

# Building a bag of words model

FEATURE ENGINEERING FOR NLP IN PYTHON



**Rounak Banik**  
Instructor

# Recap of data format for ML algorithms

For any ML algorithm,

- Data must be in tabular form
- Training features must be numerical

# Bag of words model

- Extract word tokens
- Compute frequency of word tokens
- Construct a word vector out of these frequencies and vocabulary of corpus

# Bag of words model example

## Corpus

"The lion is the king of the jungle"

"Lions have lifespans of a decade"

"The lion is an endangered species"

# Bag of words model example

Vocabulary → a , an , decade , endangered , have , is , jungle , king , lifespans , lion , Lions , of , species , the , The

```
"The lion is the king of the jungle"
```

```
[0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 2, 1]
```

```
"Lions have lifespans of a decade"
```

```
[1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0]
```

```
"The lion is an endangered species"
```

```
"The lion is an endangered species"
```

# Text preprocessing

- `Lions` , `lion` → `lion`
- `The` , `the` → `the`
- No punctuations
- No stopwords
- Leads to smaller vocabularies
- Reducing number of dimensions helps improve performance

# Bag of words model using sklearn

```
corpus = pd.Series([  
    'The lion is the king of the jungle',  
    'Lions have lifespans of a decade',  
    'The lion is an endangered species'  
])
```

# Bag of words model using sklearn

```
# Import CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer

# Create CountVectorizer object
vectorizer = CountVectorizer()

# Generate matrix of word vectors
bow_matrix = vectorizer.fit_transform(corpus)
print(bow_matrix.toarray())
```

```
array([[0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 3],
       [0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0],
       [1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1]], dtype=int64)
```



# Let's practice!

FEATURE ENGINEERING FOR NLP IN PYTHON

# Building a BoW Naive Bayes classifier

FEATURE ENGINEERING FOR NLP IN PYTHON



Rounak Banik  
Instructor

# Spam filtering

message	label	
WINNER!! As a valued network customer you have been selected to receive a \$900 prize reward! To claim call 09061701461	spam	
Ah, work. I vaguely remember that. What does it feel like?	ham	

# Steps

1. Text preprocessing
2. Building a bag-of-words model (or representation)
3. Machine learning

# Text preprocessing using CountVectorizer

## CountVectorizer arguments

- `lowercase` : `False` , `True`
- `strip_accents` : `'unicode'` , `'ascii'` , `None`
- `stop_words` : `'english'` , `list` , `None`
- `token_pattern` : `regex`
- `tokenizer` : `function`

# Building the BoW model

```
# Import CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer

# Create CountVectorizer object
vectorizer = CountVectorizer(strip_accents='ascii', stop_words='english', lowercase=False)

# Import train_test_split
from sklearn.model_selection import train_test_split

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df['message'], df['label'], test_size=0.25)
```

# Building the BoW model

```
...  
...  
# Generate training Bow vectors  
X_train_bow = vectorizer.fit_transform(X_train)  
  
# Generate test BoW vectors  
X_test_bow = vectorizer.transform(X_test)
```

# Training the Naive Bayes classifier

```
# Import MultinomialNB
from sklearn.naive_bayes import MultinomialNB

# Create MultinomialNB object
clf = MultinomialNB()

# Train clf
clf.fit(X_train_bow, y_train)

# Compute accuracy on test set
accuracy = clf.score(X_test_bow, y_test)
print(accuracy)
```

```
0.760051
```

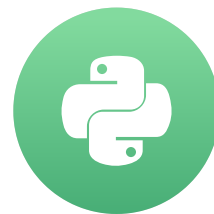


# Let's practice!

FEATURE ENGINEERING FOR NLP IN PYTHON

# Building n-gram models

FEATURE ENGINEERING FOR NLP IN PYTHON



**Rounak Banik**  
Instructor

# BoW shortcomings

review	label	
'The movie was good and not boring'	positive	
'The movie was not good and boring'	negative	

- Exactly the same BoW representation!
- Context of the words is lost.
- Sentiment dependent on the position of 'not'.

# n-grams

- Contiguous sequence of  $n$  elements (or words) in a given document.
- $n = 1 \rightarrow$  bag-of-words

```
'for you a thousand times over'
```

- $n = 2$ , n-grams:

```
[  
'for you',  
'you a',  
'a thousand',  
'thousand times',  
'times over'  
]
```

# n-grams

```
'for you a thousand times over'
```

- $n = 3$ , n-grams:

```
[  
  'for you a',  
  'you a thousand',  
  'a thousand times',  
  'thousand times over'  
]
```

- Captures more context.

# Applications

- Sentence completion
- Spelling correction
- Machine translation correction

# Building n-gram models using scikit-learn

Generates only bigrams.

```
bigrams = CountVectorizer(ngram_range=(2,2))
```

Generates unigrams, bigrams and trigrams.

```
ngrams = CountVectorizer(ngram_range=(1,3))
```

# Shortcomings

- Curse of dimensionality
- Higher order n-grams are rare
- Keep n small



# Let's practice!

FEATURE ENGINEERING FOR NLP IN PYTHON