

Bag of Words, N-Grams & Cosine Similarity

DATA 3301 Introduction to Data Science Lecture Notes (ANSWER KEY)

Name: _____ Date: _____

1. Text is data too

- **Information Retrieval (IR):** given a document **collection** and a **query**, find all documents **relevant** to the query. It is the foundation of web **search engines**.
- IR is based on the ability to compute **similarity** (or distance) between documents.
- **Context matters:** among a shelf of all kinds of books, Cat in the Hat and Fox in Socks look like a best match. Among only Dr. Seuss books, that is less clear. Similarity depends on the **collection** we compare within.
- We don't know how to compare raw text, but we DO know how to compare rows of **numbers**. Turning text into numbers is called **vectorization**.

2. Option 1: Bag of Words / Unigrams / Term Frequencies

- All three names are **synonyms** for the same idea, used in different contexts.
 - **unigram** = one **indivisible** word/term. **term frequency (TF)** = # of **times** a term occurs. **bag** = a **set** with element counts.
- Count the unigrams in: I AM SAM. I AM SAM. SAM I AM. THAT SAM-I-AM!
THAT SAM-I-AM! I DO NOT LIKE THAT SAM-I-AM!

unigram	AM	DO	I	LIKE	NOT	SAM	THAT
TF	<u>6</u>	1	<u>2</u>	1	1	<u>6</u>	<u>3</u>

- Bags of words are **sparse**: most entries are 0. In the document-term matrix, each row is a **document** and each column is one term of the **vocabulary**.
- The more documents we represent, the **larger** the vocabulary and the **sparser** the bags of words.
- **Disadvantage of unigrams:** loss of any **positional** information. "The dog bit her owner" and "Her dog bit the owner" get **identical** vectors, but mean different things.

3. Option 2: Bigrams and N-grams

- A **bigram** is a pair of **consecutive** words found in text.
- "The dog bit her owner" → "the dog", "**dog bit**", "**bit her**", "her owner"
- "Her dog bit the owner" → "her dog", "dog bit", "**bit the**", "**the owner**"

	(bit,her)	(bit,the)	(dog,bit)	(her,dog)	(the,dog)
1: The dog bit...	1	<u>0</u>	<u>1</u>	0	1
2: Her dog bit...	0	1	1	<u>1</u>	<u>0</u>

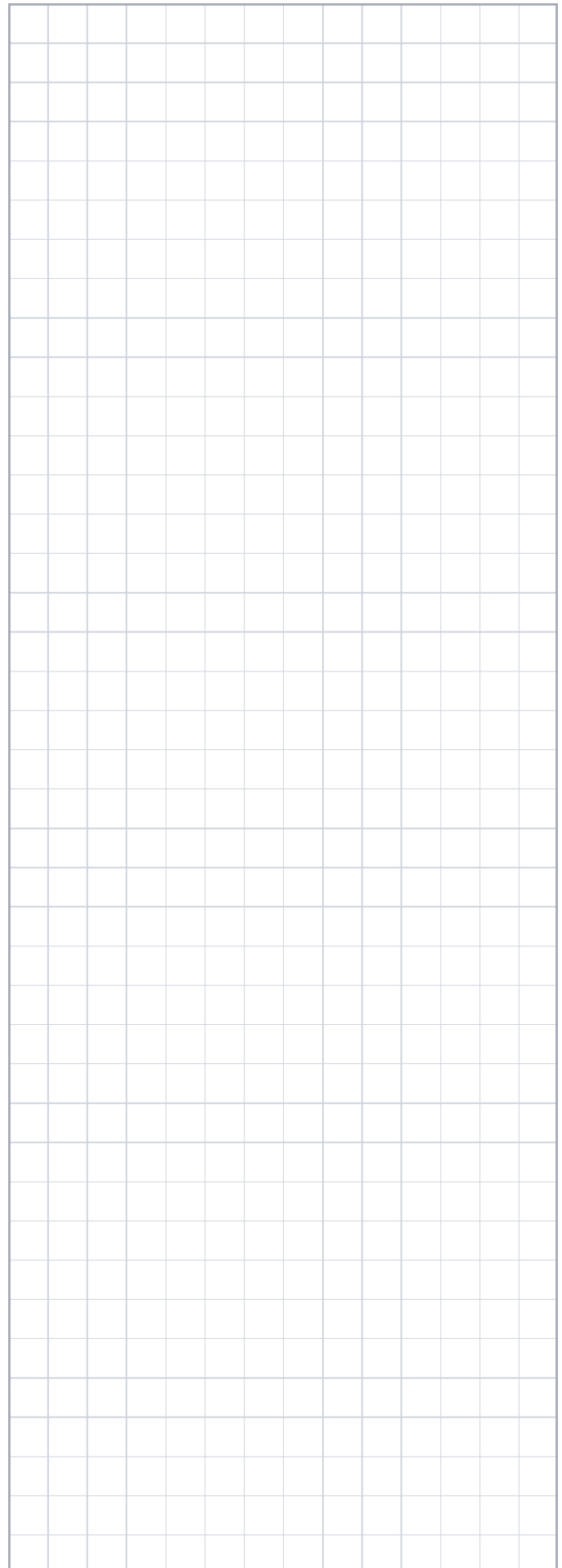
- Now the two representations are **different**.
- **Start/stop markers** (\$) and (#) are sometimes added: "\$ The dog bit her owner #" adds "**\$ the**" and "**owner #**". This keeps bigrams within a **single sentence**.
- An **n-gram** is a sequence of **n** consecutive words. Trigrams of "The dog bit her owner": "the dog bit", "**dog bit her**", "bit her owner". It has 2 4-grams.
- Option 3, **TF-IDF** (term frequency - inverse document frequency), is coming next week.

4. Information retrieval in two steps

- **Step 1:** convert text to **vectors of numbers** (unigrams, bigrams, ...). **Step 2:** compare text documents, which needs a **similarity metric**.

5. When Euclidean & Manhattan are not the right tool

- Legislators plotted by # Yes votes (x) and # No votes (y) on new taxes. Circle votes more often **against**; triangle votes more often **for**; square votes for more than against but casts few votes overall.
- Euclidean distance says the circle is closer to the **square** than the triangle is. But the square and the **triangle** are the similar legislators. (Manhattan distance tells the **same** story.)
- Sometimes it's not the distance, it's the **direction**. We can measure the **angle** between the two vectors (arrows from the origin).
- We use cosine, not sine: $\cos(0^\circ) = 1$, $\cos(90^\circ) = 0$. Smaller angles = **higher** similarity.
- For documents, a vector's length reflects the **size** of the document; its direction reflects its **content**. Cosine similarity finds similar content even when sizes differ.



6. Cosine similarity

- The cosine of the angle between two vectors is their **normalized dot product**. (Here "vector" = "data point".)
- Dot product:** $x \cdot y = x_1 \cdot y_1 + x_2 \cdot y_2 + \dots = \sum_i x_i \cdot y_i$
- "Normalized" means divided by the **lengths** of both vectors: $|x| = \sqrt{\sum_i x_i^2}$
 $\cos(x, y) = x \cdot y / (|x| \cdot |y|)$
- Worked example.** S1 = "I am Sam...", S2 = "Please, sir. I do not like this trick, sir." Only shared words add to the dot product: DO, I, LIKE, NOT $\rightarrow$ S1·S2 = 1 + 7 + 1 + 1 = **10**.
- $|S1| = 11.18$, $|S2| = 3.31$, so $\cos(S1, S2) = 10 / (11.18 \times 3.31) \approx$ **0.27**.

```
dot_product = (cat * fox).sum()      # Step 1
cat_length = sqrt((cat*cat).sum())   # Step 2
sim_c_f = dot_product / (cat_length * fox_length)
# cat~fox 0.384 | cat~eggs 0.456 | fox~eggs 0.219
```

- Of those three pairs, the most similar is Cat in the Hat & **Green Eggs and Ham**.

7. scikit-learn: CountVectorizer

- Lives in the sklearn.**feature_extraction.text** module and implements the **fit-transform** model.

```
from sklearn.feature_extraction.text import CountVectorizer
vec = CountVectorizer()
vec.fit(docs.values())
unigrams = vec.transform(docs.values())
# → 8x1344 sparse matrix ("Argh!")
mx = unigrams.todense() # sparse → dense
pd.DataFrame(mx, index=seuss_files,
             columns=vec.get_feature_names_out())
```

- Result: 8 rows (one per **document**) × 1344 columns (one per **term**).
- ngram_range=(a, b) collects all **a**-grams through **b**-grams. Default: **(1, 1)**.

ngram_range	What gets collected	# columns
(1, 1)	unigrams	1344
(2, 2)	only bigrams	5846
(1, 2)	unigrams and bigrams	7190

8. scikit-learn: cosine_similarity

- sklearn.metrics.pairwise.**cosine_similarity**(X, Y). X and Y must be **arrays of vectors**, but they can be arrays of size 1, hence the double brackets:

```
cosine_similarity(df01.loc[['fox_in_socks.txt']],
                 df01.loc[['cat_in_the_hat.txt']])
# unigrams only: 0.384
# unigrams and bigrams: 0.330
```

- Adding bigrams made Fox and Cat look **less** similar.
- All vs. all:** an 8 × 8 similarity matrix. Its diagonal is all **1**s. The most similar pair is Grinch & **Horton** (0.79).

9. In summary

- Text documents can be **vectorized** with bag-of-words (unigrams, bigrams).
- We use **cosine similarity** to compare document vectors.
- scikit-learn map: **Preprocessing** $\rightarrow$ vectorizing with **CountVectorizer** (TfidfVectorizer next); **Model evaluation** $\rightarrow$ metrics.pairwise: pairwise_distances(), euclidean_distances(), manhattan_distances(), **cosine_similarity()**.