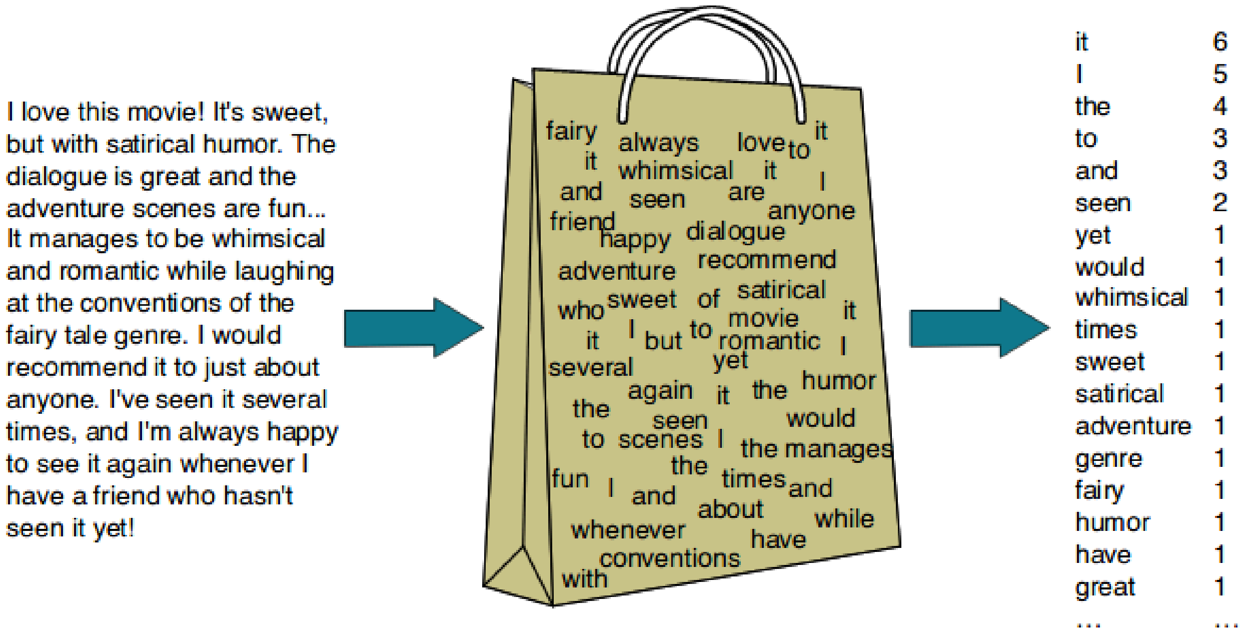
Bag Of Words



I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

What is Bag of Words (BoW)?

Bag of Words is a technique to **convert text (sentences)** into **numbers** so that **machines can understand and analyze it**.

- It **counts how many times each word** appears in a sentence or document.
- It **ignores grammar, word order, and meaning**.
- It just cares **which words exist and how many times**.

> "I love pizza and I love burgers."

Into something like this:

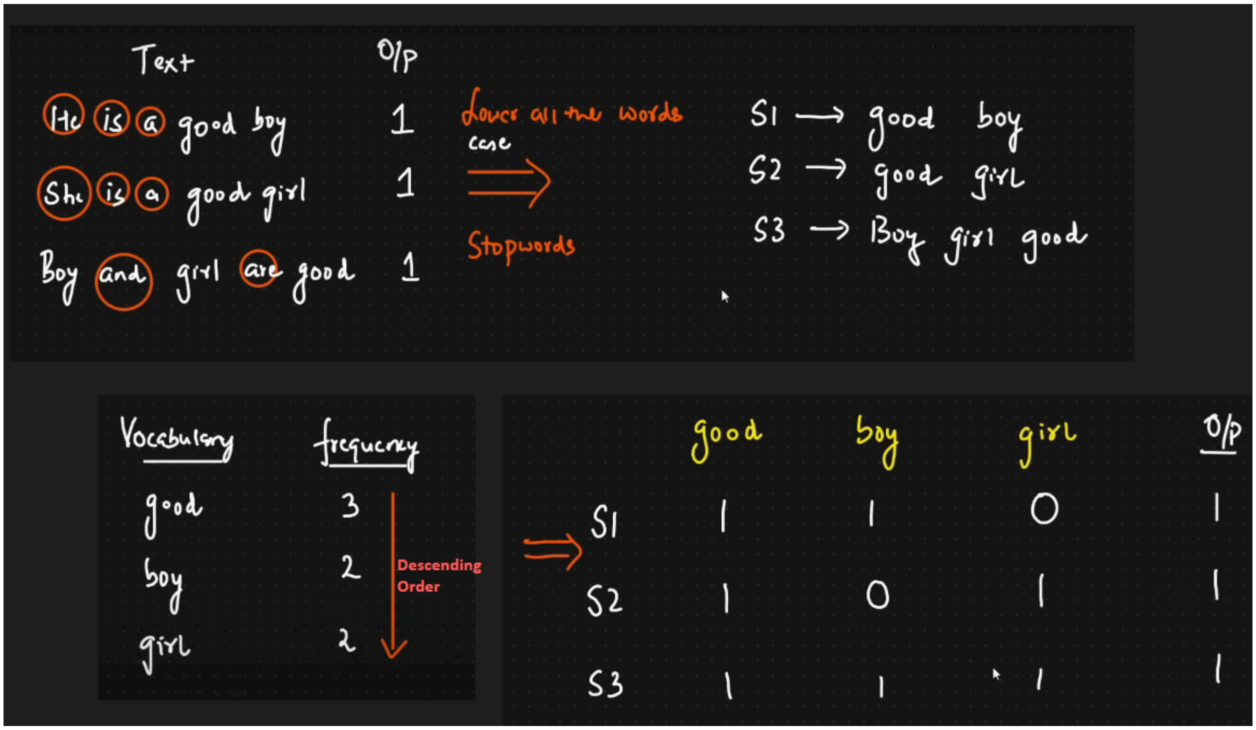
Word	Count
I	2
love	2

Word	Count
pizza	1
and	1
burgers	1

Now the sentence becomes a **vector of numbers**, not words.

How BoW Works?

1. **Tokenization**: Split text into words
2. **Vocabulary Creation**: Build a unique word dictionary
3. **Vectorization**: Count word occurrences in each document



Text	O/P
He is a good boy	1
She is a good girl	1
Boy and girl are good	1

S1 → good boy

S2 → good girL

S3 → Boy girl good

Vocabulary	frequency
good	3
boy	2
girL	2

	good	boy	girL	O/P
S1	1	1	0	1
S2	1	0	1	1
S3	1	1	1	1

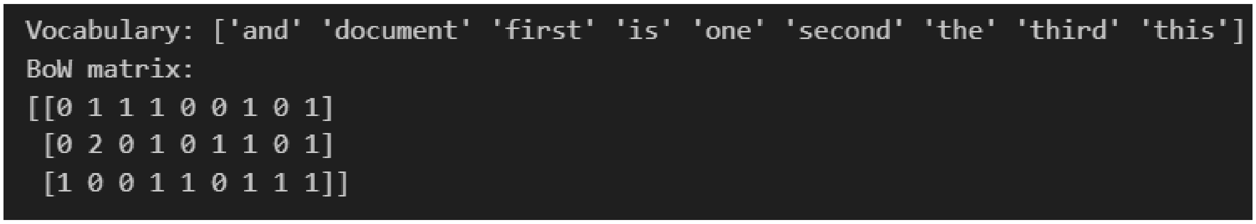
Python Code for BoW

```
from sklearn.feature_extraction.text import CountVectorizer

documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one."
]

# Initialize vectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents)

# Results
print("Vocabulary:", vectorizer.get_feature_names_out())
print("BoW matrix:")
print(X.toarray())  # Convert sparse matrix to dense array
```



```
Vocabulary: ['and' 'document' 'first' 'is' 'one' 'second' 'the' 'third' 'this']
BoW matrix:
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]]
```

💡 **In BoW, `"pizza I love"` and `"I love pizza"` → same vector.**

✅ Where BoW is Useful?

- Text classification (spam detection, sentiment analysis)
- Simple document similarity tasks
- Quick prototyping before trying advanced methods

⚠️ Limitations of BoW

Limitation	Description
❌ Ignores Order	"**dog bites man**" vs "**man bites dog**" appear identical
❌ No Context	Doesn’t understand meaning or sarcasm
❌ Sparse Matrix	For big vocabularies, most values are 0
❌ Vocabulary Grows	New words → bigger vectors, less efficient

✨Advantages:

- Simple & intuitive
- Fixed sized input

🔄 BoW vs One-Hot vs TF-IDF

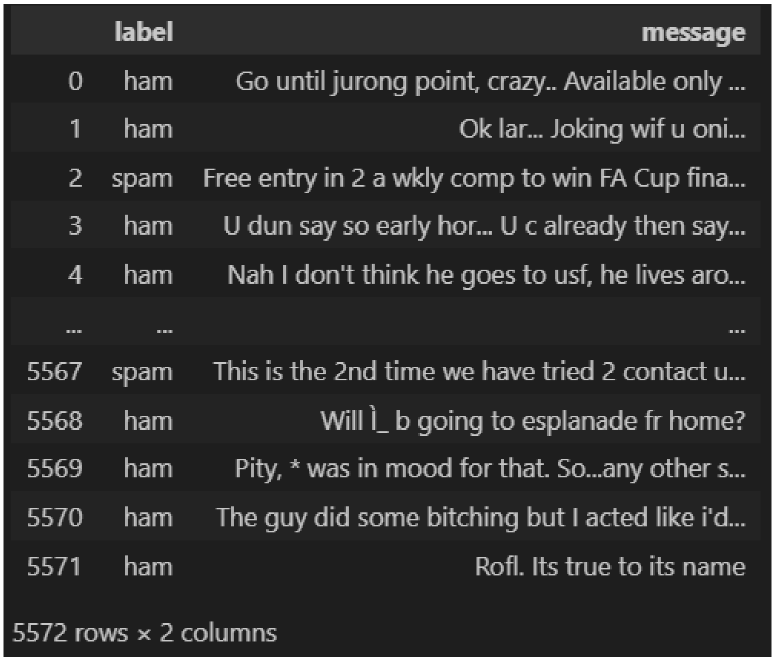
Feature	One-Hot	Bag of Words	TF-IDF
Vector Type	Binary (0/1)	Frequency Count	Weighted Frequency
Captures Count?	❌ No	✅ Yes	✅ Yes
Common Words	All treated same	All treated same	Penalizes common words
Output Size	Vocab size	Vocab size	Vocab size

BoW Practical

```
import pandas as pd

messages= pd.read_csv(r"https://raw.githubusercontent.com/shrudex/sms-spam-detection/refs/heads/main/sms-spam.csv")
messages.dropna(inplace=True, axis=1)
messages.columns = ["label", "message"]
#messages['label'] = messages['label'].map({'ham': 0, 'spam': 1})
```

```
messages
```



	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will Ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

Data Cleaning And Preprocessing

```
## Data Cleaning And Preprocessing
import re
import nltk
#nltk.download('stopwords')
```

```
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
ps=PorterStemmer()
```

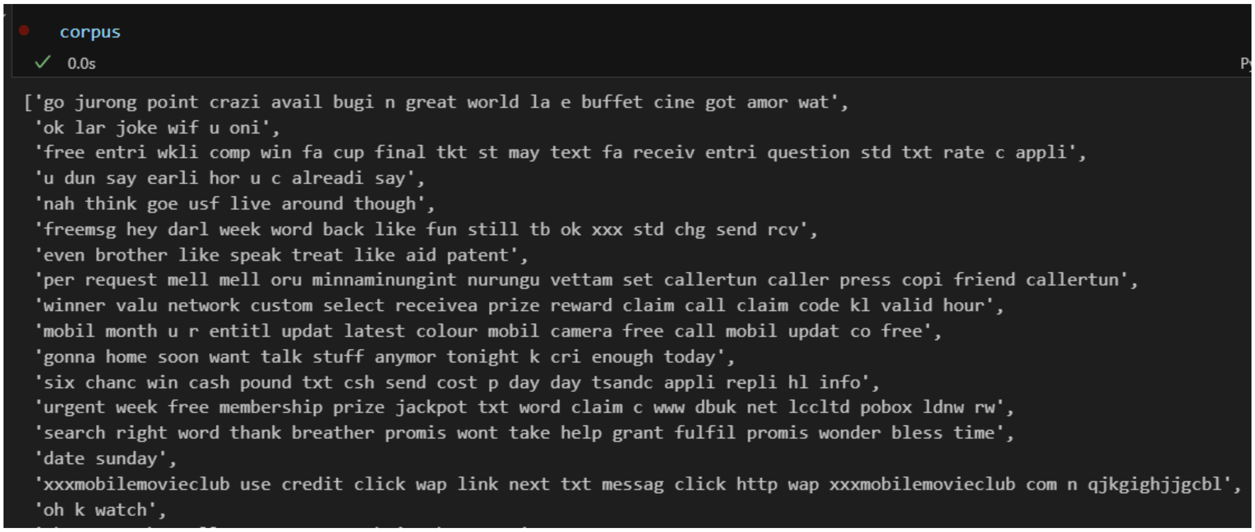
```
corpus= []

for i in range(len(messages)):
    review= re.sub('[^a-zA-z]',' ',messages['message'][i])
    review= review.lower()
    review= review.split() # Splits the review by words
    review= [ps.stem(word) for word in review if not word in stopwords.words
('english')]
    review= ' '.join(review)
    corpus.append(review)
```

`re.sub('[^a-zA-z]',' ',messages['message'][i])`

- Replace any character except A to Z with blank
    - Removes special characters
- `re. sub ()` → Regular expression **substitution**
- `[^...]` → "not" these characters

> Replace everything that is **not a letter (a–z or A–Z)** with a space.

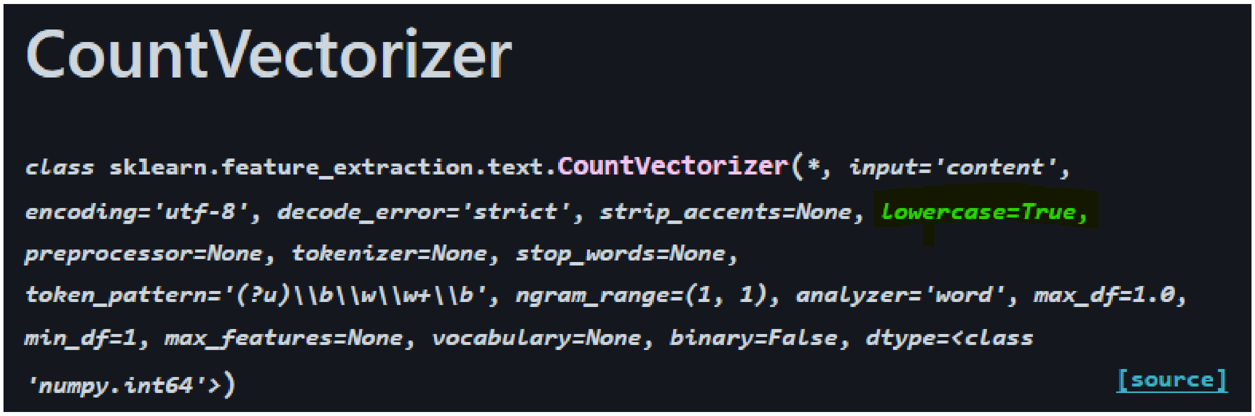


```
corpus
✓ 0.0s
['go jurong point crazi avail bugi n great world la e buffet cine got amor wat',
 'ok lar joke wif u oni',
 'free entri wkli comp win fa cup final tkt st may text fa receiv entri question std txt rate c appli',
 'u dun say earli hor u c alreadi say',
 'nah think goe usf live around though',
 'freemsg hey darl week word back like fun still tb ok xxx std chg send rcv',
 'even brother like speak treat like aid patent',
 'per request mell mell oru minnaminungint nurungu vettam set callertun caller press copi friend callertun',
 'winner valu network custom select receivea prize reward claim call claim code kl valid hour',
 'mobil month u r entitl updat latest colour mobil camera free call mobil updat co free',
 'gonna home soon want talk stuff anymor tonight k cri enough today',
 'six chanc win cash pound txt csh send cost p day day tsandc appli repli hl info',
 'urgent week free membership prize jackpot txt word claim c www dbuk net lccltd pobox ldnw rw',
 'search right word thank breather promis wont take help grant fulfil promis wonder bless time',
 'date sunday',
 'xxxmobilemovieclub use credit click wap link next txt messag click http wap xxxmobilemovieclub com n qjkgighjjgcbl',
 'oh k watch',
```

Create Bag of Words

```
from sklearn.feature_extraction.text import CountVectorizer
## for Binary BOW enable binary=True
```

```
cv=CountVectorizer(max_features=100,binary=True)
```



CountVectorizer

```
class sklearn.feature_extraction.text.CountVectorizer(*, input='content',
encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True,
preprocessor=None, tokenizer=None, stop_words=None,
token_pattern='(?u)\\b\\w\\w+\\b', ngram_range=(1, 1), analyzer='word', max_df=1.0,
min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class
'numpy.int64'>)
```

[source]

> 💡 `CountVectorizer` **already has a lowercase parameter. You don't need to do** `.lower()`

`max_features=100`

- Limits the number of words (features) in the vocabulary to the **top 100 most frequent** across the entire dataset.
- Helps reduce dimensionality and focus on the most important words.

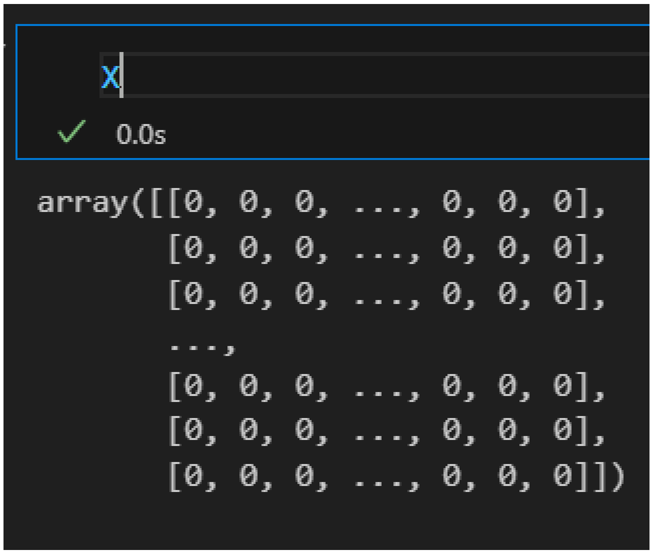
`binary=True`

- Instead of counting word occurrences, it returns:
  - **1** if the word is **present** in the document
  - **0** if it's **absent**
  - This makes it a **binary presence/absence matrix** — useful when word frequency isn't important (e.g., spam detection, sentiment classification).

```
X=cv.fit_transform(corpus).toarray()
```

- `.toarray()`:

- Converts the **sparse matrix** into a **dense NumPy array** (i.e., a regular 2D array).



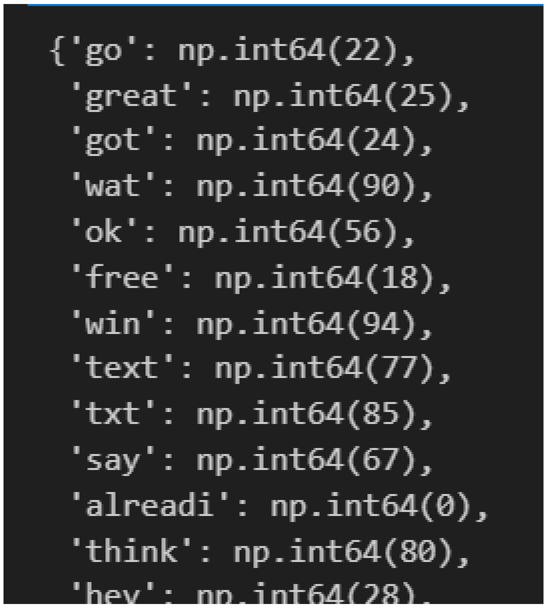
```
X
```

✓ 0.0s

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

Vocabulary:

```
cv.vocabulary_
```



```
{'go': np.int64(22),
 'great': np.int64(25),
 'got': np.int64(24),
 'wat': np.int64(90),
 'ok': np.int64(56),
 'free': np.int64(18),
 'win': np.int64(94),
 'text': np.int64(77),
 'txt': np.int64(85),
 'say': np.int64(67),
 'alreadi': np.int64(0),
 'think': np.int64(80),
 'hey': np.int64(28),
```

N-grams

🔠 What Are N-Grams?

An **N-gram** is a **sequence of N words** from a given text.

N-Gram Type	Example from "I love cats"
Unigram (1-gram)	`['I', 'love', 'cats']`
Bigram (2-gram)	`['I love', 'love cats']`
Trigram (3-gram)	`['I love cats']`

✅ Why Use N-Grams?

They help in:

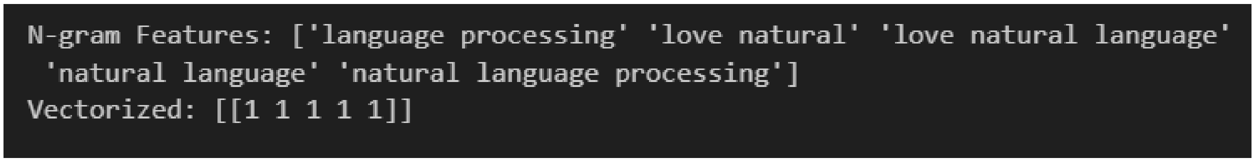
- **Text classification** (e.g., spam detection)
- **Language modeling** (e.g., predictive typing)
- **Feature extraction** for ML models
- Understanding **word context** or collocations

```
from sklearn.feature_extraction.text import CountVectorizer

text = ["I love natural language processing"]

# Use bigrams and trigrams
vectorizer = CountVectorizer(ngram_range=(2, 3))
X = vectorizer.fit_transform(text)

print("N-gram Features:", vectorizer.get_feature_names_out())
print("Vectorized:", X.toarray())
```



```
N-gram Features: ['language processing' 'love natural' 'love natural language'
 'natural language' 'natural language processing']
Vectorized: [[1 1 1 1 1]]
```

📌 `ngram_range=(N, M)` explained:

Parameter	Meaning
`(1, 1)`	Unigrams
`(1, 2)`	Unigrams + Bigrams
`(2, 2)`	Bigrams only
`(2, 3)`	Bigrams and Trigrams

🧠 When to Use What?

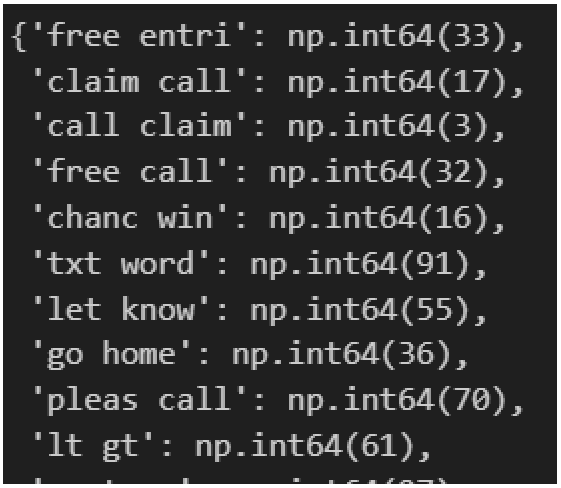
N-Gram Type	Use Case
Unigram	Basic feature extraction
Bigram	Captures simple word pairs (e.g., "not good")
Trigram+	Useful in complex contexts or phrase modeling

🚫 Limitations of N-Grams

- **Higher N = higher dimensionality**
- Doesn't capture **long-range dependencies**
- Can be very **sparse** with small data

Same code with Ngram:

```
## Create the Bag OF Words model with ngram
from sklearn.feature_extraction.text import CountVectorizer
## for Binary BOW enable binary=True
cv=CountVectorizer(max_features=100,binary=True,ngram_range=(2,3))
X=cv.fit_transform(corpus).toarray()
```

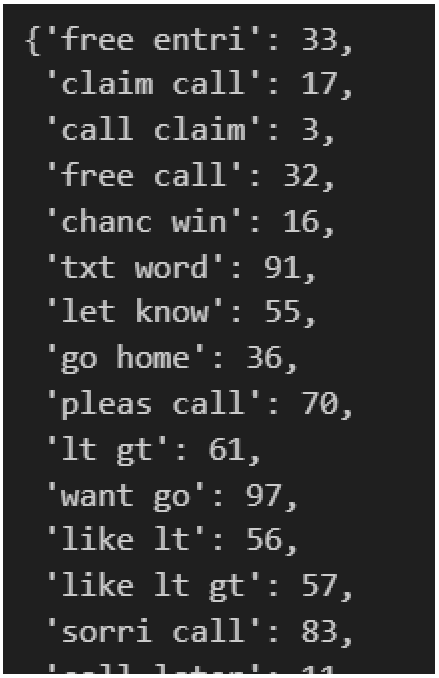


```
{'free entri': np.int64(33),
 'claim call': np.int64(17),
 'call claim': np.int64(3),
 'free call': np.int64(32),
 'chanc win': np.int64(16),
 'txt word': np.int64(91),
 'let know': np.int64(55),
 'go home': np.int64(36),
 'pleas call': np.int64(70),
 'lt gt': np.int64(61),
```

> 💡 **Numbers are the indices and not number of occurances.**

Get rid of `int64`:

```
vocab2 = {word: int(idx) for word, idx in cv.vocabulary_.items()}
vocab2
```



```
{'free entri': 33,
 'claim call': 17,
 'call claim': 3,
 'free call': 32,
 'chanc win': 16,
 'txt word': 91,
 'let know': 55,
 'go home': 36,
 'pleas call': 70,
 'lt gt': 61,
 'want go': 97,
 'like lt': 56,
 'like lt gt': 57,
 'sorri call': 83,
```

💡 **Try increasing `ngram_range` if you’re not getting desired accuracy.**

Also increase the `max_features` along with `ngram_range.`