Natural Language Processing

ACTL3143 & ACTL5111 Deep Learning for Actuaries

Patrick Laub

Lecture Outline

- **Natural Language Processing**
- Car Crash Police Reports
- Text Vectorisation
- Bag Of Words
- Limiting The Vocabulary
- Intelligently Limit The Vocabulary

What is NLP?

A field of research at the intersection of computer science, linguistics, and artificial intelligence that takes the **naturally spoken or written language** of humans and **processes it with machines** to automate or help in certain tasks.

How the computer sees text

Spot the odd one out:

```
[112, 97, 116, 114, 105, 99, 107, 32, 108, 97, 117, 98]
[80, 65, 84, 82, 73, 67, 75, 32, 76, 65, 85, 66]
[76, 101, 118, 105, 32, 65, 99, 107, 101, 114, 109, 97, 110]
```

Generated by:

```python
print([ord(x) for x in "patrick laub"])
print([ord(x) for x in "PATRICK LAUB"])
print([ord(x) for x in "Levi Ackerman"])
```

The `ord` built-in turns characters into their ASCII form.

Question

The largest value for a character is 127, can you guess why?

ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

American Standard Code for Information Interchange

Unicode is the new standard.

Source: Wikipedia

Random strings

The built-in `chr` function turns numbers into characters.

```
rnd.seed(1)
```

```
chars = [chr(rnd.randint(32, 127)) for _ in range(10)]
chars
```

```
['E', ',', 'h', ')', 'k', '%', 'o', '`', '0', '!']
```

```
" ".join(chars)
```

```
'E , h ) k % o ` 0 !'
```

```
"".join([chr(rnd.randint(32, 127)) for _ in range(50)])
```

```
"lg&9R42t+⩽.Rdww~v-)'_]6Y! \\q(x-Oh>g#f5QY#d8Kl:TpI"
```

```
"".join([chr(rnd.randint(0, 128)) for _ in range(50)])
```

```
'R\x0f@D\x19obW\x07\x1a\x19h\x16\tCg~\x17}d\x1b%9S&\x08 "\n\x17\x0foW\x19Gs\\J>.
X\x177AqM\x03\x00x'
```

Escape characters

```
1 print("Hello,\tworld!")
```

```
Hello,	world!
```

```
1 print("Line 1\nLine 2")
```

```
Line 1
Line 2
```

```
1 print("Patrick\rLaub")
```

```
Laubick
```

```
1 print("C:\tom\new folder")
```

```
C:	om
ew folder
```

Escape the backslash:

```
1 print("C:\\tom\\new folder")
```

```
C:\tom\new folder
```

```
1 repr("Hello,\rworld!")
```

```
"'Hello,\\rworld!'"
```

Non-natural language processing I

How would you evaluate

> 10 + 2 * -3

All that Python sees is a string of characters.

```
1 [ord(c) for c in "10 + 2 * -3"]
```

```
[49, 48, 32, 43, 32, 50, 32, 42, 32, 45, 51]
```

```
1 10 + 2 * -3
```

```
4
```

Non-natural language processing II

Python first tokenizes the string:

```
1 import tokenize
2 import io
3
4 code = "10 + 2 * -3"
5 tokens = tokenize.tokenize(io.BytesIO(code.encode("utf-8")).readline)
6 for token in tokens:
7     print(token)
```

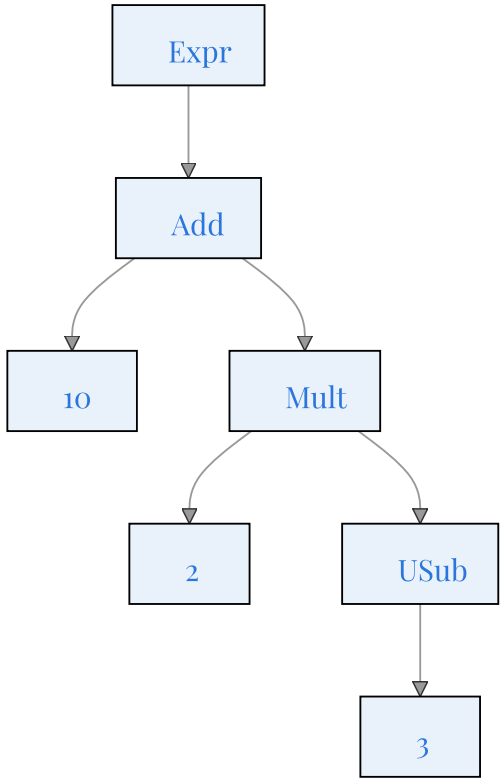
```
TokenInfo(type=68 (ENCODING), string='utf-8', start=(0, 0), end=(0, 0), line='')
TokenInfo(type=2 (NUMBER), string='10', start=(1, 0), end=(1, 2), line='10 + 2 * -3')
TokenInfo(type=55 (OP), string='+', start=(1, 3), end=(1, 4), line='10 + 2 * -3')
TokenInfo(type=2 (NUMBER), string='2', start=(1, 5), end=(1, 6), line='10 + 2 * -3')
TokenInfo(type=55 (OP), string='*', start=(1, 7), end=(1, 8), line='10 + 2 * -3')
TokenInfo(type=55 (OP), string='-', start=(1, 9), end=(1, 10), line='10 + 2 * -3')
TokenInfo(type=2 (NUMBER), string='3', start=(1, 10), end=(1, 11), line='10 + 2 * -3')
TokenInfo(type=4 (NEWLINE), string='', start=(1, 11), end=(1, 12), line='10 + 2 * -3')
TokenInfo(type=0 (ENDMARKER), string='', start=(2, 0), end=(2, 0), line='')
```

Non-natural language processing III

Python needs to *parse* the tokens into an abstract syntax tree.

```python
import ast

print(ast.dump(ast.parse("10 + 2 * -3"), indent="  "))
```

```
Module(
  body=[
    Expr(
      value=BinOp(
        left=Constant(value=10),
        op=Add(),
        right=BinOp(
          left=Constant(value=2),
          op=Mult(),
          right=UnaryOp(
            op=USub(),
            operand=Constant(value=3)))))])
```

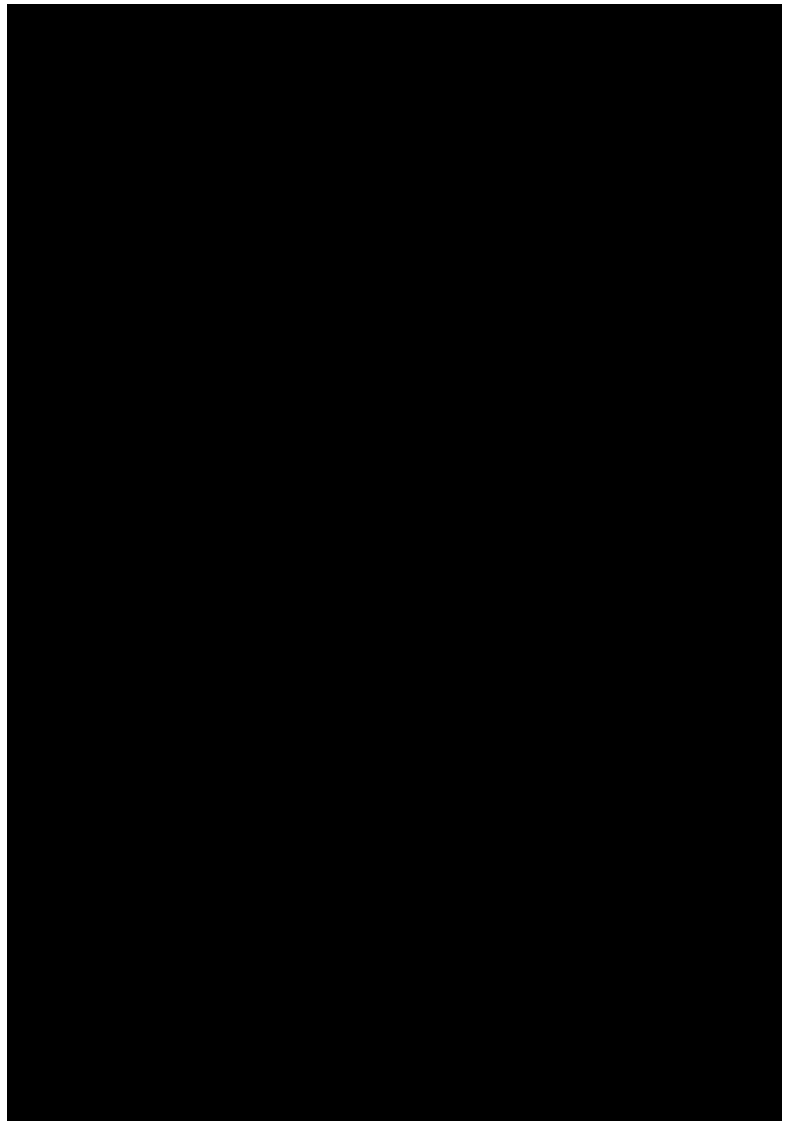


Non-natural language processing IV

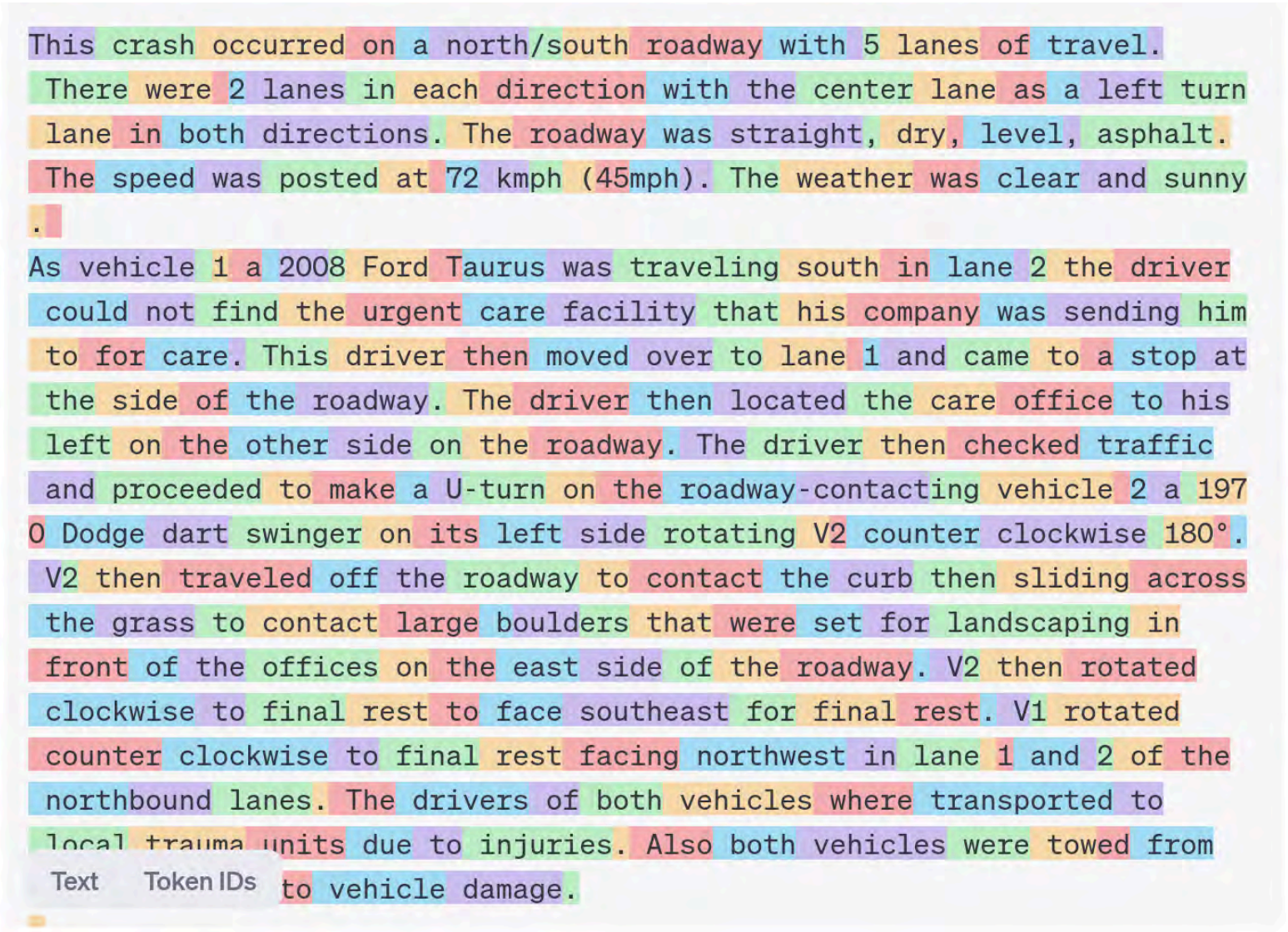
The abstract syntax tree is then compiled into bytecode.

```python
import dis

def expression(a, b, c):
    return a + b * -c

dis.dis(expression)
```

```
  3           RESUME                   0

  4           LOAD_FAST_BORROW_LOAD_FAST_BORROW 1 (a,
b)
              LOAD_FAST_BORROW         2 (c)
              UNARY_NEGATIVE
              BINARY_OP                5 (*)
              BINARY_OP                0 (+)
              RETURN_VALUE
```



ChatGPT tokenization



Source: https://platform.openai.com/tokenizer

E.g. radical for animals

- gǒu (dog)
- māo (cat)
- láng (wolf)
- shī (lion)

Applications of NLP in Industry

1) Classifying documents: Using the language within a body of text to classify it into a particular category, e.g.:

- Grouping emails into high and low urgency
- Movie reviews into positive and negative sentiment (i.e. *sentiment analysis*)
- Company news into bullish (positive) and bearish (negative) statements

2) Machine translation: Assisting language translators with machine-generated suggestions from a source language (e.g. English) to a target language

Applications of NLP in Industry II

3) Search engine functions, including:

- Autocomplete
- Predicting what information or website user is seeking

4) Speech recognition: Interpreting voice commands to provide information or take action. Used in virtual assistants such as Alexa, Siri, and Cortana

Deep learning & NLP?

Simple NLP applications such as spell checkers and synonym suggesters **do not require deep learning** and can be solved with **deterministic, rules-based code** with a dictionary/thesaurus.

More complex NLP applications such as classifying documents, search engine word prediction, and chatbots are complex enough to be solved using deep learning methods.

NLP in 1966–1973 I

> A typical story occurred in early machine translation efforts, which were generously funded by the U.S. National Research Council in an attempt to speed up the translation of Russian scientific papers in the wake of the Sputnik launch in 1957. It was thought initially that simple syntactic transformations, based on the grammars of Russian and English, and word replacement from an electronic dictionary, would suffice to preserve the exact meanings of sentences.

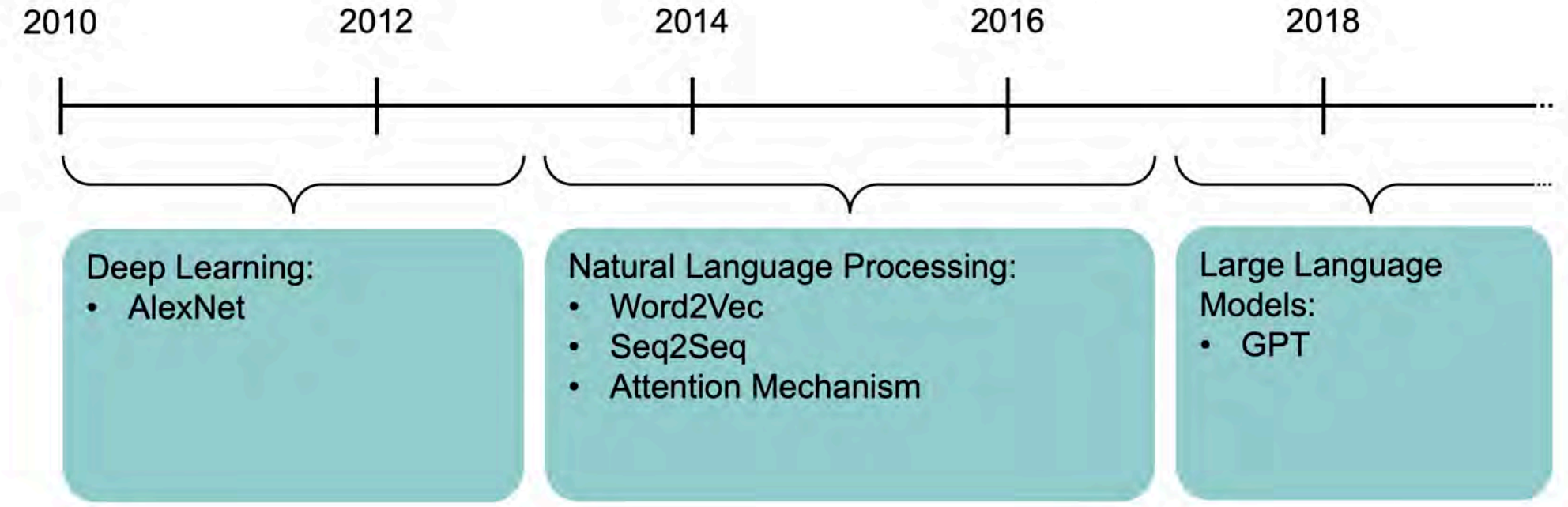
Source: Russell and Norvig (2016), *Artificial Intelligence: A Modern Approach*, Third Edition, p. 21.

NLP in 1966–1973 II

> The fact is that accurate translation requires background knowledge in order to resolve ambiguity and establish the content of the sentence. The famous retranslation of **"the spirit is willing but the flesh is weak"** as **"the vodka is good but the meat is rotten"** illustrates the difficulties encountered. In 1966, a report by an advisory committee found that "there has been no machine translation of general scientific text, and none is in immediate prospect." All U.S. government funding for academic translation projects was canceled.

Source: Russell and Norvig (2016), *Artificial Intelligence: A Modern Approach*, Third Edition, p. 21.

High-level history of deep learning



A brief history of deep learning.

Source: Melissa Renard (2025)

Lecture Outline

- Natural Language Processing
- **Car Crash Police Reports**
- Text Vectorisation
- Bag Of Words
- Limiting The Vocabulary
- Intelligently Limit The Vocabulary

Downloading the dataset

Look at the (U.S.) National Highway Traffic Safety Administration's (NHTSA) National Motor Vehicle Crash Causation Survey (NMVCCS) dataset.

```
from pathlib import Path

if not Path("../data/NHTSA_NMVCCS_extract.parquet.gzip").exists():
    print("Downloading dataset")
    !wget -P ../data https://github.com/JSchelldorfer/ActuarialDataScience/raw/master/12%

df = pd.read_parquet("../data/NHTSA_NMVCCS_extract.parquet.gzip")
print(f"shape of DataFrame: {df.shape}")
```

```
shape of DataFrame: (6949, 16)
```

Features

- `level_0`, `index`, `SCASEID`: all useless row numbers
- `SUMMARY_EN` and `SUMMARY_GE`: summaries of the accident
- `NUMTOTV`: total number of vehicles involved in the accident
- `WEATHER1` to `WEATHER8` (**not one-hot**):
  - `WEATHER1`: cloudy
  - `WEATHER2`: snow
  - `WEATHER3`: fog, smog, smoke
  - `WEATHER4`: rain
  - `WEATHER5`: sleet, hail (freezing drizzle or rain)
  - `WEATHER6`: blowing snow
  - `WEATHER7`: severe crosswinds
  - `WEATHER8`: other
- `INJSEVA` and `INJSEVB`: injury severity & (binary) presence of bodily injury

Source: JSchelldorfer's GitHub.

Crash summaries

```
1 df["SUMMARY_EN"]
```

```
0       V1, a 2000 Pontiac Montana minivan, made a lef ...
1       The crash occurred in the eastbound lane of a  ...
2       This crash occurred just after the noon time h ...
                              ...
6946    The crash occurred in the eastbound lanes of a ...
6947    This single-vehicle crash occurred in a rural  ...
6948    This two vehicle daytime collision occurred mi ...
Name: SUMMARY_EN, Length: 6949, dtype: str
```

```
1 df["SUMMARY_EN"].map(lambda summary: len(summary)).hist(grid=False);
```

A crash summary

```
1 df["SUMMARY_EN"].iloc[1]
```

"The crash occurred in the eastbound lane of a two-lane, two-way asphalt roadway on level grade. The conditions were daylight and wet with cloudy skies in the early afternoon on a weekday.\t\r \r V1, a 1995 Chevrolet Lumina was traveling eastbound. V2, a 2004 Chevrolet Trailblazer was also traveling eastbound on the same roadway. V2, was attempting to make a left-hand turn into a private drive on the North side of the roadway. While turning V1 attempted to pass V2 on the left-hand side contacting it's front to the left side of V2. Both vehicles came to final rest on the roadway at impact.\r \r The driver of V1 fled the scene and was not identified, so no further information could be obtained from him. The Driver of V2 stated that the driver was a male and had hit his head and was bleeding. She did not pursue the driver because she thought she saw a gun. The officer said that the car had been reported stolen.\r \r The Critical Precrash Event for the driver of V1 was this vehicle traveling over left lane line on the left side of travel. The Critical Reason for the Critical Event was coded as unknown reason for the critical event because the driver was not available. \r \r The driver of V2 was a 41-year old female who had reported that she had stopped prior to turning to make sure she was at the right house. She was going to show a house for a client. She had no health related problems. She had taken amoxicillin. She does not wear corrective lenses

Carriage returns

```
1 print(df["SUMMARY_EN"].iloc[1])
```

The Critical Precrash Event for the driver of V2 was other vehicle encroachment from adjacent lane over left lane line. The Critical Reason for the Critical Event was not coded for this vehicle and the driver of V2 was not thought to have contributed to the crash.r corrective lenses and felt rested. She was not injured in the crash. of V2. Both vehicles came to final rest on the roadway at impact.

```
1 # Replace every \r with \n
2 def replace_carriage_return(summary):
3     return summary.replace("\r", "\n")
4 
5 df["SUMMARY_EN"] = df["SUMMARY_EN"].map(replace_carriage_return)
6 print(df["SUMMARY_EN"].iloc[1][:500])
```

The crash occurred in the eastbound lane of a two-lane, two-way asphalt roadway on level grade. The conditions were daylight and wet with cloudy skies in the early afternoon on a weekday.

 V1, a 1995 Chevrolet Lumina was traveling eastbound. V2, a 2004 Chevrolet Trailblazer was also traveling eastbound on the same roadway. V2, was attempting to make a left-hand turn into a private drive on the North side of the roadway. While turning V1 attempted to pass V2 on the left-hand side contactin

Target

Predict number of vehicles in the crash.

```
df["NUMTOTV"].value_counts()\
    .sort_index()
```

```
NUMTOTV
1    1822
2    4151
3     783
4     150
5      34
6       5
7       2
8       1
9       1
Name: count, dtype: int64
```

```
np.sum(df["NUMTOTV"] > 3)
```

```
np.int64(193)
```

Simplify the target to just:

- 1 vehicle
- 2 vehicles
- 3+ vehicles

```
df["NUM_VEHICLES"] = \
  df["NUMTOTV"].map(lambda x: \
    str(x) if x ⩽ 2 else "3+")
df["NUM_VEHICLES"].value_counts()\
  .sort_index()
```

```
NUM_VEHICLES
1     1822
2     4151
3+     976
Name: count, dtype: int64
```

Just ignore this for now...

```
rnd.seed(123)

for i, summary in enumerate(df["SUMMARY_EN"]):
    word_numbers = ["one", "two", "three", "four", "five", "six", "seven", "eight", "nine
    num_cars = 10
    new_car_nums = [f"V{rnd.randint(100, 10000)}" for _ in range(num_cars)]
    num_spaces = 4

    for car in range(1, num_cars+1):
        new_num = new_car_nums[car-1]
        summary = summary.replace(f"V-{car}", new_num)
        summary = summary.replace(f"Vehicle {word_numbers[car-1]}", new_num).replace(f"ve
        summary = summary.replace(f"Vehicle #{word_numbers[car-1]}", new_num).replace(f"v
        summary = summary.replace(f"Vehicle {car}", new_num).replace(f"vehicle {car}", ne
        summary = summary.replace(f"Vehicle #{car}", new_num).replace(f"vehicle #{car}",
        summary = summary.replace(f"Vehicle # {car}", new_num).replace(f"vehicle # {car}"

        for j in range(num_spaces+1):
            summary = summary.replace(f"V{' '*j}{car}", new_num).replace(f"V{' '*j}#{car}
            summary = summary.replace(f"v{' '*j}{car}", new_num).replace(f"v{' '*j}#{car}

    df.loc[i, "SUMMARY_EN"] = summary
```

Convert y to integers & split the data

```
from sklearn.preprocessing import LabelEncoder
target_labels = df["NUM_VEHICLES"]
target = LabelEncoder().fit_transform(target_labels)
target
```

```
array([1, 1, 1, ..., 2, 0, 1], shape=(6949,))
```

```
weather_cols = [f"WEATHER{i}" for i in range(1, 9)]
features = df[["SUMMARY_EN"] + weather_cols]

X_main, X_test, y_main, y_test = \
    train_test_split(features, target, test_size=0.2, random_state=1)

# As 0.25 x 0.8 = 0.2
X_train, X_val, y_train, y_val = \
    train_test_split(X_main, y_main, test_size=0.25, random_state=1)

X_train.shape, X_val.shape, X_test.shape
```

```
((4169, 9), (1390, 9), (1390, 9))
```

```
print([np.mean(y_train == y) for y in [0, 1, 2]])
```

```
[np.float64(0.25833533221396016), np.float64(0.6032621731830176),
np.float64(0.1384024946030223)]
```

Lecture Outline

- Natural Language Processing
- Car Crash Police Reports
- **Text Vectorisation**
- Bag Of Words
- Limiting The Vocabulary
- Intelligently Limit The Vocabulary

Grab the start of a few summaries

```
1 first_summaries = X_train["SUMMARY_EN"].iloc[:3]
2 first_summaries
```

```
2532    This crash occurred in the early afternoon of  ...
6209    This two-vehicle crash occurred in a four-legg ...
2561    The crash occurred in the eastbound direction  ...
Name: SUMMARY_EN, dtype: str
```

```
1 first_words = first_summaries.map(lambda txt: txt.split(" ")[:7])
2 first_words
```

```
2532    [This, crash, occurred, in, the, early, aftern ...
6209    [This, two-vehicle, crash, occurred, in, a, fo ...
2561    [The, crash, occurred, in, the, eastbound, dir ...
Name: SUMMARY_EN, dtype: object
```

```
1 start_of_summaries = first_words.map(lambda txt: " ".join(txt))
2 start_of_summaries
```

```
2532          This crash occurred in the early afternoon
6209    This two-vehicle crash occurred in a four-legged
2561       The crash occurred in the eastbound direction
Name: SUMMARY_EN, dtype: str
```

Count words in the first summaries

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 vect = CountVectorizer()
4 counts = vect.fit_transform(start_of_summaries)
5 vocab = vect.get_feature_names_out()
6 print(len(vocab), vocab)
```

```
13 ['afternoon' 'crash' 'direction' 'early' 'eastbound' 'four' 'in' 'legged'
 'occurred' 'the' 'this' 'two' 'vehicle']
```

```
1 counts
```

```
<Compressed Sparse Row sparse matrix of dtype 'int64'
    with 21 stored elements and shape (3, 13)>
```

```
1 counts.toarray()
```

```
array([[1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0],
       [0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1],
       [0, 1, 1, 0, 1, 0, 1, 0, 1, 2, 0, 0, 0]])
```

Encode new sentences to BoW

```
vect.transform([
    "first car hit second car in a crash",
    "ipad os 26 beta released",
])
```

```
<Compressed Sparse Row sparse matrix of dtype 'int64'
    with 2 stored elements and shape (2, 13)>
```

```
vect.transform([
    "first car hit second car in a crash",
    "ipad os 18 beta released",
]).toarray()
```

```
array([[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

```
print(vocab)
```

```
['afternoon' 'crash' 'direction' 'early' 'eastbound' 'four' 'in' 'legged'
 'occurred' 'the' 'this' 'two' 'vehicle']
```

Bag of n-grams

```
1  vect = CountVectorizer(ngram_range=(1, 2))
2  counts = vect.fit_transform(start_of_summaries)
3  vocab = vect.get_feature_names_out()
4  print(len(vocab), vocab)
```

```
27 ['afternoon' 'crash' 'crash occurred' 'direction' 'early'
 'early afternoon' 'eastbound' 'eastbound direction' 'four' 'four legged'
 'in' 'in four' 'in the' 'legged' 'occurred' 'occurred in' 'the'
 'the crash' 'the early' 'the eastbound' 'this' 'this crash' 'this two'
 'two' 'two vehicle' 'vehicle' 'vehicle crash']
```

```
1  counts.toarray()
```

```
array([[1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
        0, 0, 0, 0, 0],
       [0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0,
        1, 1, 1, 1, 1],
       [0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 2, 1, 0, 1, 0, 0,
        0, 0, 0, 0, 0]])
```

See: Google Books Ngram Viewer

UNSW SYDNEY

TF-IDF

Stands for *term frequency–inverse document frequency*.

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

Infographic explaining TF-IDF

Source: FiloTechnologia (2014), A simple Java class for TF-IDF scoring, Blog post.

Lecture Outline

- Natural Language Processing
- Car Crash Police Reports
- Text Vectorisation
- **Bag Of Words**
- Limiting The Vocabulary
- Intelligently Limit The Vocabulary

Count words in all the summaries

```
1 vect = CountVectorizer()
2 vect.fit(X_train["SUMMARY_EN"])
3 vocab = list(vect.get_feature_names_out())
4 len(vocab)
```

```
18866
```

```
1 vocab[:5], vocab[len(vocab)//2:(len(vocab)//2 + 5)], vocab[-5:]
```

```
(['00', '000', '000lbs', '003', '005'],
 ['swinger', 'swinging', 'swipe', 'swiped', 'swiping'],
 ['zorcor', 'zotril', 'zx2', 'zx5', 'zyrtec'])
```

Create the X matrices

```python
def vectorise_dataset(X, vect, txt_col="SUMMARY_EN", dataframe=False):
    X_vects = vect.transform(X[txt_col]).toarray()
    X_other = X.drop(txt_col, axis=1)

    if not dataframe:
        return np.concatenate([X_vects, X_other], axis=1)
    else:
        # Add column names and indices to the combined dataframe.
        vocab = list(vect.get_feature_names_out())
        X_vects_df = pd.DataFrame(X_vects, columns=vocab, index=X.index)
        return pd.concat([X_vects_df, X_other], axis=1)
```

```python
X_train_bow = vectorise_dataset(X_train, vect)
X_val_bow = vectorise_dataset(X_val, vect)
X_test_bow = vectorise_dataset(X_test, vect)
```

Check the input matrix

```
1 vectorise_dataset(X_train, vect, dataframe=True)
```

	00	000	000lbs	003	005	007	00am	00pm	00tyd02	01	...	zx5
2532	0	0	0	0	0	0	0	0	0	0	...	0
6209	0	0	0	0	0	0	0	0	0	0	...	0
2561	0	0	0	0	0	0	0	0	0	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
6882	0	0	0	0	0	0	0	0	0	0	...	0
206	0	0	0	0	0	0	0	0	0	0	...	0
6356	0	0	0	0	0	0	0	0	0	0	...	0

4169 rows × 18874 columns

Make a simple dense model

```
num_features = X_train_bow.shape[1]
num_cats = 3 # 1, 2, 3+ vehicles

def build_model(num_features, num_cats):
    random.seed(42)

    model = Sequential([
        Input((num_features,)),
        Dense(100, activation="relu"),
        Dense(num_cats, activation="softmax")
    ])

    topk = SparseTopKCategoricalAccuracy(k=2, name="topk")
    model.compile("adam", "sparse_categorical_crossentropy",
        metrics=["accuracy", topk])

    return model
```

Inspect the model

```
1 model = build_model(num_features, num_cats)
2 model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 100)	1,887,500
dense_1 (Dense)	(None, 3)	303

Total params: 1,887,803 (7.20 MB)
Trainable params: 1,887,803 (7.20 MB)
Non-trainable params: 0 (0.00 B)

Fit & evaluate the model

```
es = EarlyStopping(patience=1, restore_best_weights=True,
    monitor="val_accuracy", verbose=2)
%time hist = model.fit(X_train_bow, y_train, epochs=10, \
    callbacks=[es], validation_data=(X_val_bow, y_val), verbose=0);
```

```
Epoch 2: early stopping
Restoring model weights from the end of the best epoch: 1.
CPU times: user 2.9 s, sys: 795 ms, total: 3.69 s
Wall time: 2.02 s
```

```
model.evaluate(X_train_bow, y_train, verbose=0)
```

```
[0.05817717686295509, 0.9901655316352844, 0.9995202422142029]
```

```
model.evaluate(X_val_bow, y_val, verbose=0)
```

```
[0.1902538239955902, 0.9503597021102905, 0.9942445755004883]
```

Lecture Outline

- Natural Language Processing
- Car Crash Police Reports
- Text Vectorisation
- Bag Of Words
- **Limiting The Vocabulary**
- Intelligently Limit The Vocabulary

The `max_features` value

```
1 vect = CountVectorizer(max_features=10)
2 vect.fit(X_train["SUMMARY_EN"])
3 vocab = vect.get_feature_names_out()
4 len(vocab)
```

```
10
```

```
1 print(vocab)
```

```
['and' 'driver' 'for' 'in' 'lane' 'of' 'the' 'to' 'vehicle' 'was']
```

What is left?

```python
for i in range(3):
    sentence = X_train["SUMMARY_EN"].iloc[i]
    for word in sentence.split(" ")[:10]:
        word_or_qn = word if word in vocab else "?"
        print(word_or_qn, end=" ")
    print() # Same as print("\n", end="")
```

```
? ? ? in the ? ? of ? ?
? ? ? ? in ? ? ? ? ?
? ? ? in the ? ? of ? ?
```

```python
for i in range(3):
    sentence = X_train["SUMMARY_EN"].iloc[i]
    num_words = 0
    for word in sentence.split(" "):
        if word in vocab:
            print(word, end=" ")
            num_words += 1
        if num_words == 10:
            break
    print()
```

```
in the of in the of of was and was
in and of in and for the of the and
in the of to was was of was was and
```

Remove stop words

```
vect = CountVectorizer(max_features=10, stop_words="english")
vect.fit(X_train["SUMMARY_EN"])
vocab = vect.get_feature_names_out()
len(vocab)
```

```
10
```

```
print(vocab)
```

```
['coded' 'crash' 'critical' 'driver' 'event' 'intersection' 'lane' 'left'
 'roadway' 'vehicle']
```

```
for i in range(3):
    sentence = X_train["SUMMARY_EN"].iloc[i]
    num_words = 0
    for word in sentence.split(" "):
        if word in vocab:
            print(word, end=" ")
            num_words += 1
        if num_words == 10:
            break
    print()
```

```
crash intersection roadway roadway roadway intersection lane lane intersection driver
crash roadway left roadway roadway roadway lane lane roadway crash
crash vehicle left left vehicle driver vehicle lane lane coded
```

Keep 1,000 most frequent words

```
vect = CountVectorizer(max_features=1_000, stop_words="english")
vect.fit(X_train["SUMMARY_EN"])
vocab = vect.get_feature_names_out()
len(vocab)
```

```
1000
```

```
print(vocab[:5], vocab[len(vocab)//2:(len(vocab)//2 + 5)], vocab[-5:])
```

```
['10' '105' '113' '12' '15'] ['interruption' 'intersected' 'intersecting' 'intersection'
'interstate'] ['year' 'years' 'yellow' 'yield' 'zone']
```

Create the X matrices:

```
X_train_bow = vectorise_dataset(X_train, vect)
X_val_bow = vectorise_dataset(X_val, vect)
X_test_bow = vectorise_dataset(X_test, vect)
```

What is left?

```python
for i in range(8):
    sentence = X_train["SUMMARY_EN"].iloc[i]
    num_words = 0
    for word in sentence.split(" "):
        if word in vocab:
            print(word, end=" ")
            num_words += 1
        if num_words == 10:
            break
    print()
```

```
crash occurred early afternoon weekday middle suburban intersection consisted lanes
crash occurred roadway level consists lanes direction center left turn
crash occurred eastbound direction entrance ramp right curved road uphill
crash occurred straight roadway consists lanes direction center left turn
collision occurred evening hours crash occurred level bituminous roadway residential
vehicle crash occurred daylight location lane undivided left curved downhill
vehicle crash occurred early morning daylight hours roadway traffic roadway
crash occurred northbound lanes northbound southbound slightly street curved posted
```

Note

We hope to see SMS-like language, with limited vocabulary but still able to understand it.

Check the input matrix

```
1 vectorise_dataset(X_train, vect, dataframe=True)
```

	10	105	113	12	15	150	16	17	18	180	...	yield	zone	WEATHER1	W
2532	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
6209	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
2561	1	0	1	0	0	0	0	0	0	0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6882	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
206	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
6356	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0

4169 rows × 1008 columns

Make & inspect the model

```
num_features = X_train_bow.shape[1]
model = build_model(num_features, num_cats)
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 100)	100,900
dense_3 (Dense)	(None, 3)	303

Total params: 101,203 (395.32 KB)
Trainable params: 101,203 (395.32 KB)
Non-trainable params: 0 (0.00 B)

Fit & evaluate the model

```
es = EarlyStopping(patience=1, restore_best_weights=True,
    monitor="val_accuracy", verbose=2)
%time hist = model.fit(X_train_bow, y_train, epochs=10, \
    callbacks=[es], validation_data=(X_val_bow, y_val), verbose=0);
```

```
Epoch 3: early stopping
Restoring model weights from the end of the best epoch: 2.
CPU times: user 1.35 s, sys: 336 ms, total: 1.69 s
Wall time: 1.42 s
```

```
model.evaluate(X_train_bow, y_train, verbose=0)
```

```
[0.08720420300960541, 0.9848884344100952, 0.9997601509094238]
```

```
model.evaluate(X_val_bow, y_val, verbose=0)
```

```
[0.18279702961444855, 0.9460431933403015, 0.9949640035629272]
```

Lecture Outline

- Natural Language Processing
- Car Crash Police Reports
- Text Vectorisation
- Bag Of Words
- Limiting The Vocabulary
- **Intelligently Limit The Vocabulary**

Keep 1,000 most frequent words

```
vect = CountVectorizer(max_features=1_000, stop_words="english")
vect.fit(X_train["SUMMARY_EN"])
vocab = vect.get_feature_names_out()
len(vocab)
```

```
1000
```

```
print(vocab[:5], vocab[len(vocab)//2:(len(vocab)//2 + 5)], vocab[-5:])
```

```
['10' '105' '113' '12' '15'] ['interruption' 'intersected' 'intersecting' 'intersection'
'interstate'] ['year' 'years' 'yellow' 'yield' 'zone']
```

Install spacy

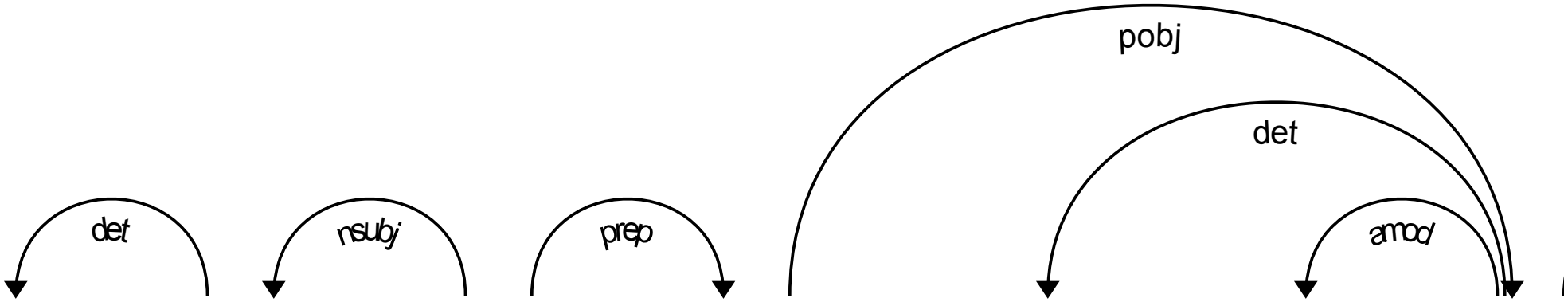
```
!pip install spacy
!python -m spacy download en_core_web_sm
```

```
import spacy

nlp = spacy.load("en_core_web_trf")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
for token in doc:
    print(token.text, token.pos_, token.dep_, token.lemma_)
```

```
Apple PROPN nsubj Apple
is AUX aux be
looking VERB ROOT look
at ADP prep at
buying VERB pcomp buy
U.K. PROPN compound U.K.
startup NOUN dobj startup
for ADP prep for
$ SYM quantmod $
1 NUM compound 1
billion NUM pobj billion
```

Dependency visualiser

```
1 from spacy import displacy
2 doc = nlp(df["SUMMARY_EN"].iloc[1])
3 displacy.render(doc, style="dep")
```



Entity recognition

```
1 doc = nlp(df["SUMMARY_EN"].iloc[1])
2 displacy.render(doc, style="ent")
```

The crash occurred in the eastbound lane of a two CARDINAL -lane, two CARDINAL -way asphalt roadway on level grade. The conditions were daylight and wet with cloudy skies in the early afternoon TIME on a weekday DATE .

V342542243 PRODUCT , a 1995 DATE Chevrolet ORG Lumina PRODUCT was traveling eastbound. V342542269 PRODUCT , a 2004 DATE Chevrolet ORG Trailblazer PRODUCT was also traveling eastbound on the same roadway. V342542269 PRODUCT , was attempting to make a left-hand turn into a private drive on the North side of the roadway. While turning V342542243 PRODUCT attempted to

Stemming

> "Stemming refers to the process of removing suffixes and reducing a word to some base form such that all different variants of that word can be represented by the same form (e.g., "car" and "cars" are both reduced to "car"). This is accomplished by applying a fixed set of rules (e.g., if the word ends in "-es," remove "-es"). More such examples are shown in Figure 2-7. Although such rules may not always end up in a linguistically correct base form, stemming is commonly used in search engines to match user queries to relevant documents and in text classification to reduce the feature space to train machine learning models."

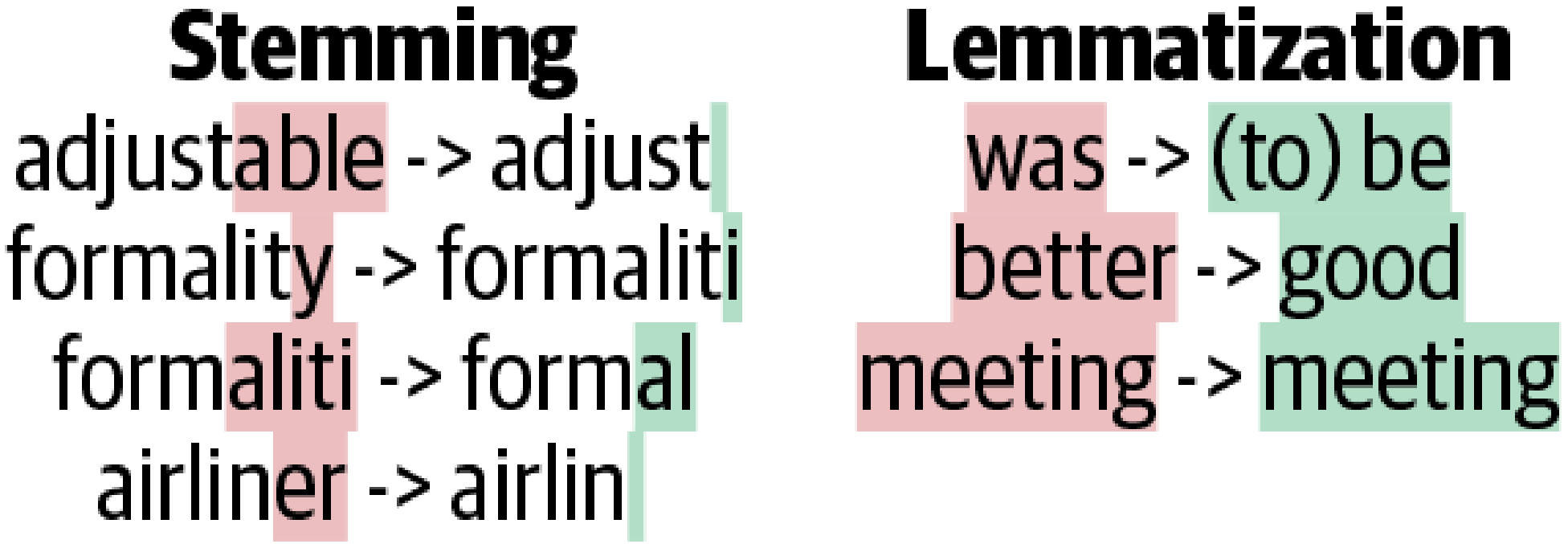
Source: Vajjala et al. (2020), *Practical natural language processing: a comprehensive guide to building real-world NLP systems*, O'Reilly Media.

Lemmatization

> "Lemmatization is the process of mapping all the different forms of a word to its base word, or lemma. While this seems close to the definition of stemming, they are, in fact, different. For example, the adjective "better," when stemmed, remains the same. However, upon lemmatization, this should become "good," as shown in Figure 2-7. Lemmatization requires more linguistic knowledge, and modeling and developing efficient lemmatizers remains an open problem in NLP research even now."

Source: Vajjala et al. (2020), *Practical natural language processing: a comprehensive guide to building real-world NLP systems*, O'Reilly Media.

Stemming and lemmatizing



Examples of stemming and lemmatization

Original: “The striped bats are hanging on their feet for best”

Stemmed: “the stripe bat are hang on their feet for best”

Lemmatized: “the stripe bat be hang on their foot for good”

Source: Kushwah (2019) What is difference between stemming and lemmatization?, Quora.

Examples

Stemmed

organization ↠ organ

civilization ↠ civil

information ↠ inform

consultant ↠ consult

Lemmatized

Here's looking at you, kid. ↠ here be look at you , kid .

Lemmatize the text

```
def lemmatize(txt):
    doc = nlp(txt)
    good_tokens = [token.lemma_.lower() for token in doc \
        if not token.like_num and \
           not token.is_punct and \
           not token.is_space and \
           not token.is_currency and \
           not token.is_stop]
    return " ".join(good_tokens)
```

```
test_str = "Incident at 100kph and '10 incidents -13.3%' are incidental?\t $5"
lemmatize(test_str)
```

```
'incident 100kph incident incidental'
```

```
test_str = "I interviewed 5-years ago, 150 interviews every year at 10:30 are.."
lemmatize(test_str)
```

```
'interview year ago interview year 10:30'
```

Apply to the whole dataset

```
df["SUMMARY_EN_LEMMA"] = df["SUMMARY_EN"].map(lemmatize)
```

```
weather_cols = [f"WEATHER{i}" for i in range(1, 9)]
features = df[["SUMMARY_EN_LEMMA"] + weather_cols]

X_main, X_test, y_main, y_test = \
    train_test_split(features, target, test_size=0.2, random_state=1)

# As 0.25 x 0.8 = 0.2
X_train, X_val, y_train, y_val = \
    train_test_split(X_main, y_main, test_size=0.25, random_state=1)

X_train.shape, X_val.shape, X_test.shape
```

```
((4169, 9), (1390, 9), (1390, 9))
```

What is left?

```
1 print("Original:", df["SUMMARY_EN"].iloc[0][:250])
```

```
Original: V6357885318682, a 2000 Pontiac Montana minivan, made a left turn from a private driveway onto a northbound 5-lane two-way, dry asphalt roadway on a downhill grade.  The posted speed limit on this roadway was 80 kmph (50 MPH). V6357885318682 entered t
```

```
1 print("Lemmatized:", df["SUMMARY_EN_LEMMA"].iloc[0][:250])
```

```
Lemmatized: v6357885318682 pontiac montana minivan left turn private driveway northbound lane way dry asphalt roadway downhill grade post speed limit roadway kmph mph v6357885318682 enter roadway cross southbound lane enter northbound lane left turn lane way int
```

```
1 print("Original:", df["SUMMARY_EN"].iloc[1][:250])
```

```
Original: The crash occurred in the eastbound lane of a two-lane, two-way asphalt roadway on level grade.  The conditions were daylight and wet with cloudy skies in the early afternoon on a weekday.

 V342542243, a 1995 Chevrolet Lumina was traveling eastbou
```

```
1 print("Lemmatized:", df["SUMMARY_EN_LEMMA"].iloc[1][:250])
```

```
Lemmatized: crash occur eastbound lane lane way asphalt roadway level grade condition daylight wet cloudy sky early afternoon weekday v342542243 chevrolet lumina travel eastbound v342542269 chevrolet trailblazer travel eastbound roadway v342542269 attempt left h
```

Keep 1,000 most frequent lemmas

```
vect = CountVectorizer(max_features=1_000, stop_words="english")
vect.fit(X_train["SUMMARY_EN_LEMMA"])
vocab = vect.get_feature_names_out()
len(vocab)
```

```
1000
```

```
print(vocab[:5], vocab[len(vocab)//2:(len(vocab)//2 + 5)], vocab[-5:])
```

```
['10' '150' '48kmph' '4×4' '56kmph'] ['let' 'level' 'lexus' 'license' 'light'] ['yaw' 'year'
'yellow' 'yield' 'zone']
```

Create the X matrices:

```
X_train_bow = vectorise_dataset(X_train, vect, "SUMMARY_EN_LEMMA")
X_val_bow = vectorise_dataset(X_val, vect, "SUMMARY_EN_LEMMA")
X_test_bow = vectorise_dataset(X_test, vect, "SUMMARY_EN_LEMMA")
```

Check the input matrix

```
1 vectorise_dataset(X_train, vect, "SUMMARY_EN_LEMMA", dataframe=True)
```

	10	150	48kmph	4x4	56kmph	64kmph	72kmph	ability	able	acceler
2532	0	0	0	0	0	0	0	0	0	0
6209	0	0	0	0	0	0	0	0	0	0
2561	0	0	0	0	1	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
6882	0	0	0	0	0	0	0	0	0	0
206	0	0	0	0	0	0	0	0	0	0
6356	0	0	0	0	0	0	0	0	0	0

4169 rows × 1008 columns

Make & inspect the model

```
num_features = X_train_bow.shape[1]
model = build_model(num_features, num_cats)
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 100)	100,900
dense_5 (Dense)	(None, 3)	303

Total params: 101,203 (395.32 KB)
Trainable params: 101,203 (395.32 KB)
Non-trainable params: 0 (0.00 B)

Fit & evaluate the model

```
es = EarlyStopping(patience=1, restore_best_weights=True,
    monitor="val_accuracy", verbose=2)
%time hist = model.fit(X_train_bow, y_train, epochs=10, \
    callbacks=[es], validation_data=(X_val_bow, y_val), verbose=0);
```

```
Epoch 4: early stopping
Restoring model weights from the end of the best epoch: 3.
CPU times: user 2 s, sys: 455 ms, total: 2.45 s
Wall time: 2.08 s
```

```
model.evaluate(X_train_bow, y_train, verbose=0)
```

```
[0.05321091413497925, 0.9923242926597595, 1.0]
```

```
model.evaluate(X_val_bow, y_val, verbose=0)
```

```
[0.16859304904937744, 0.9467625617980957, 0.9971222877502441]
```

Package Versions

```
1 from watermark import watermark
2 print(watermark(python=True, packages="keras,matplotlib,numpy,pandas,seaborn,scipy,torch"
```

```
Python implementation: CPython
Python version       : 3.14.3
IPython version      : 9.13.0

keras     : 3.14.1
matplotlib: 3.10.9
numpy     : 2.4.4
pandas    : 3.0.2
seaborn   : 0.13.2
scipy     : 1.17.1
torch     : 2.11.0
```

Glossary

- bag of words
- lemmatization
- n-grams
- one-hot embedding
- TF-IDF
- vocabulary
- word embedding
- word2vec