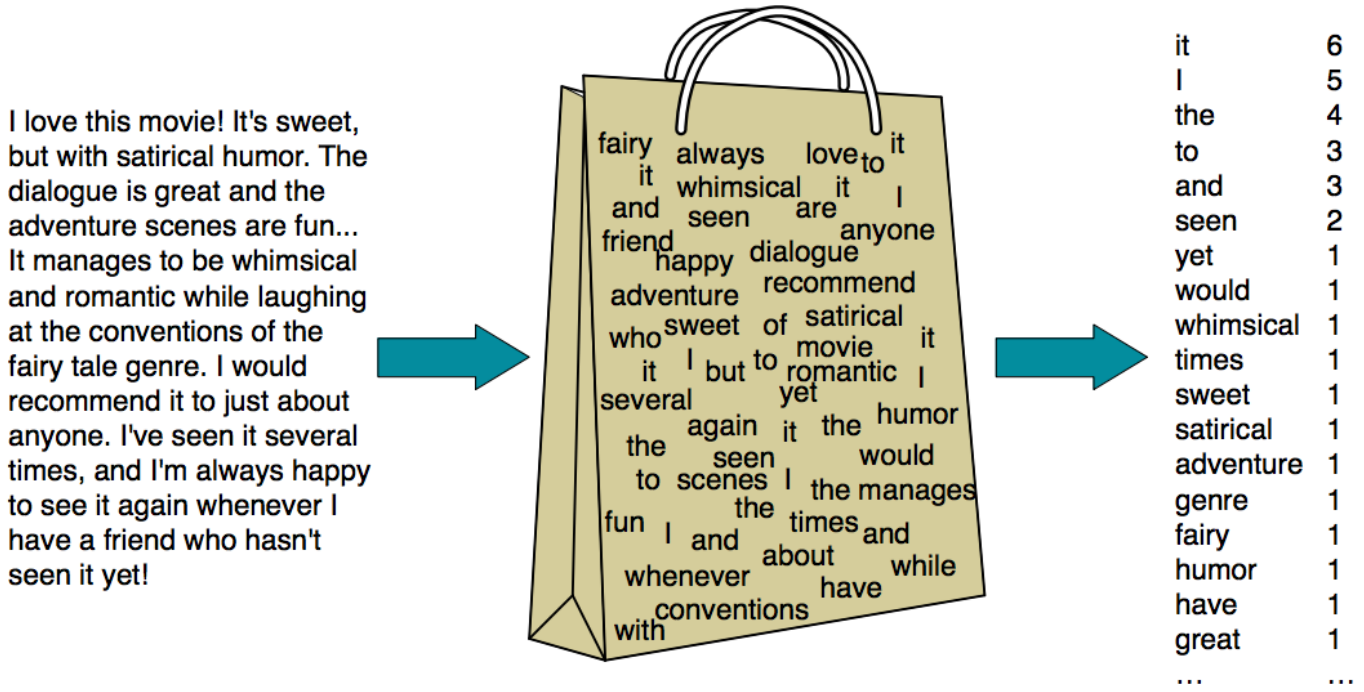
Output: a predicted class $c \in C$

Supervised Machine Learning



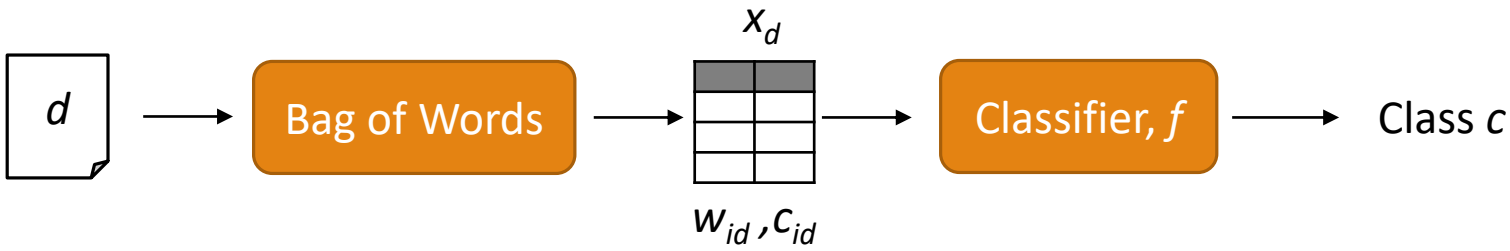
Bag of Words

- Word ordering does not matter (**Bag of Words**)



Simple Classifier: Naïve Bayes

Classification



$$f(x_d) = \text{argmax}_c P(c|x_d) = \text{argmax}_c P(c) \prod_{w_{id}, c_{id}} P(w_{id}|c)^{c_{id}}$$

Prob. of class for document

How common is class c?

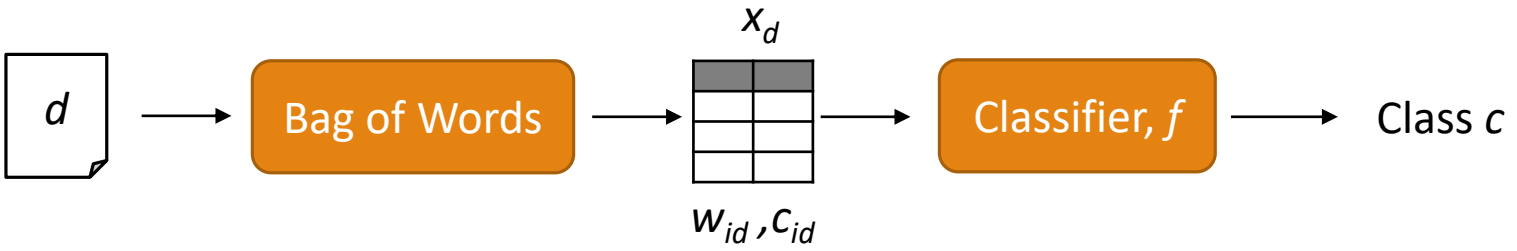
How common is word w_{id} for class c?

Features for Classification

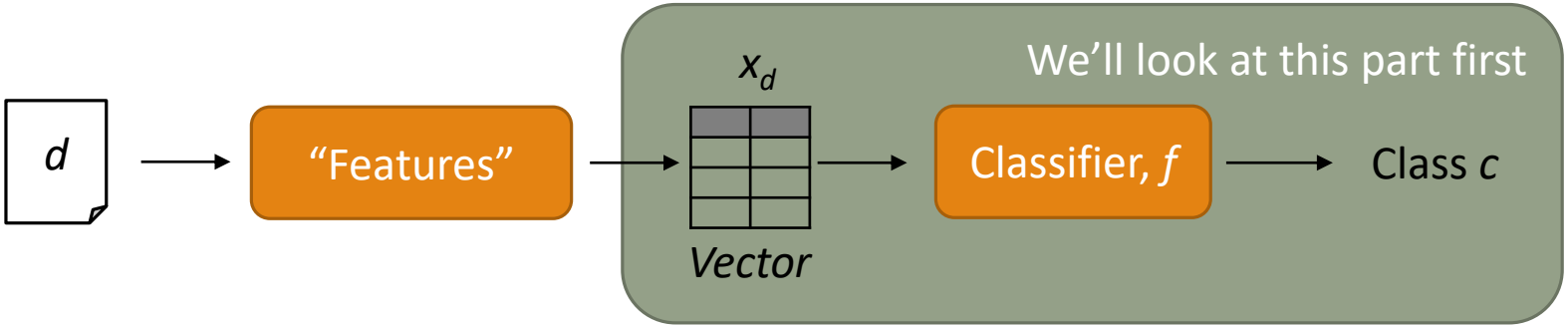
THINKING IN TERMS OF VECTORS

Using Vectors

Last lecture



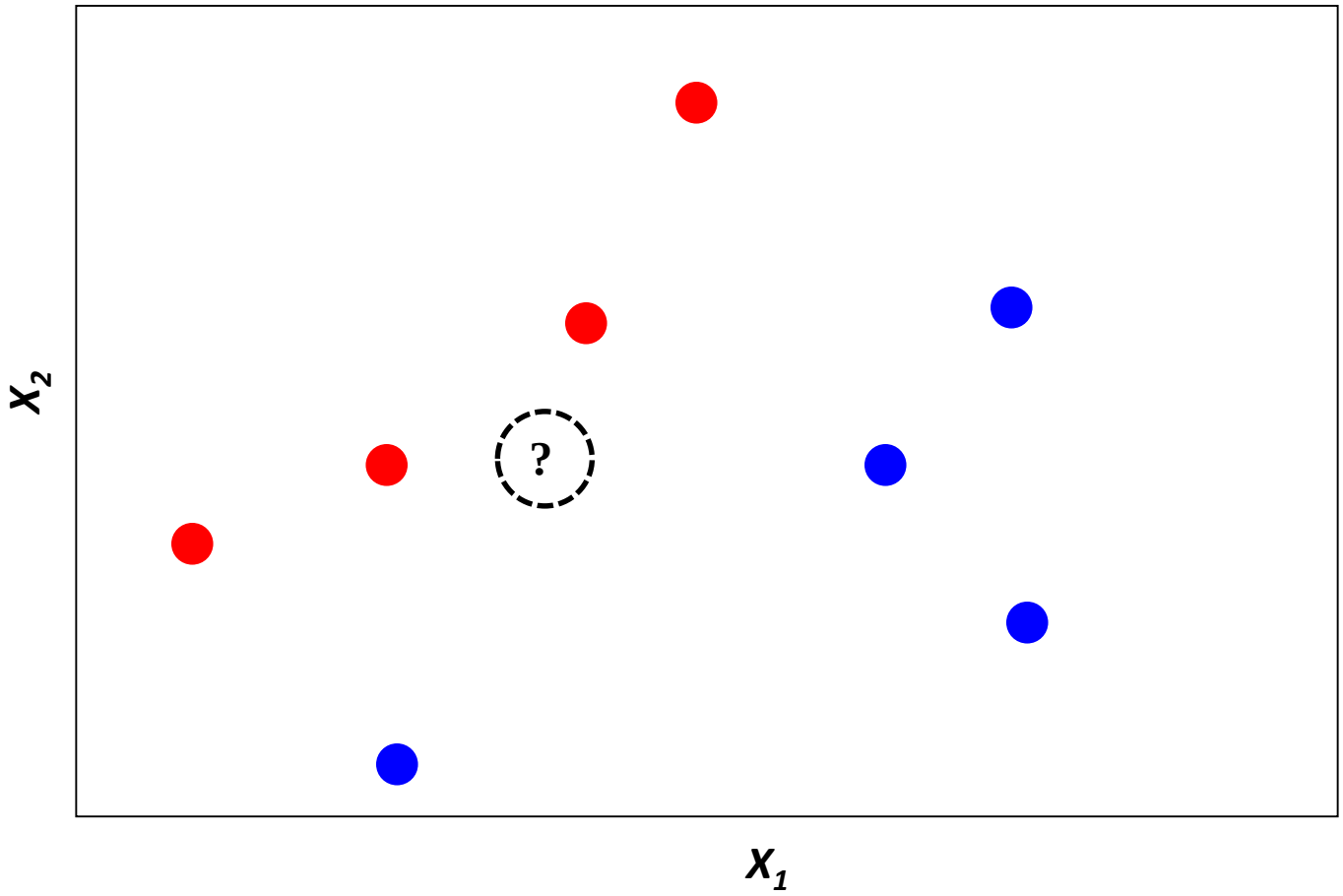
Machine Learning



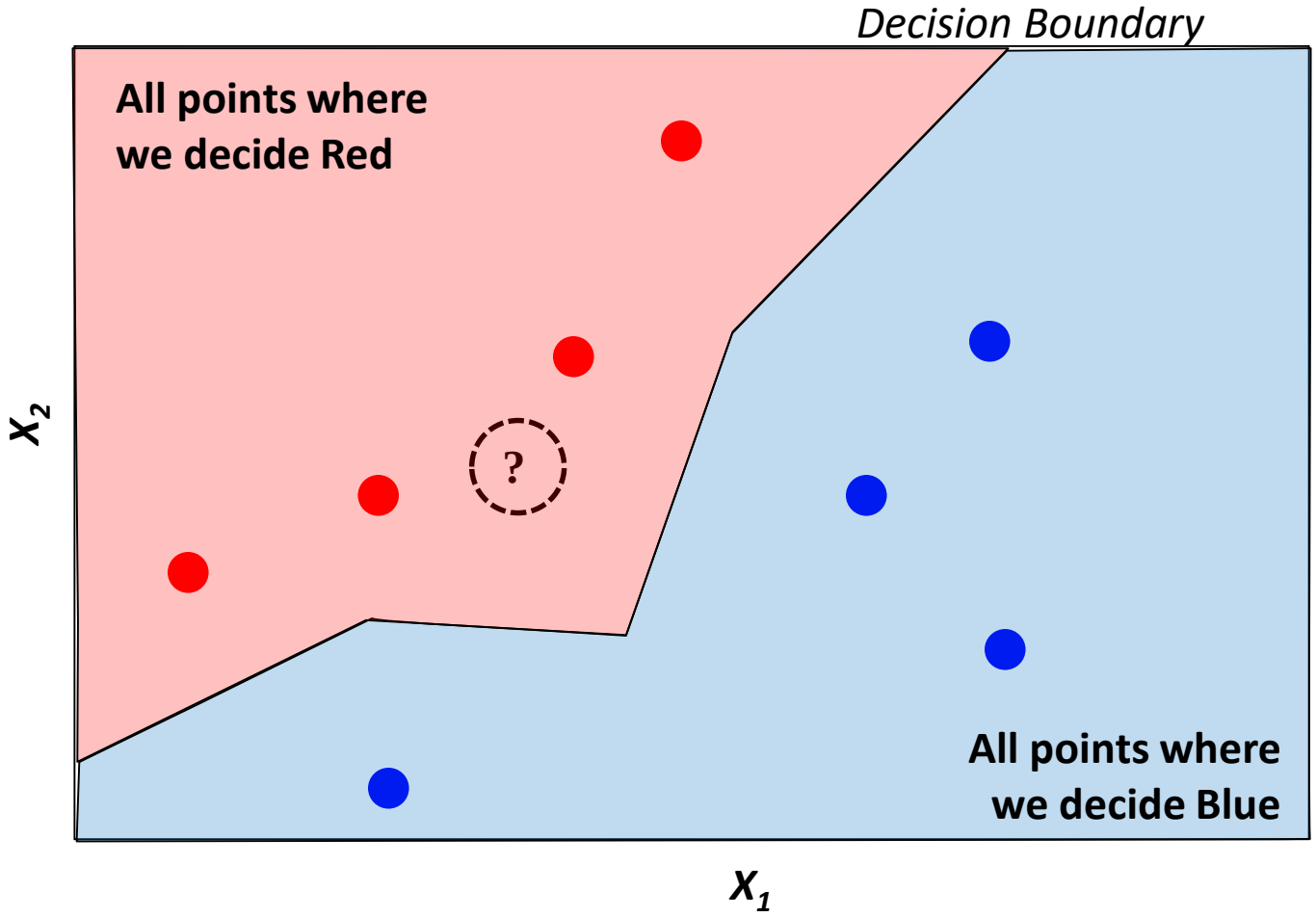

```
def classify(d):
    # implement code…
    return pred
```

```
def classify(x):
    # do some math…
    return pred
```

Classification



Nearest Neighbor Classification



K-Nearest Neighbor (kNN)

Find the k-nearest neighbors to *x* in the training data

- i.e., select the k vectors/documents that have smallest distance to x

Classification

- ranking yields k feature vectors and a set of k class labels
- pick the class label which is most common in this set ("vote")
- classify x as belonging to this class

"Training" is trivial

- store training data as a lookup table, and search to classify a new datum

K-Nearest Neighbor (kNN)

```
def train(data):
    # just save it!
    D = data
```

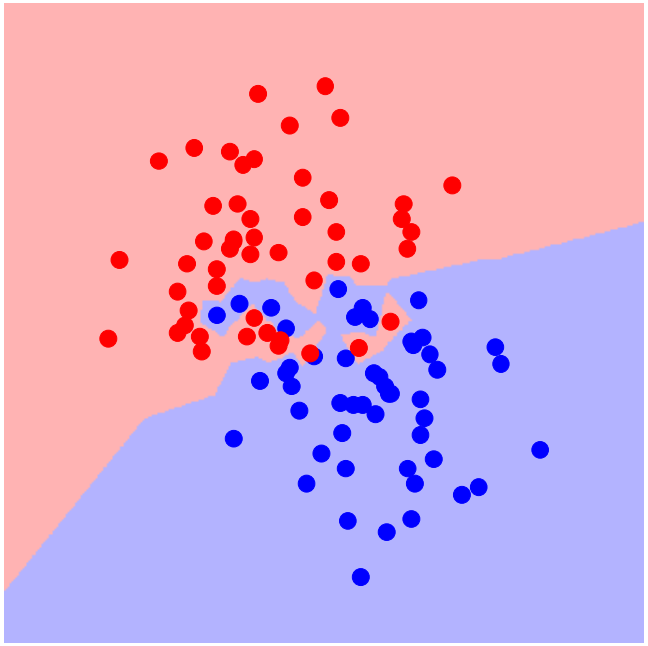
```
def distance(x1, x2): # we'll get back to this later

def classify(x):
    # find k-nearest neighbors of x
    neighs = heapq.nsmallest(k, D,
                key = lambda xt,yt: distance(x, xt))
    # get most common label
    labels = [y for (x,y) in neighs]
    return max(set(labels), key=labels.count)
```

kNN Decision Boundary

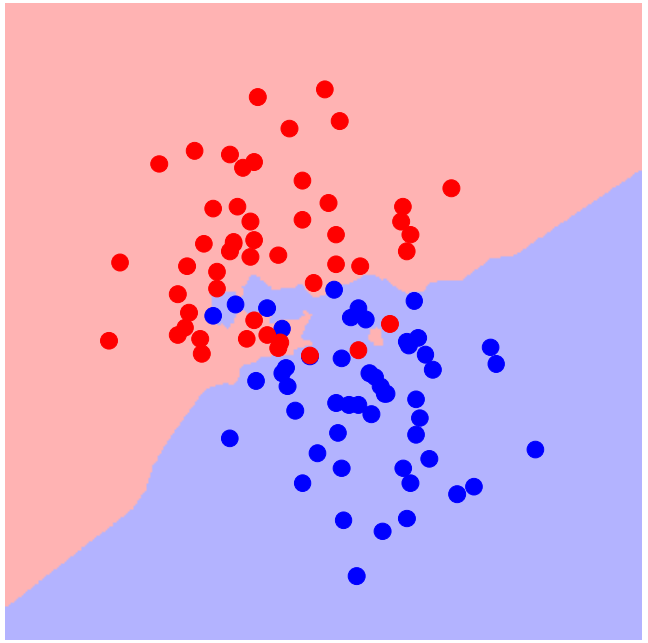
Increasing k "simplifies" decision boundary

- Majority voting means less emphasis on individual points

- K = 1



K = 3

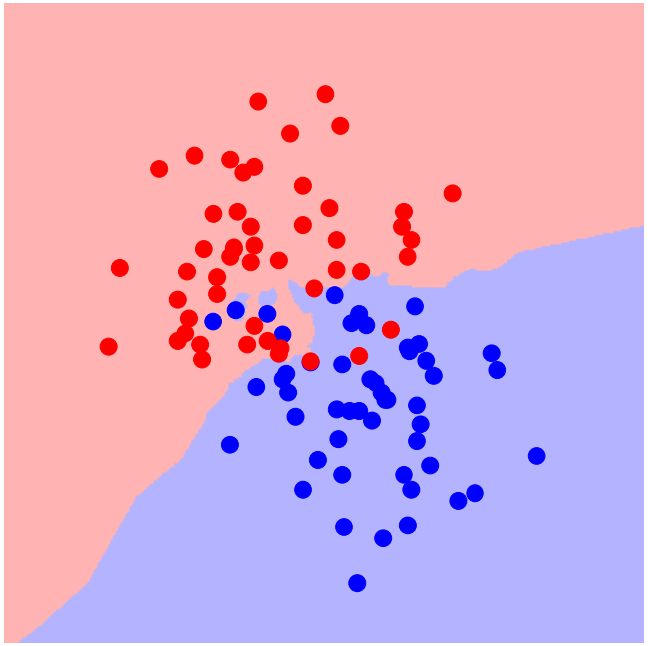


kNN Decision Boundary

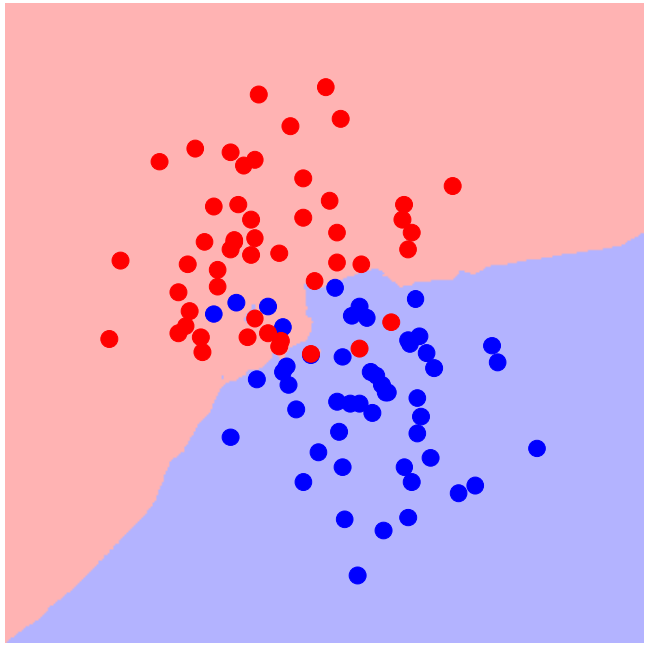
Increasing k "simplifies" decision boundary

- Majority voting means less emphasis on individual points

- K = 5



K = 7

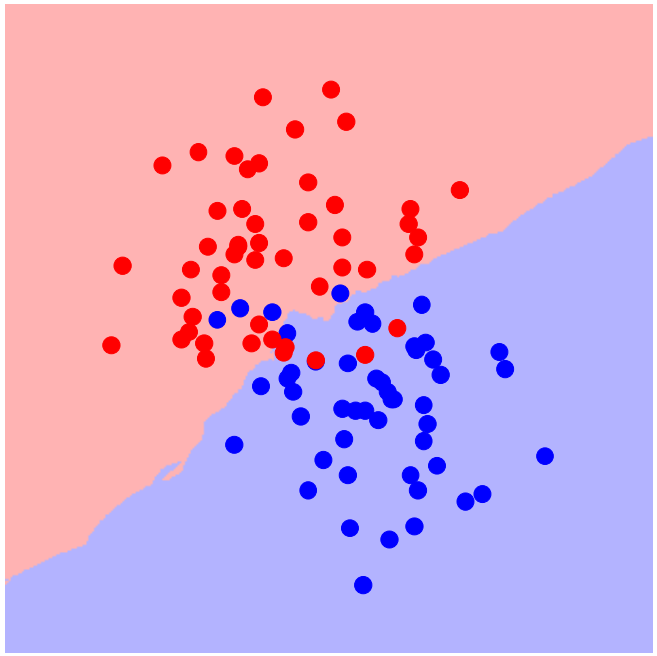


kNN Decision Boundary

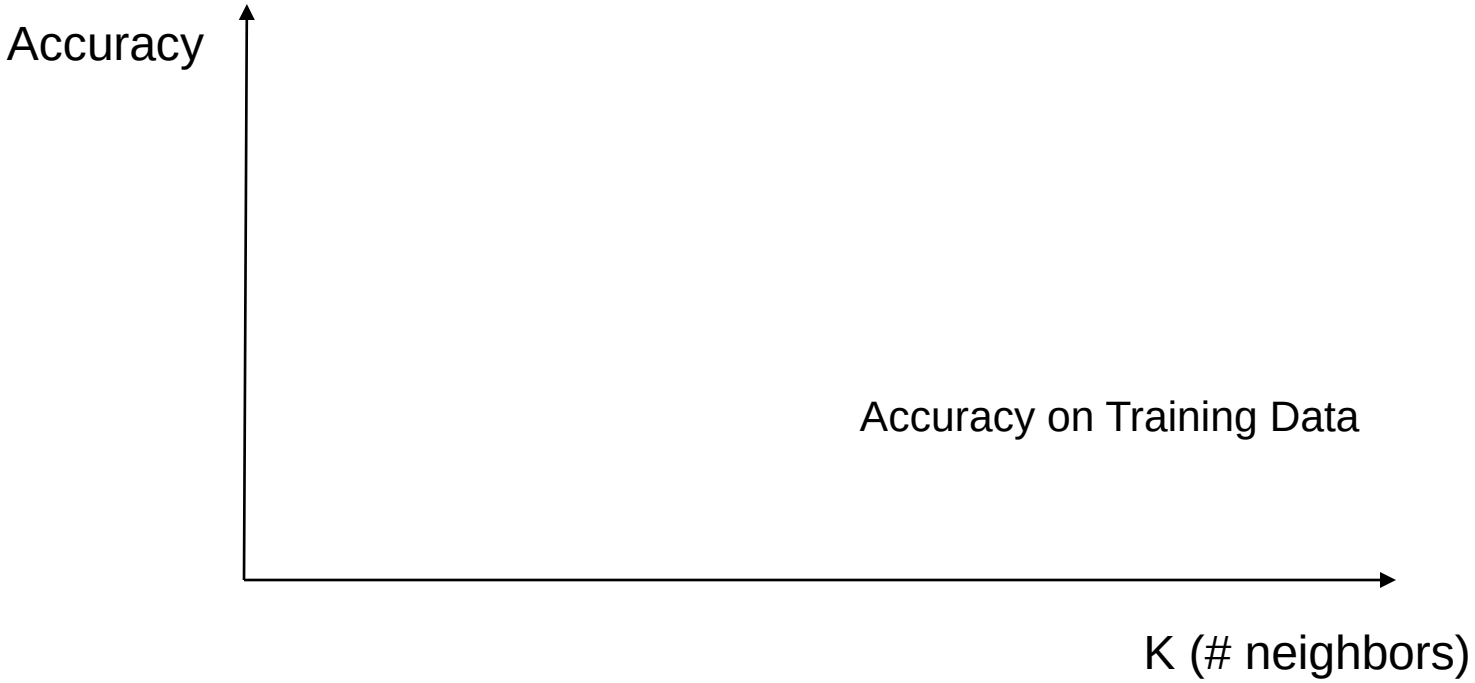
Increasing k "simplifies" decision boundary

- Majority voting means less emphasis on individual points

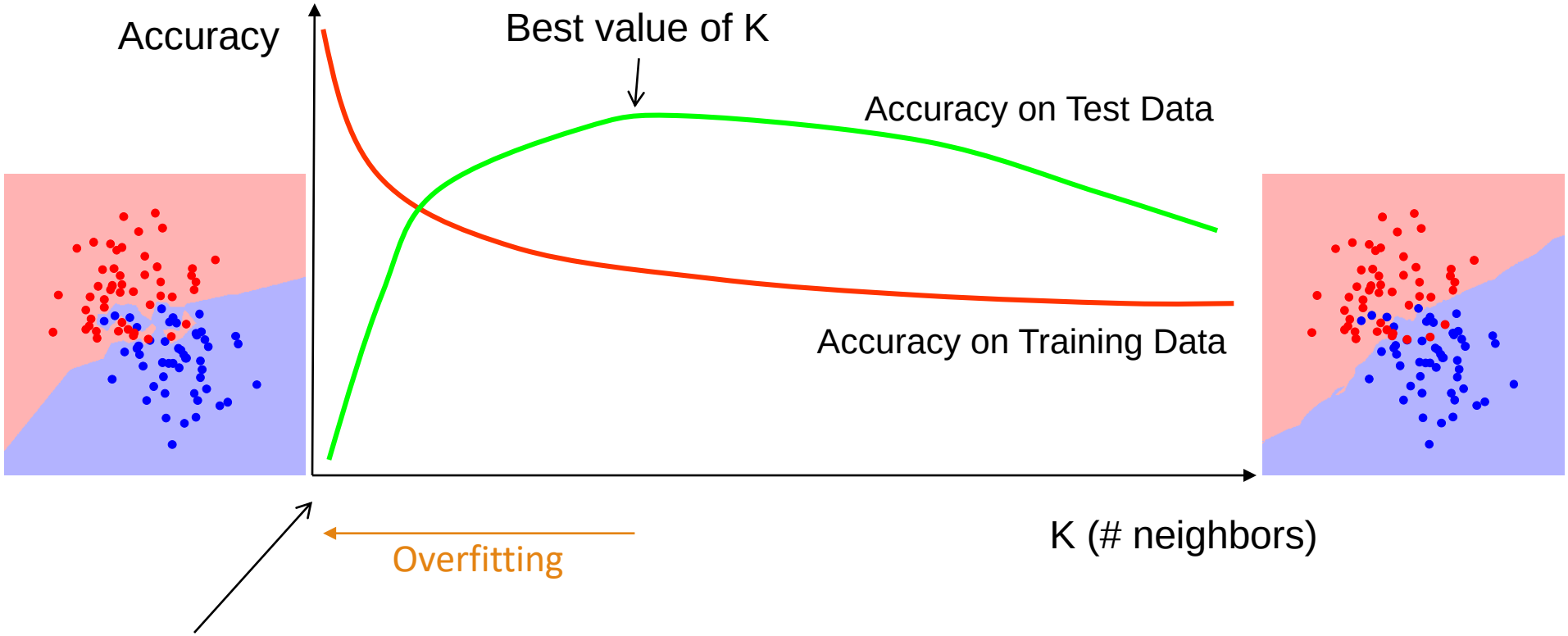
- K = 25



Accuracy and K

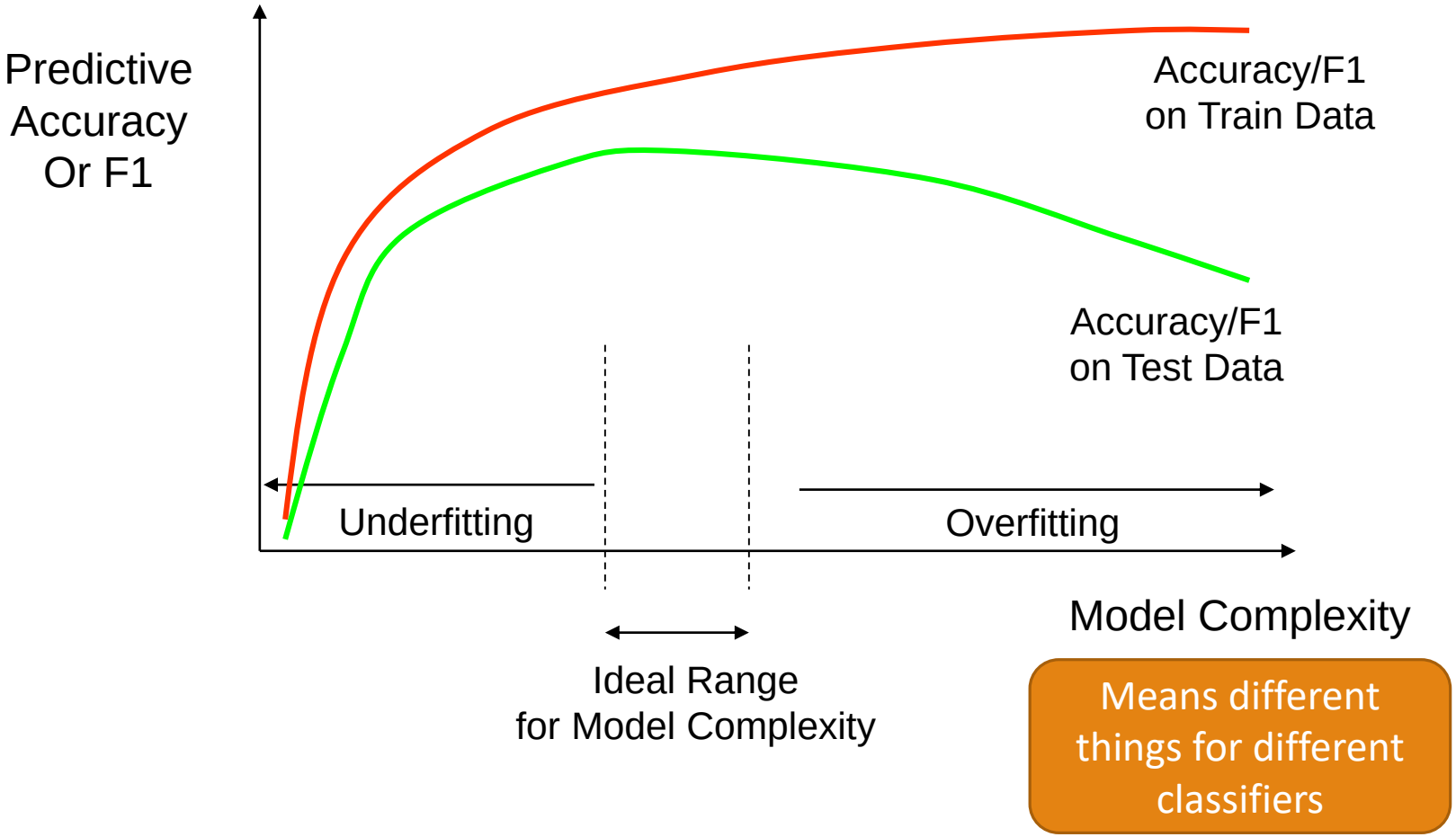


Accuracy and K



K=1? Perfect accuracy!
Training data has been memorized...

Reminder: In general…



How do we select K?

Or regularization strength? or count cut-offs? or threshold?

Training data?

- Clearly a bad choice, overfits!
- e.g. K will always be 1, regularization will be 0, etc.

Evaluation/Test data?

- Seems like a good choice, since it was held-out
- **But,** if we optimize on this data, we get an artificial boost
  - how do we know how it will perform in real-world?

For learning the *classifier*

For tuning hyper-*parameters*

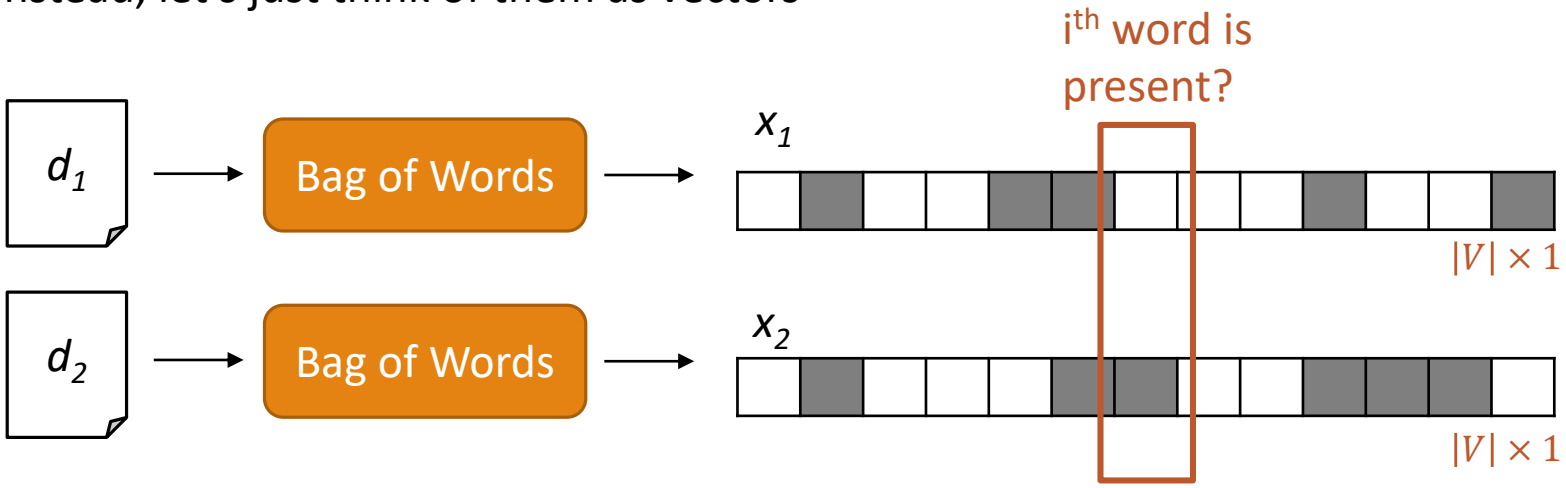
For final evaluation (often "hidden")

What about the distance?

```
def distance(x1, x2): # we'll get back to this later
```

What is the distance between two documents?

- Can be an incredibly difficult task!
  - Synonyms, similar words, paraphrases, etc.
- Instead, let's just think of them as vectors



Documents as vectors

So we have a |V|-dimensional vector space

Words/Terms are axes of the space

Documents are points or vectors in this space

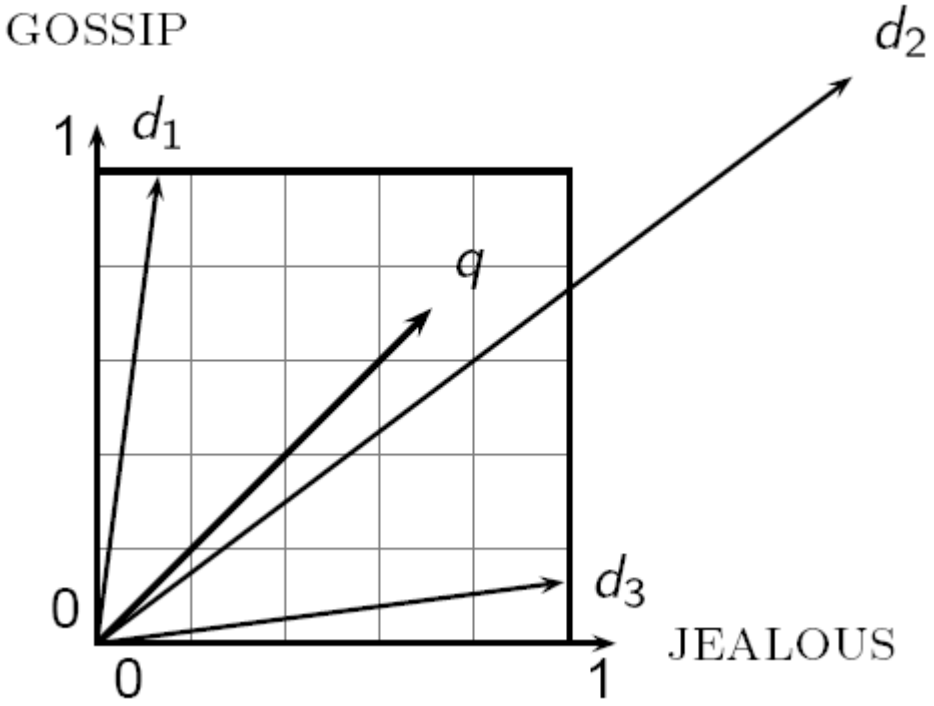
Very high-dimensional: hundred thousand dimensions

These are very sparse vectors - most entries are zero.

Euclidean Distance

$$d(x_1, x_2) = \sqrt{\sum_i \left(x_1^i - x_2^i\right)^2}$$

- dist(q,d_1) < dist(d_1,d_2)
- Non-similar documents closer than similar ones?



Bad idea! because distance is large for vectors of different lengths (words should matter, not the size of documents)

Use angle instead of distance

Thought experiment: take document d and append it to itself.

- Call this document d′.

"Semantically" d and d′ have the same content

The Euclidean distance between the two documents can be quite large

The angle between the two documents is 0

- Should correspond to maximal similarity.

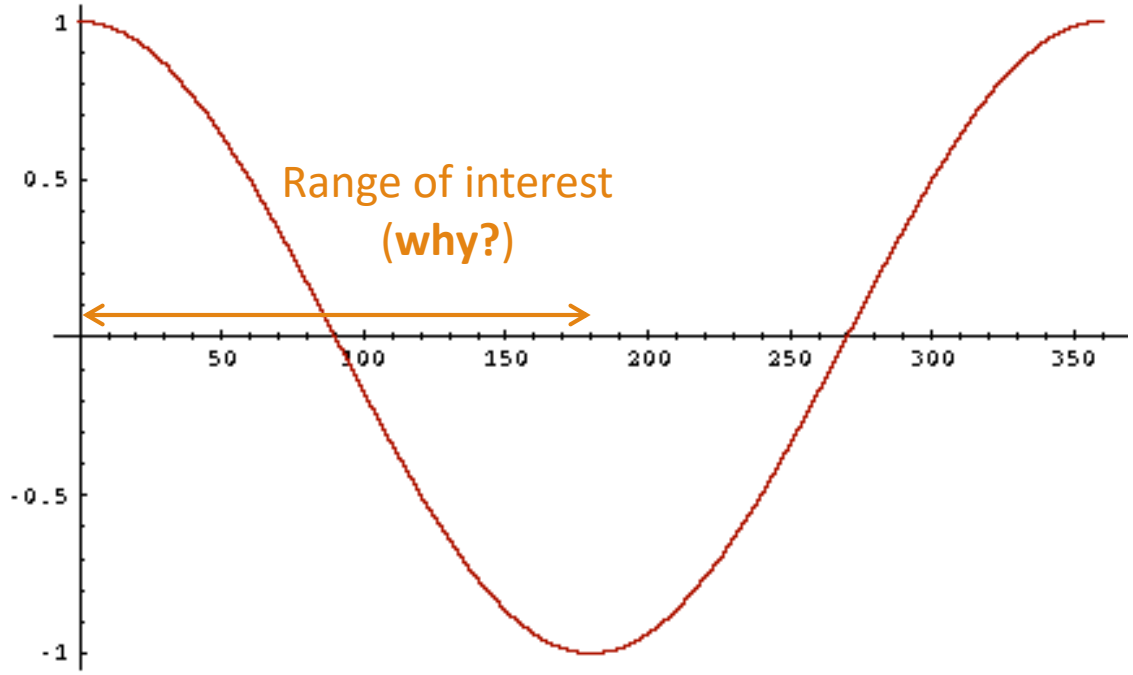
Key idea: Measure "distance" according to angle between docs

The following two notions are equivalent.

Smaller angle between the documents

Larger cosine between the documents

From angles to cosines



But how – and why – should we be computing cosines?

Length normalization

A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

Dividing a vector by its L_2 norm makes it a unit (length) vector

Effect on the documents d and d' (d appended to itself) from earlier:

- They have **identical vectors** after length-normalization!
- Long and short documents now have comparable weights

Cosine Similarity

Dot product

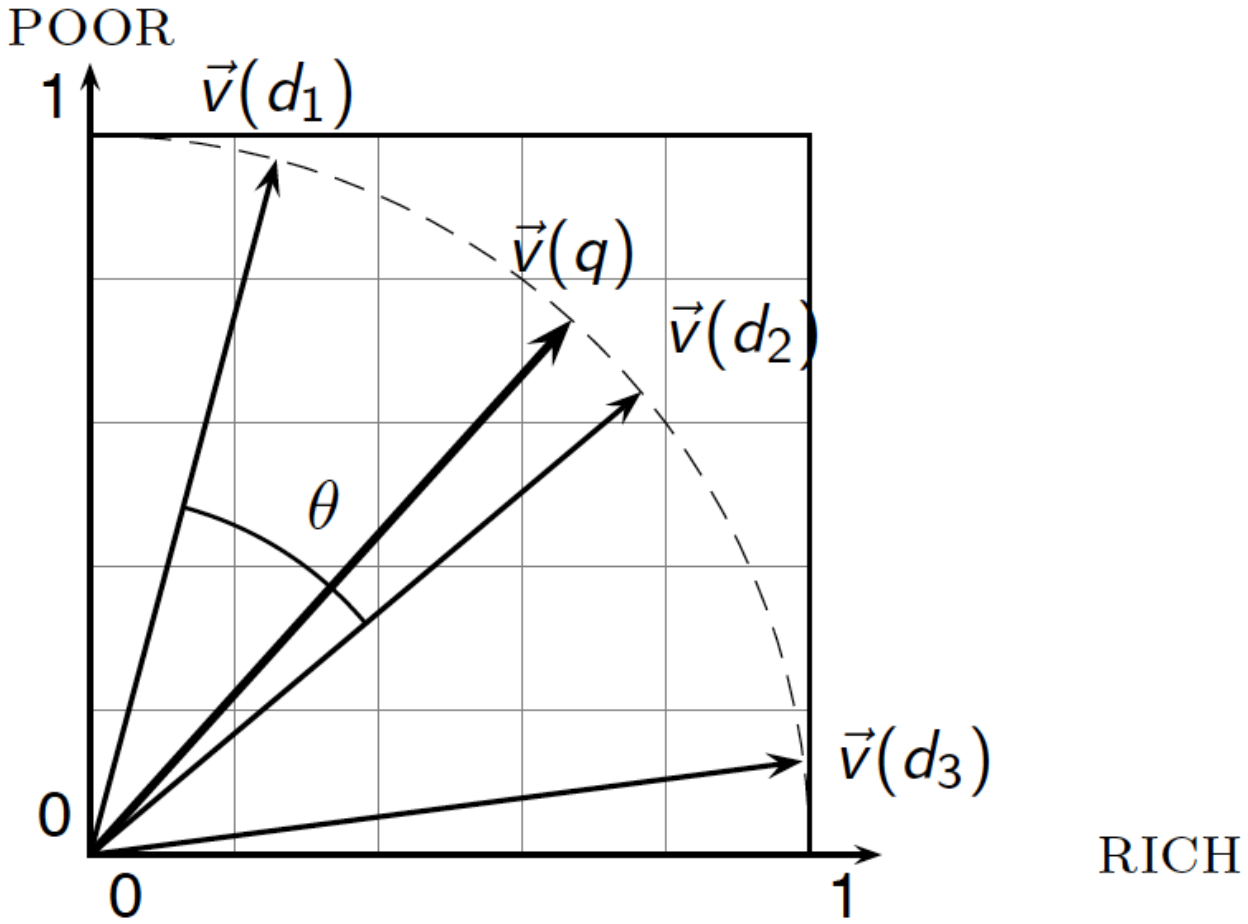
Unit vectors

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

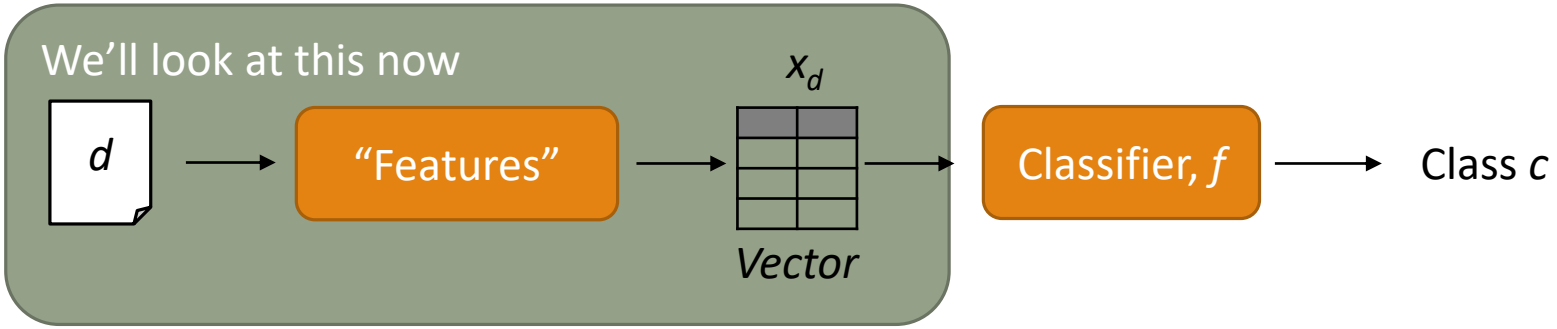
$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Cosine Distance = 1 – Cosine Similarity

Cosine similarity illustrated



Beyond Bag of Words



So far ….

Bag of words model

Problem 1: Don't consider the ordering of words in a document

John is quicker than Mary

vs

Mary is quicker than John

Bag of words model

Problem 1: Don't consider the ordering of words in a document

John is quicker than Mary
vs
Mary is quicker than John

Problem 2: Treat all the words in the document equally

d1: Arachnophobia is the fear of spiders

d2: Spiders! I hate spiders due to Arachnophobia

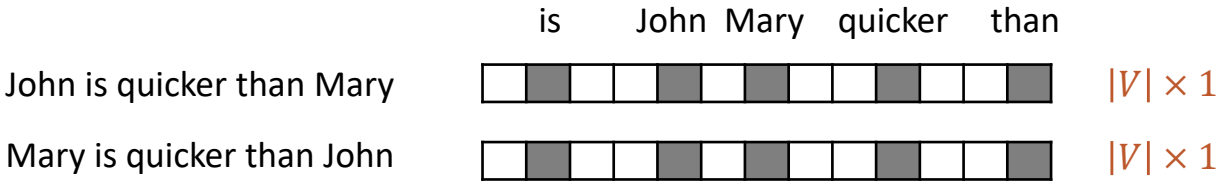
d3: I hate that my bills are due tomorrow.

d3 is closer to **d2** than **d1**!

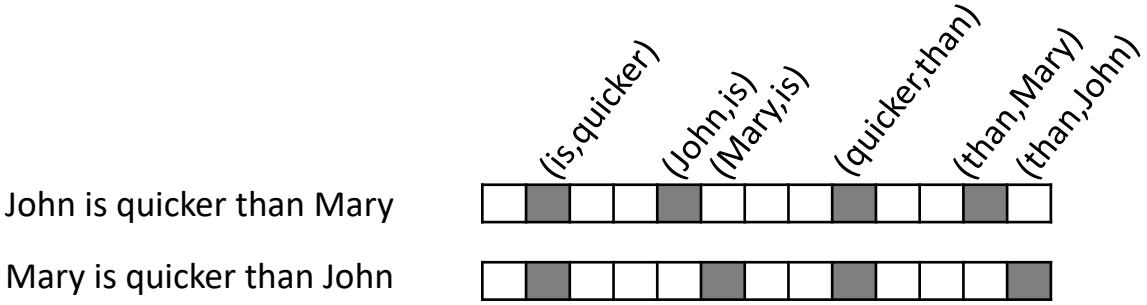
Order Information: Ngrams!

Unigram: Each component of the vector is a word

- Same as Bag of Words model



Bi-grams: Each component of the vector is a "pair of words"

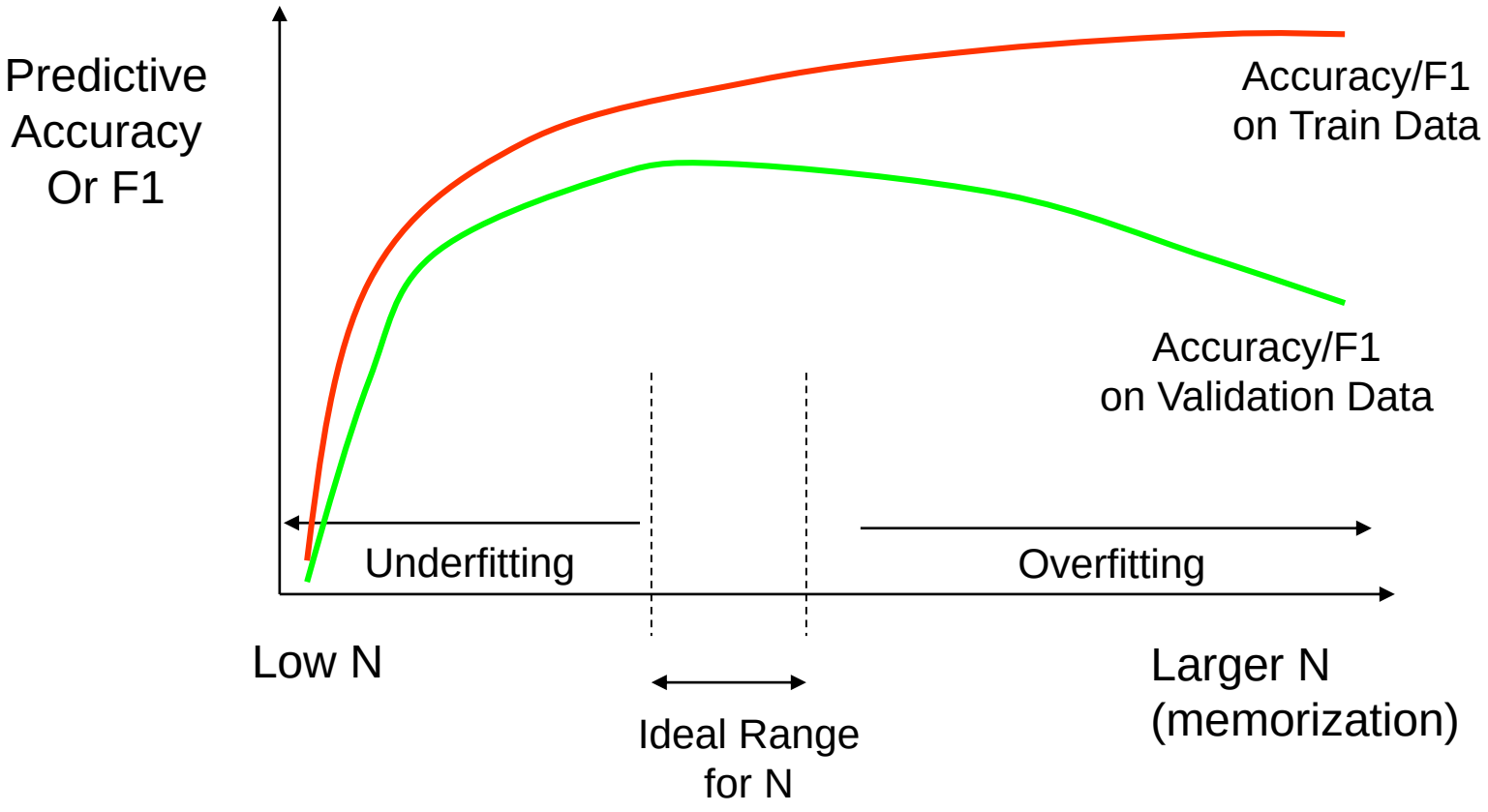


They are different now!

N-grams: Each component of the vector is a "n-tuple of words"

3= Trigram, 4=Fourgram, etc.

How do we select N?



Character n-grams

- In addition to word n-grams, you can also have character n-grams
- E.g.
  - For the sentence “John is quicker than Mary”, the character trigrams would be:
    - Joh
    - ohn
    - hn<space> i
    - n<space>is
    - <space>is<space>
    - is<space>q
    - ….
- Why would this be useful?

Words are not Equal

d1: Arachnophobia is the fear of spiders

d2: Spiders! I hate spiders due to Arachnophobia

d3: I hate that my bills are due tomorrow.

d3 is closer to **d2** than **d1**!

Words are not Equal

d1: Arachnophobia is the fear of spiders

d2: Spiders! I hate spiders due to Arachnophobia

d3: I hate that my bills are due tomorrow.

d3 is closer to **d2** than **d1**!

Spiders is counted once

Term-frequency weighting
(number of times you see word in document)
tf("spiders",d1) = 1 tf("spiders",d2) = 2

Words are not Equal

d1: Arachnophobia is the fear of spiders

d2: Spiders! I hate spiders due to Arachnophobia

d3: I hate that my bills are due tomorrow.

d3 is closer to **d2** than **d1**!

Spiders is counted once

Term-frequency weighting
(number of times you see word in document)
tf("spiders",d1) = 1 tf("spiders",d2) = 2

Arachnophia is rare

Rare words should matter more
(if they appear in two documents, it's informative!)
Captured by **inverse-document frequency** (next…)

IDF weight

df_t is the document frequency of t: number of documents that contain t

- df_t is an inverse measure of the informativeness of word t
- $df_t \leq N$ (number of documents)

We define the idf (inverse document frequency) of t by

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

- We use log (N/dft) instead of N/dft to "dampen" the effect of idf.

Will turn out the base of the log is immaterial.

idf("hate") < idf("arachnophobia")

But, depends on corpus.. In journal of spiders, idf("hate") might be high!

suppose N = 1 million

IDF Example

term	df_t	idf_t
calpurnia	1	?
arachnophobia	100	?
spiders	1,000	?
due	10,000	?
hate	100,000	?
I	1,000,000	?

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

There is one idf value for each term t in a collection.

suppose N = 1 million

TF-IDF Example

d1: Arachnophobia is the fear of spiders

d2: Spiders! I hate spiders due to Arachnophobia

d3: I hate that my bills are due tomorrow.

$$tfidf_{t,d} = tf_{t,d} * log_{10}(N/df_t)$$

term	df_t	idf_t	tf-idf$_{t,d1}$	tf-idf$_{t,d2}$	tf-idf$_{t,d3}$
calpurnia	1	6	?	?	?
arachnophobia	100	4	?	?	?
spiders	1,000	3	?	?	?
due	10,000	2	?	?	?
hate	100,000	1	?	?	?
I	1,000,000	0	?	?	?

Comparing bag of words (BoW) vs. tf-idf

d1: Arachnophobia is the fear of spiders

d2: Spiders! I hate spiders due to Arachnophobia

d3: I hate that my bills are due tomorrow.

term	$BoW_{t,d1}$	$BoW_{t,d2}$	$BoW_{t,d3}$
calpurnia	0	0	0
arachnophobia	1	1	0
spiders	1	1	0
due	0	1	1
hate	0	1	1
I	0	1	1

Comparing bag of words (BoW) vs. tf-idf

d1: Arachnophobia is the fear of spiders

d2: Spiders! I hate spiders due to Arachnophobia

d3: I hate that my bills are due tomorrow.

term	$BoW_{t,d1}$	$BoW_{t,d2}$	$BoW_{t,d3}$
calpurnia	0	0	0
arachnophobia	1	1	0
spiders	1	1	0
due	0	1	1
hate	0	1	1
I	0	1	1

Distance between d1 and d2

$$= 1 - \cos(d1, d2)$$

$$= 1 - \frac{d1 \cdot d2}{\| d1 \| \| d2 \|}$$

$$= 1 - \frac{2}{\sqrt{2}\sqrt{5}} = 0.36$$

Distance between d2 and d3

$$= 1 - \cos(d2, d3)$$

$$= 1 - \frac{3}{\sqrt{5}\sqrt{3}} = 0.23$$

Distance between d1 and d3

$$= 1$$

Comparing bag of words (BoW) vs. tf-idf

d1: Arachnophobia is the fear of spiders

d2: Spiders! I hate spiders due to Arachnophobia

d3: I hate that my bills are due tomorrow.

term	$\text{tf-idf}_{t,d1}$	$\text{tf-idf}_{t,d2}$	$\text{tf-idf}_{t,d3}$
calpurnia	0	0	0
arachnophobia	4	4	0
spiders	3	6	0
due	0	2	2
hate	0	1	1
I	0	0	0

Comparing bag of words (BoW) vs. tf-idf

d1: Arachnophobia is the fear of spiders

d2: Spiders! I hate spiders due to Arachnophobia

d3: I hate that my bills are due tomorrow.

Distance(d1, d2)

$$= 1 - \cos(d1, d2)$$

$$= 1 - \frac{d1 \cdot d2}{\| d1 \| \| d2 \|}$$

$$= 1 - \frac{4 * 4 + 3 * 6}{\sqrt{4^2 + 3^2}\sqrt{4^2 + 6^2 + 2^2 + 1^2}}$$

$$= 0.099$$

term	$tf\text{-}idf_{t,d1}$	$tf\text{-}idf_{t,d2}$	$tf\text{-}idf_{t,d3}$
calpurnia	0	0	0
arachnophobia	4	4	0
spiders	3	6	0
due	0	2	2
hate	0	1	1
I	0	0	0

Distance between d2 and d3 = 0.7

Distance between d1 and d3 = 1

A tweak to term frequency tf

The term frequency tf of term t in document d is defined as the number of times that t occurs in d.

Raw term frequency may not be what we want though

- A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
- But not 10 times more relevant.

Relevance does not increase proportionally with term frequency.

NB: frequency = count in IR

tf-idf weighting

The tf-idf weight of a term is product of its tf weight and its idf weight.

$$w_{t,d} = \log(1+\mathrm{tf}_{t,d}) \times \log_{10}(N/\mathrm{df}_t)$$

Best known weighting scheme in information retrieval

- Note: the "-" in tf-idf is a hyphen, not a minus sign!
- Alternative names: tf.idf, tf x idf

Increases with the number of occurrences within a document

Increases with the rarity of the term in the collection

TF-IDF+Ngrams in Scikit-Learn

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

unigram = CountVectorizer(ngram_range=(1,1))
bigram = CountVectorizer(ngram_range=(2,2))
trigram = CountVectorizer(ngram_range=(3,3))

tfidf = TfidfVectorizer()

#As before, you need to call fit and transform
#ex.
tfidf.fit(train.data)
X_train_vector = tfidf.transform(train.data)
X_test_vector = tfidf.transform(test.data)
```

In-Class Activity 1

TF-IDF AND NGRAM BASED VECTORS