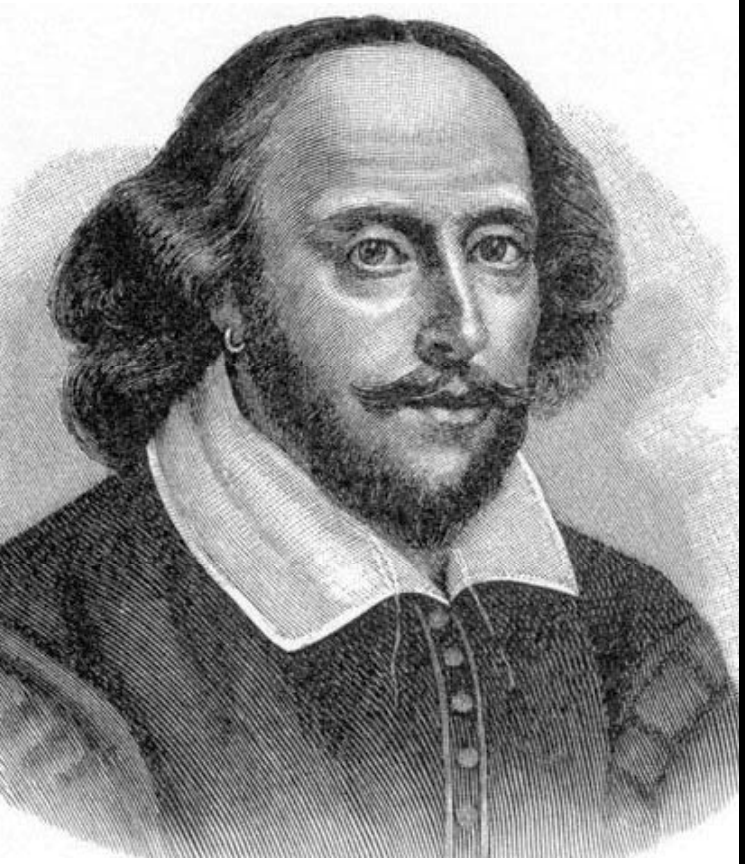
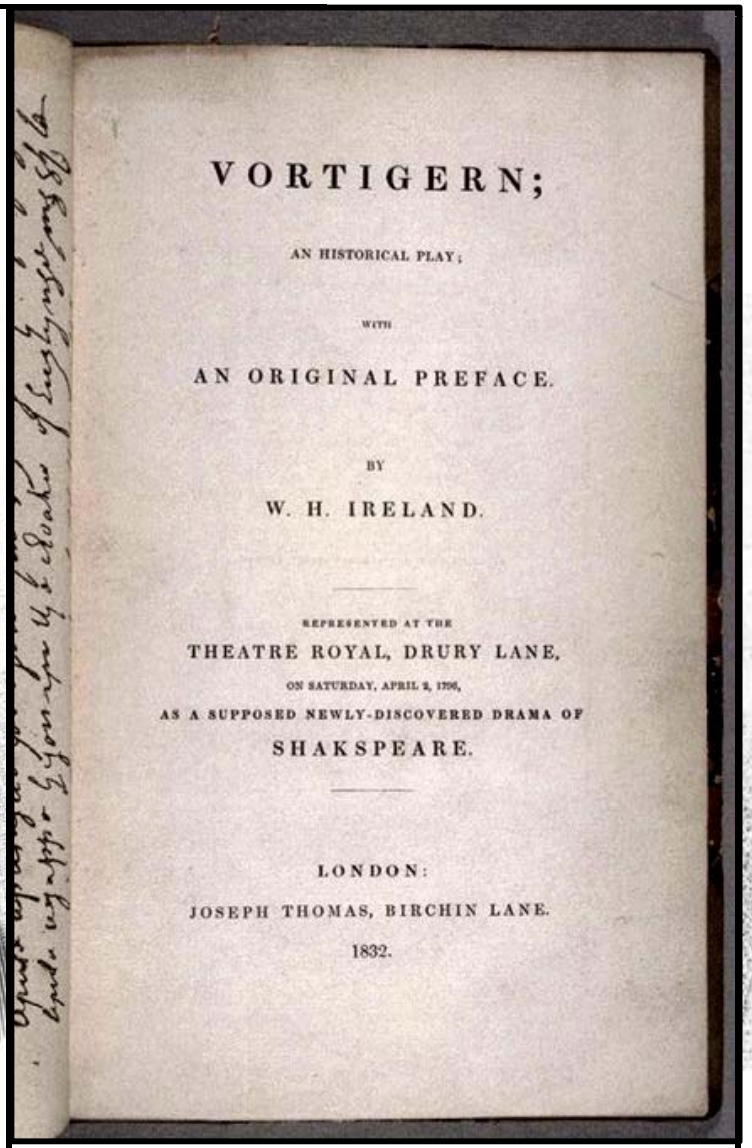
Who wrote this?



William Shakespeare??



VORTIGERN;

AN HISTORICAL PLAY;

WITH

AN ORIGINAL PREFACE.

BY

W. H. IRELAND.

REPRESENTED AT THE

THEATRE ROYAL, DRURY LANE,

ON SATURDAY, APRIL 2, 1796,

AS A SUPPOSED NEWLY-DISCOVERED DRAMA OF

SHAKSPEARE.

LONDON:

JOSEPH THOMAS, BIRCHIN LANE.

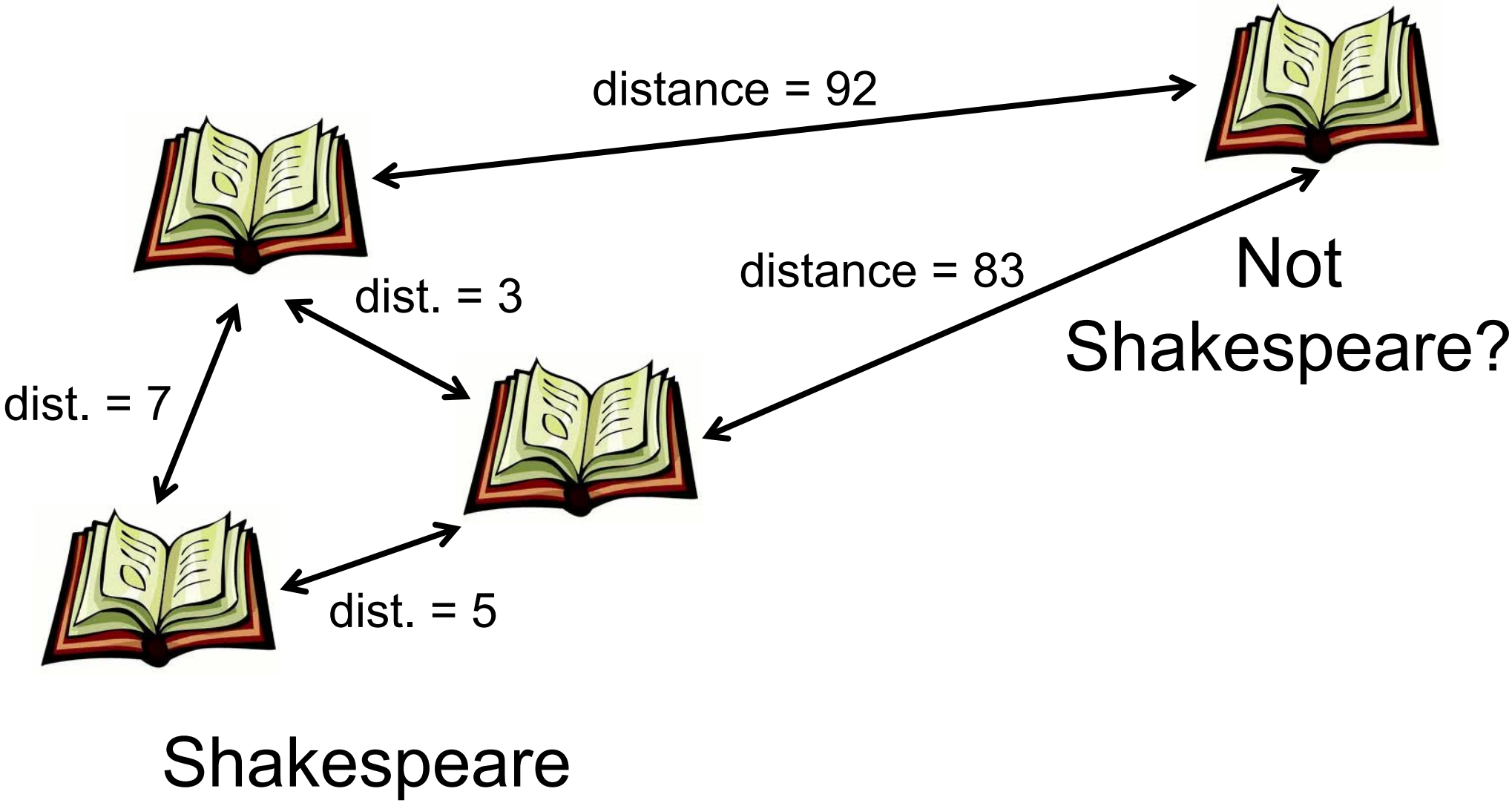
1832.

***mystery* play "found" in 1796**

William Henry Ireland??

Document distance

How similar are two documents?



Metrics of similarity

Measure of distance:

- Number of words in the same order
- Number of shared *uncommon* words
- Same # of words per sentence
- Same ratio of adjectives / nouns
- Written on similar paper / using similar ink

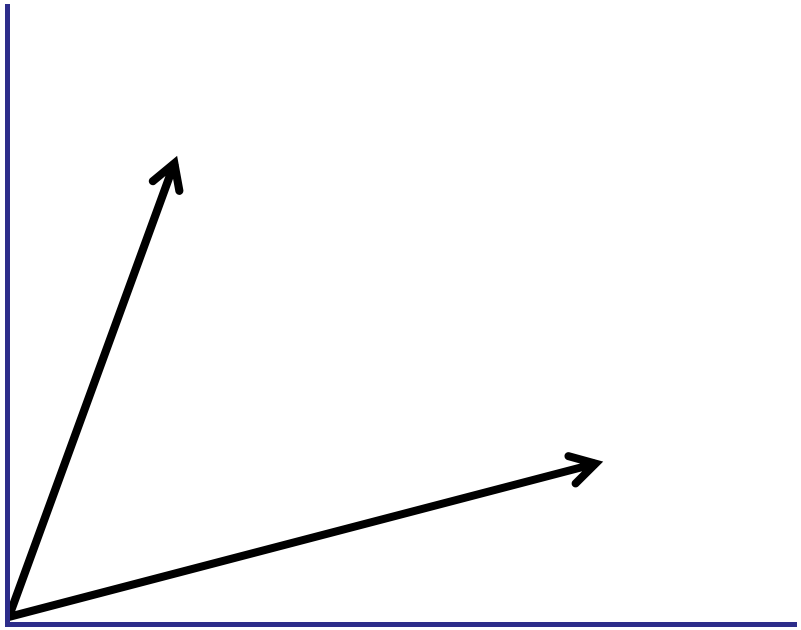
How many times is each word used?

Vector Space Model

Strategy:

- View each document as a high-dimensional vector.

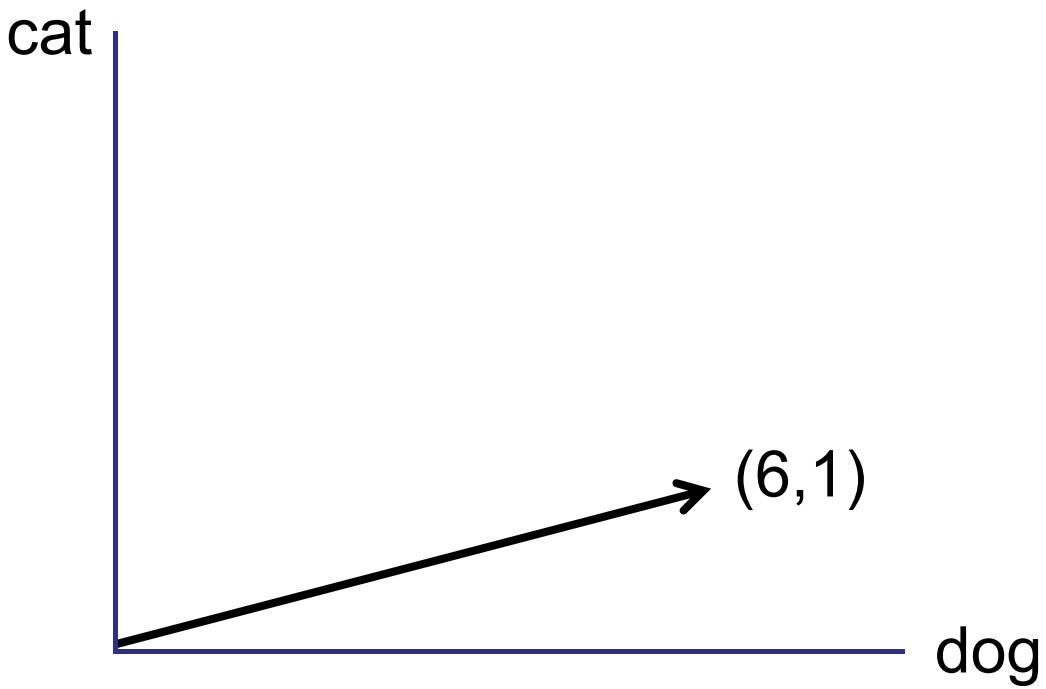
[Salton, Wang, Yang '75]



Vector Space Model

Two word language: dog + cat

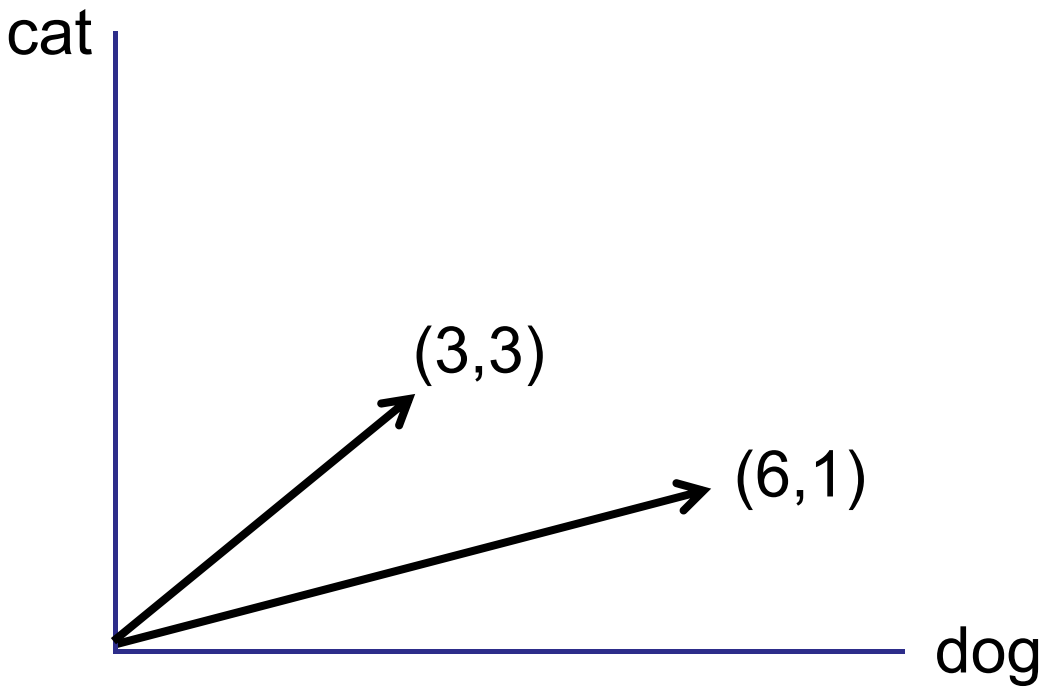
1. dog dog dog cat dog dog dog



Vector Space Model

Two word language: dog + cat

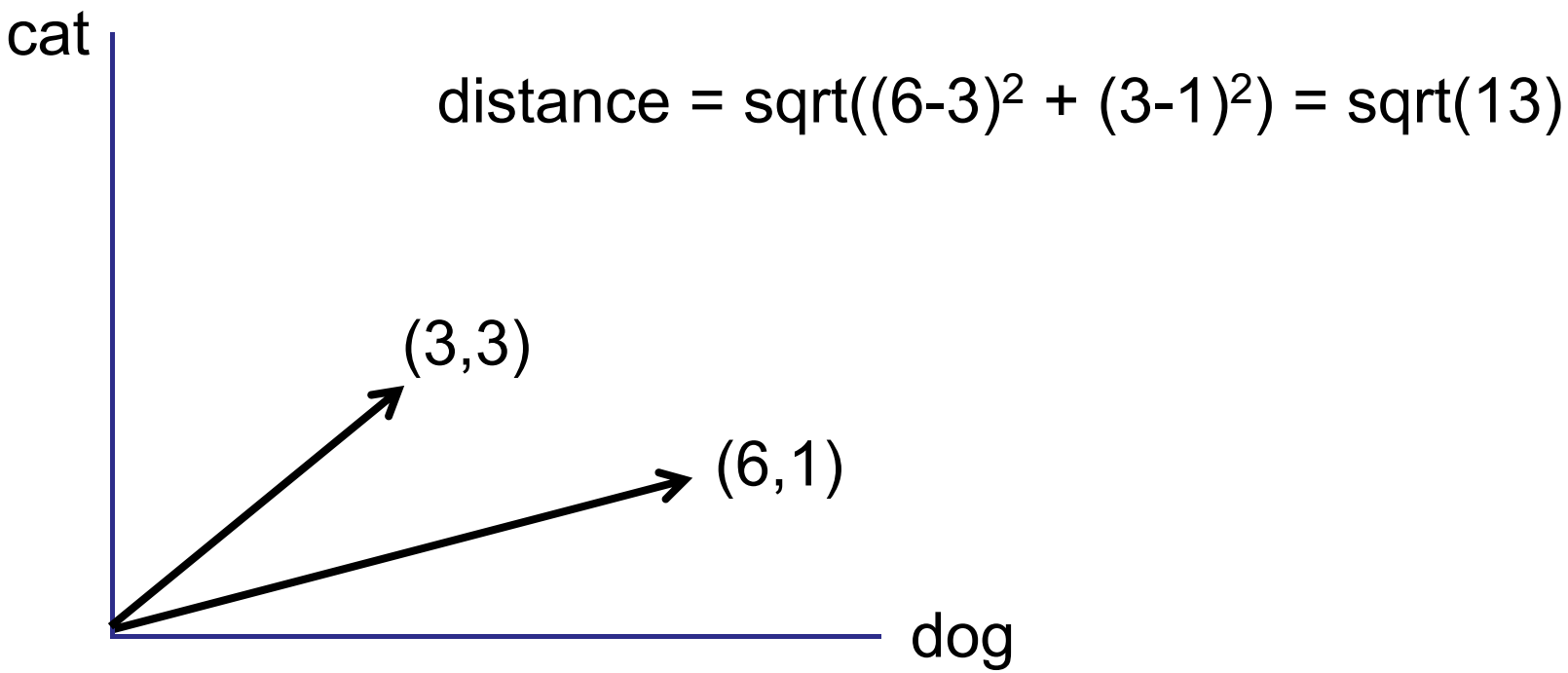
1. dog dog dog cat dog dog dog
2. cat cat dog cat dog dog



Vector Space Model

Two word language: dog + cat

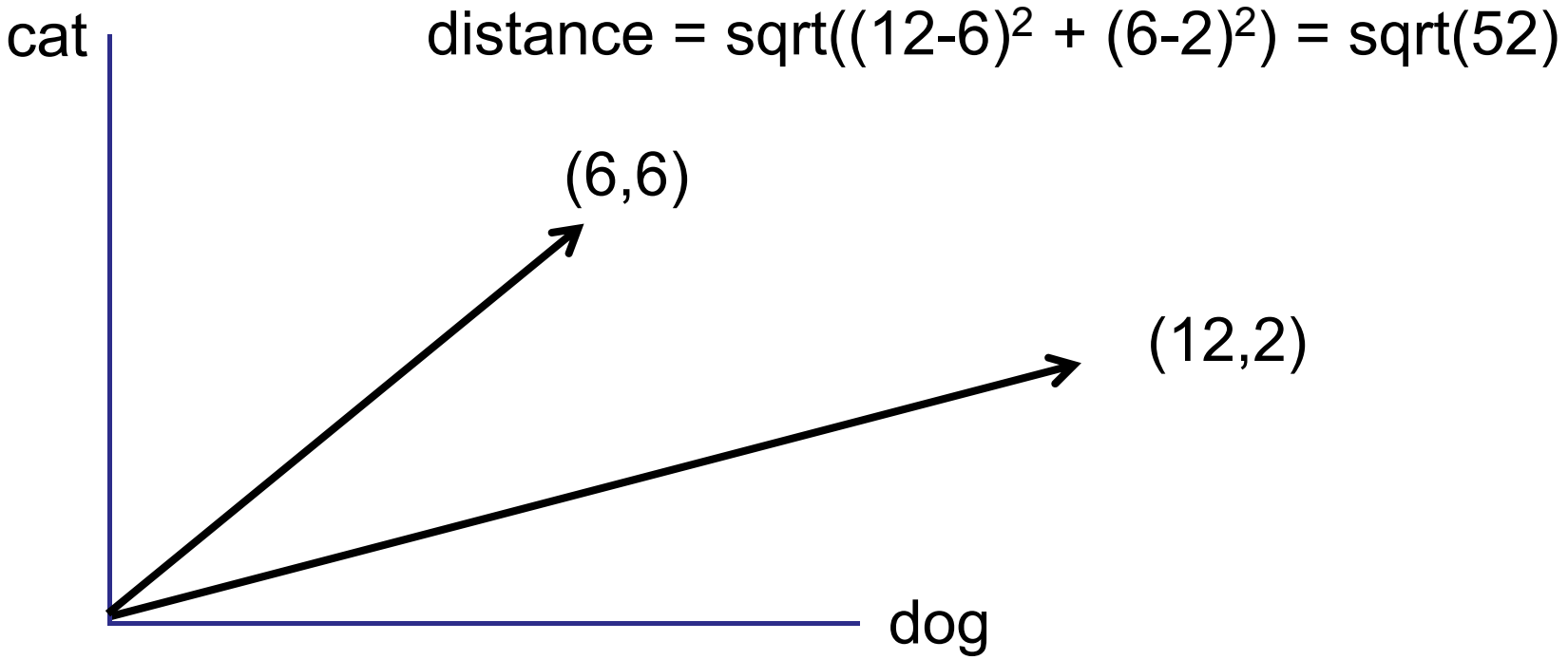
1. dog dog dog cat dog dog dog
2. cat cat dog cat dog dog



Vector Space Model

Two word language: dog + cat

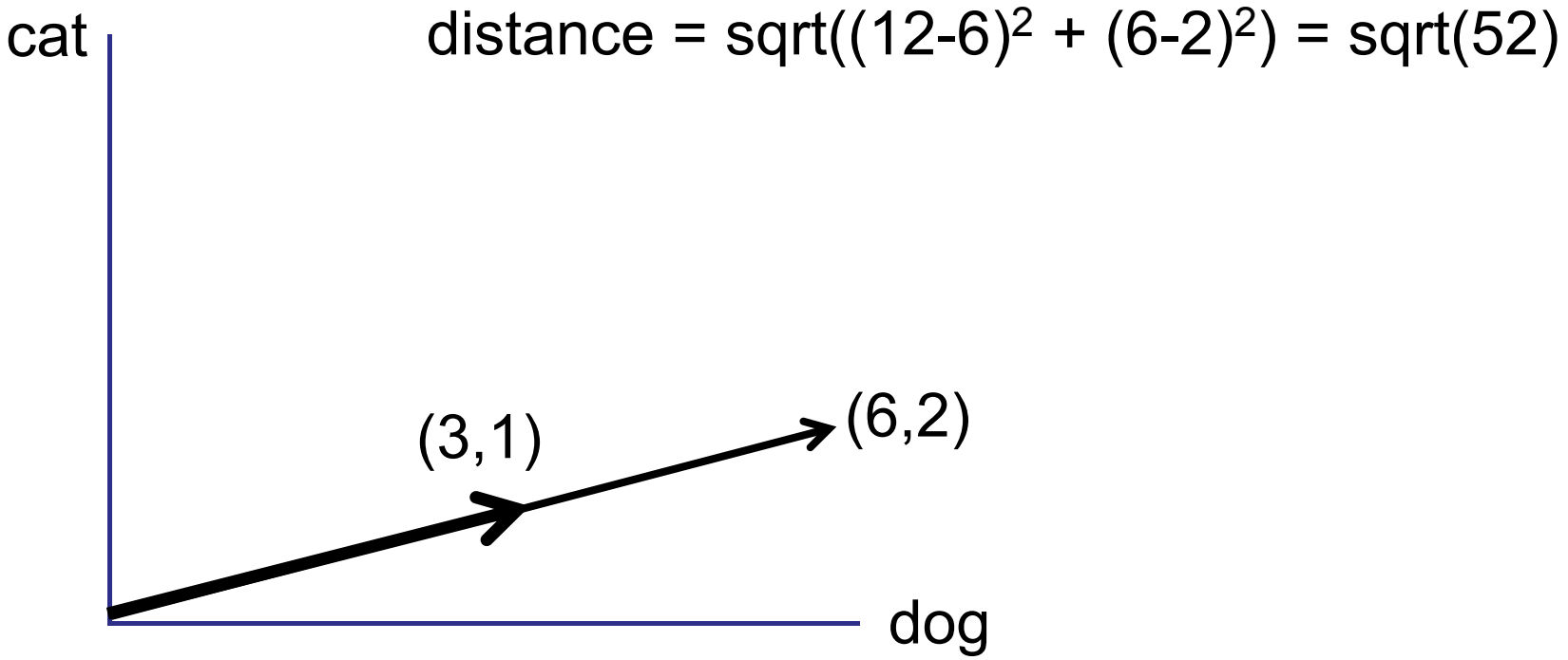
1. (dog dog dog cat dog dog dog) x 2
2. (cat cat dog cat dog dog)



Vector Space Model

Two word language: dog + cat

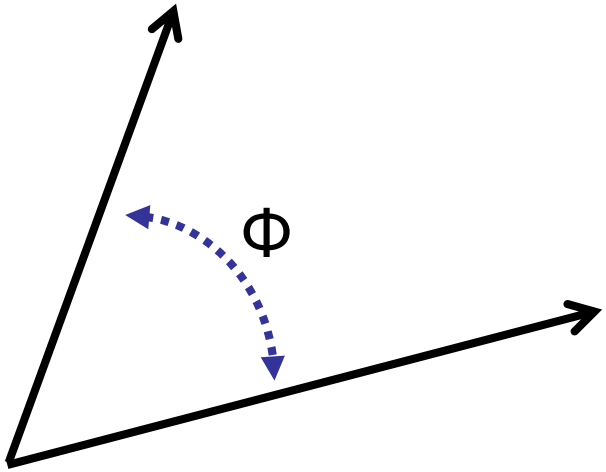
1. dog dog dog cat
2. dog dog dog dog dog dog cat cat



Vector Space Model

Strategy:

- View each document as a high-dimensional vector.
- The *metric of similarity* is the angle between the two vectors.



- Identical: $\Phi = 0$
- No words in common: $\Phi = \pi/2$

[Salton, Wang, Yang '75]

Vector Space Model

Document as vector:

Example 1:

"to be or not to be" = [2,1,1,2]

be	not	or	to
2	1	1	2

Vector Space Model

Example 1:

"to be or not to be" = [0,2,0,1,0,1,2]

Example 2:

"be not afraid of greatness" = [1,1,1,1,1,0,0]

afraid	be	greatness	not	of	or	to
1	1	1	1	1	0	0

Vector Space Model

Law of cosines:

$$\Theta(v, w) = \cos^{-1}\left(\frac{v \cdot w}{\|v\| \cdot \|w\|}\right)$$

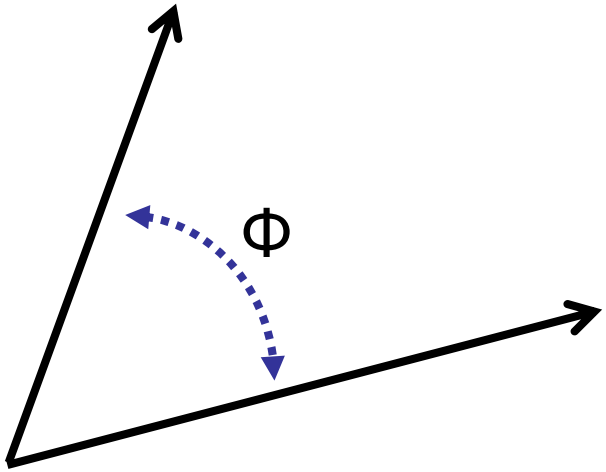
Notes:

- Φ is an angle between (0, pi)
- If (v=w), then Φ=0.
- If (v • w) = 0, then Φ=pi.

Vector Space Model

Strategy:

- View each document as a high-dimensional vector.
- The *metric of similarity* is the angle between the two vectors.



- Identical: $\Phi = 0$
- No words in common: $\Phi = \pi/2$

[Salton, Wang, Yang '75]

Comparing two texts

Compare Two Documents

Given: documents A and B

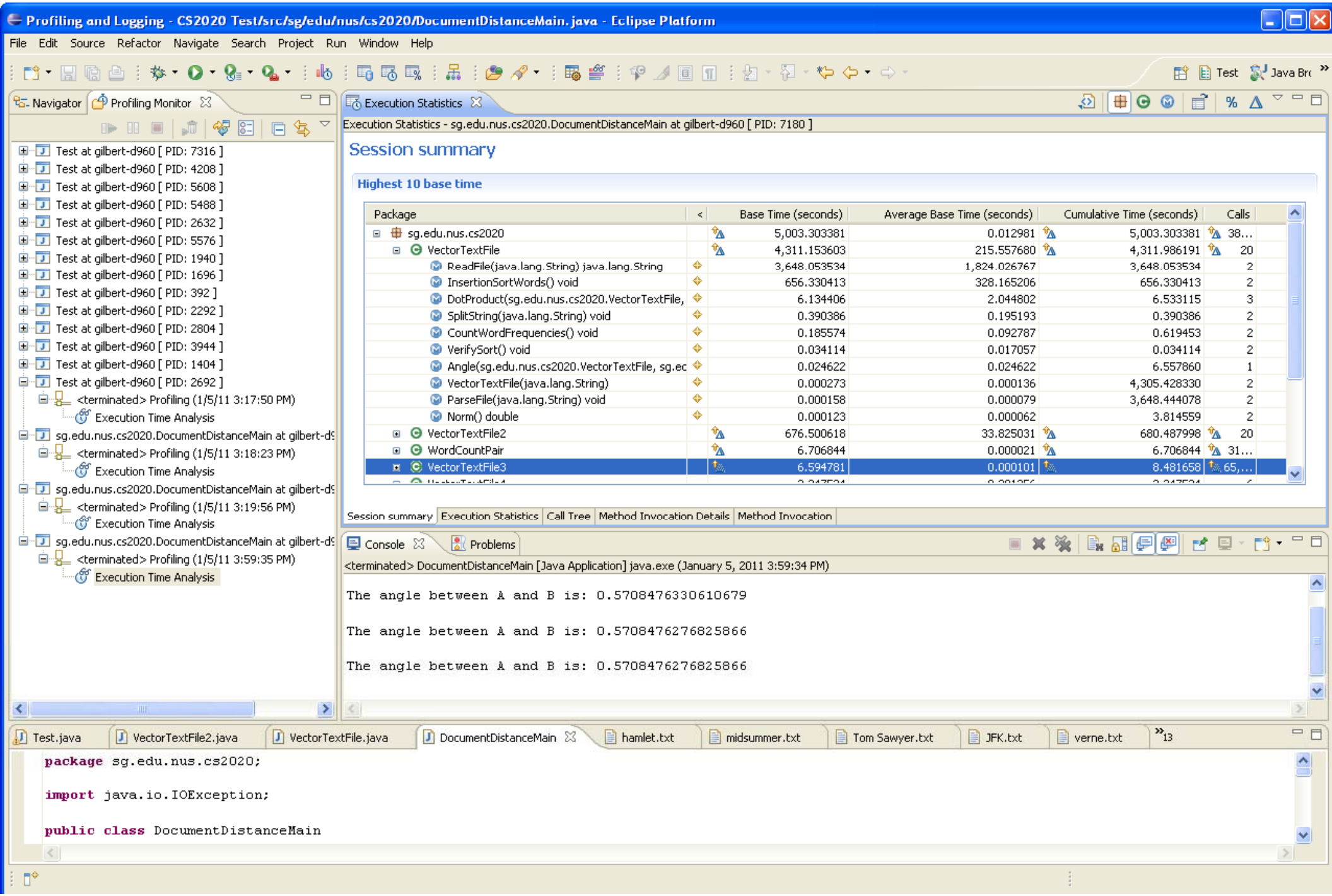
1. Read and parse text
2. Create vectors v_A and v_B
3. Calculate norm: $|v_A|$
4. Calculate norm: $|v_B|$
5. Calculate dot product: $(v_A \cdot v_B)$
6. Calculate angle $\Phi(v_A, v_B)$

Performance Profiling

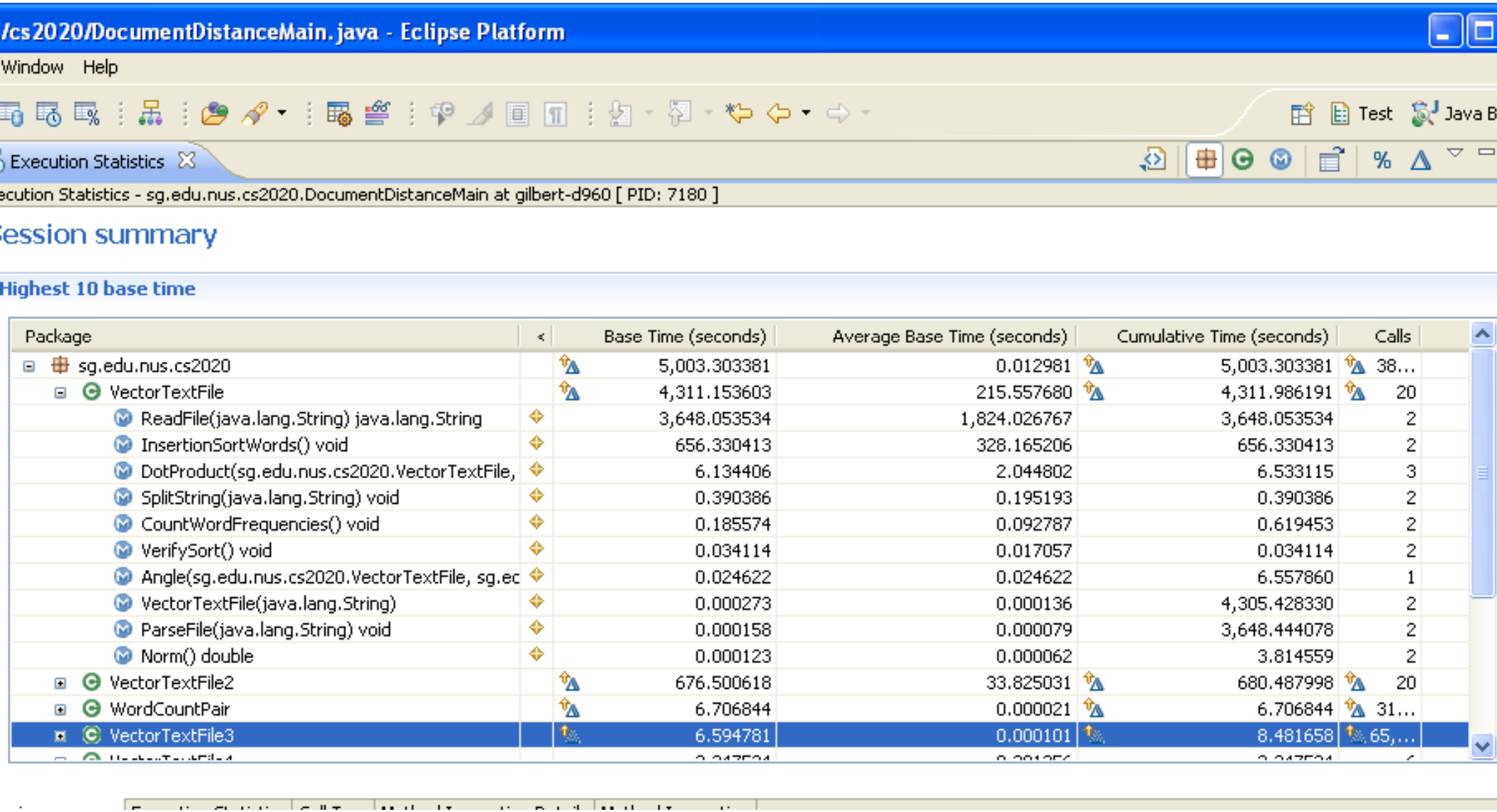
(*Dracula* vs. *Lewis & Clark*)

Step	Function	Running Time
Create vectors:	Read each file	1,824.00s
	Parse each file	0.20s
	Sort words in each file	328.00s
	Count word frequencies	0.31s
Dot product:		6.12s
Norm:		3.81s
Angle:		6.56s
Total:		**72minutes ≈ 4,311.00s**

Eclipse-TPTP



Eclipse-TPTP



Package	<	Base Time (seconds)	Average Base Time (seconds)	Cumulative Time (seconds)	Calls
sg.edu.nus.cs2020		5,003.303381	0.012981	5,003.303381	38...
VectorTextFile		4,311.153603	215.557680	4,311.986191	20
ReadFile(java.lang.String) java.lang.String		3,648.053534	1,824.026767	3,648.053534	2
InsertionSortWords() void		656.330413	328.165206	656.330413	2
DotProduct(sg.edu.nus.cs2020.VectorTextFile,		6.134406	2.044802	6.533115	3
SplitString(java.lang.String) void		0.390386	0.195193	0.390386	2
CountWordFrequencies() void		0.185574	0.092787	0.619453	2
VerifySort() void		0.034114	0.017057	0.034114	2
Angle(sg.edu.nus.cs2020.VectorTextFile, sg.ec		0.024622	0.024622	6.557860	1
VectorTextFile(java.lang.String)		0.000273	0.000136	4,305.428330	2
ParseFile(java.lang.String) void		0.000158	0.000079	3,648.444078	2
Norm() double		0.000123	0.000062	3.814559	2
VectorTextFile2		676.500618	33.825031	680.487998	20
WordCountPair		6.706844	0.000021	6.706844	31...
VectorTextFile3		6.594781	0.000101	8.481658	65,...

Performance Profiling

StopWatch

```
watch = new StopWatch();
watch.start();

      // your code here

watch.stop();
float time = watch.getTime();
```

Performance Profiling

(*Dracula* vs. *Lewis & Clark*)

Step	Function	Running Time
Create vectors:	Read each file	1,824.00s
	Parse each file	0.20s
	Sort words in each file	328.00s
	Count word frequencies	0.31s
Dot product:		6.12s
Norm:		3.81s
Angle:		6.56s
Total:		**72minutes ≈ 4,311.00s**

Performance Profiling

(*Dracula* vs. *Lewis & Clark*)

Version	Change	Running Time
Version 1		4,311.00s
Version 2	Better file handling	676.50s
Version 3	Faster sorting	6.59s
Version 4	No sorting!	2.35s

- Version 4 will be released later in the semester…

Performance Profiling

(*Dracula* vs. *Lewis & Clark*)

Step	Function	Running Time
Create vectors:	Read each file	1,824.00s
	Parse each file	0.20s
	Sort words in each file	328.00s
	Count word frequencies	0.31s
Dot product:		6.12s
Norm:		3.81s
Angle:		6.56s
Total:		**72minutes ≈ 4,311.00s**

ReadFile (excerpt)

```
// Open the file as a stream and find its size
inputStream = new FileInputStream(fileName);
iSize = inputStream.available();

// Read in the file, one character at a time, normalizing as we go.
for (int i=0; i<iSize; i++)
{
    // Read a character
    char c = (char)inputStream.read();

    // Ensure that the character is lower-case
    c = Character.toLowerCase(c);

    // Check if the character is a letter
    if (Character.isLetter(c))
    {
        strTextFile = strTextFile + c;
    }
    // Check if the character is a space or an end-of-line marker
    else if (((c == ' ') || (c == '\n')) && (!strTextFile.endsWith(" ")))
    {
            strTextFile = strTextFile + ' ';
    }
}
```

String Problem!

What happens when:

- strTextFile = strTextFile + c

1. Creates new temporary string.
2. Copies strTextFile to the new string.
3. Adds the new character *c*.
4. Reassigns strTextFile to point to the new string.

String Problem!

What happens when:

- strTextFile = strTextFile + c

1. Creates new temporary string.
2. **Copies strTextFile to the new string**.
3. Adds the new character *c*.
4. Reassigns strTextFile to point to the new string.

Copying a string of k characters takes time O(k).

How long does it take to read a file containing n characters?

```
// Open the file as a stream and find its size
inputStream = new FileInputStream(fileName);
iSize = inputStream.available();

// Read in the file, one character at a time, normalizing as we go.
for (int i=0; i<iSize; i++)
{
    // Read a character
    char c = (char)inputStream.read();

    // Ensure that the character is lower-case
    c = Character.toLowerCase(c);

    // Check if the character is a letter
    if (Character.isLetter(c))
    {
        strTextFile = strTextFile + c;
    }
    // Check if the character is a space or an end-of-line marker
    else if (((c == ' ') || (c == '\n')) && (!strTextFile.endsWith(" ")))
    {
                strTextFile = strTextFile + ' ';
    }
}
```

How long does it take here to read a file containing n characters?

1. O(n)
2. O(n log n)
3. ✓ $O(n^2)$
4. $O(2^n)$
5. Big-O notation?

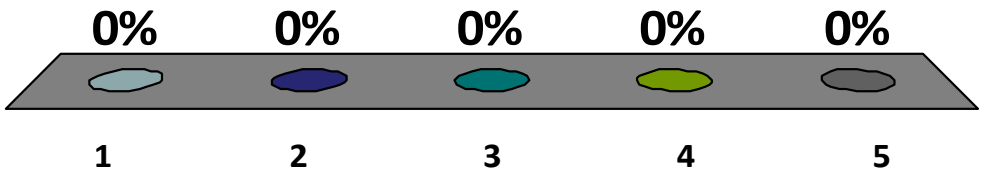
```
// Open the file as a stream and find its size
inputStream = new FileInputStream(fileName);
iSize = inputStream.available();

// Read in the file, one character at a time, normalizing as
for (int i=0; i<iSize; i++)
{
    // Read a character
    char c = (char)inputStream.read();

    // Ensure that the character is lower-case
    c = Character.toLowerCase(c);

    // Check if the character is a letter
    if (Character.isLetter(c))
    {
        strTextFile = strTextFile + c;
    }
    // Check if the character is a space or an end-of-line ma
    else if (((c == ' ') || (c == '\n')) && (!strTextFile.end
    {
            strTextFile = strTextFile + ' ';
    }
}
```

Response Counter



String Problem!

How long to read in a file of n characters?.

$$1 + 2 + 3 + 4 + \ldots + n = n(n+1)/2 = \Theta(n^2)$$

Very, very, very slow!

Fix the string problem!

```
// Open the file as a stream and find its size
inputStream = new FileInputStream(fileName);
iSize = inputStream.available();

// Initialize the char buffer to be arrays of the appropriate size.
charBuffer = new char[iSize];

// Read in the file, one character at a time, normalizing as we go.
for (int i=0; i<iSize; i++)
{
    // Read a character
    char c = (char)inputStream.read();

    // Ensure that the character is lower-case
    c = Character.toLowerCase(c);

    // Check if the character is a letter
    if (Character.isLetter(c))
    {
        charBuffer[iCharCount] = c;
        iCharCount++;
    }
    // Check if the character is a space or an end-of-line marker
    else if (((c == ' ') || (c == '\n')) && (!strTextFile.endsWith(" ")))
    {
        charBuffer[iCharCount] = ' ';
        iCharCount++;
    }
}
```

Performance Profiling, V2

(*Dracula* vs. *Lewis & Clark*)

Step	Function	Running Time
Create vectors:	Read each file	1.09s
	Parse each file	3.68s
	Sort words in each file	332.13s
	Count word frequencies	0.30s
Dot product:		6.06s
Norm:		3.80s
Angle:		6.06s
Total:		**11minutes ≈ 680.49s**

Goals for the Semester

Algorithms:

- Design of efficient algorithms
- Analysis of algorithms

Implementation:

- Solve real problems
- Analyze and profile performance
- Improve performance via better algorithms

Document Distance

(*Dracula* vs. *Lewis & Clark*)

Version	Change	Running Time
Version 1		4,311.00s
Version 2	Better file handling	676.50s
Version 3	Faster sorting	6.59s
Version 4	No sorting!	2.35s