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
- Interrogate The Model
- Word Embeddings
- Word Embeddings II
- Car Crash NLP Part II

Source: Krohn (2019), *Deep Learning Illustrated*, Chapter 2.

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

UNSW SYDNEY

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:

```
1 print([ord(x) for x in "patrick laub"])
2 print([ord(x) for x in "PATRICK LAUB"])
3 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?

UNSW SYDNEY

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'
```

UNSW SYDNEY

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!'"
```

UNSW SYDNEY

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)
```

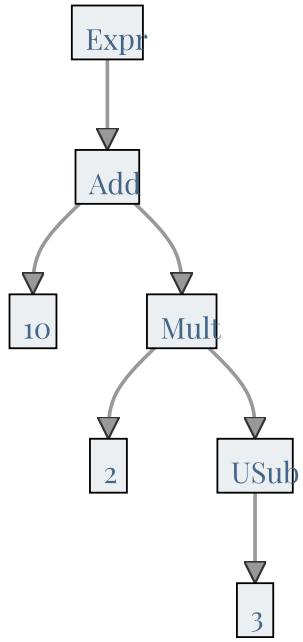
```
TokenInfo(type=63 (ENCODING), string='utf-8', start=(0, 0), end=(0, 0), line='')
TokenInfo(type=2 (NUMBER), string='10', start=(1, 0), end=(1, 2), line='10 + 2 * -3')
TokenInfo(type=54 (OP), string='+', start=(1, 3), end=(1, 4), line='10 + 2 * -3')
TokenInfo(type=2 (NUMBER), string='2', start=(1, 5), end=(1, 6), line='10 + 2 * -3')
TokenInfo(type=54 (OP), string='*', start=(1, 7), end=(1, 8), line='10 + 2 * -3')
TokenInfo(type=54 (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='')
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))))],
  type_ignores=[])
```



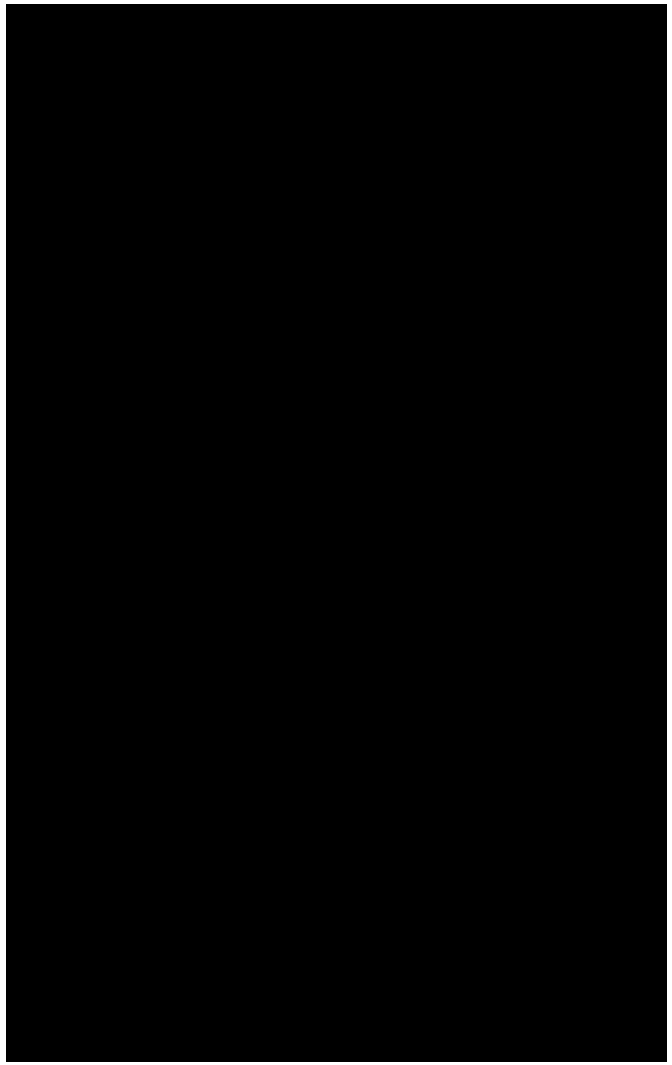
UNSW SYDNEY

Non-natural language processing IV

The abstract syntax tree is then compiled into bytecode.

```
1 import dis
2
3 def expression(a, b, c):
4     return a + b * -c
5
6 dis.dis(expression)
```

```
3           0 RESUME                   0

4           2 LOAD_FAST                0 (a)
            4 LOAD_FAST                1 (b)
            6 LOAD_FAST                2 (c)
            8 UNARY_NEGATIVE
           10 BINARY_OP                5 (*)
           14 BINARY_OP                0 (+)
           18 RETURN_VALUE
```



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

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 #1

> 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.

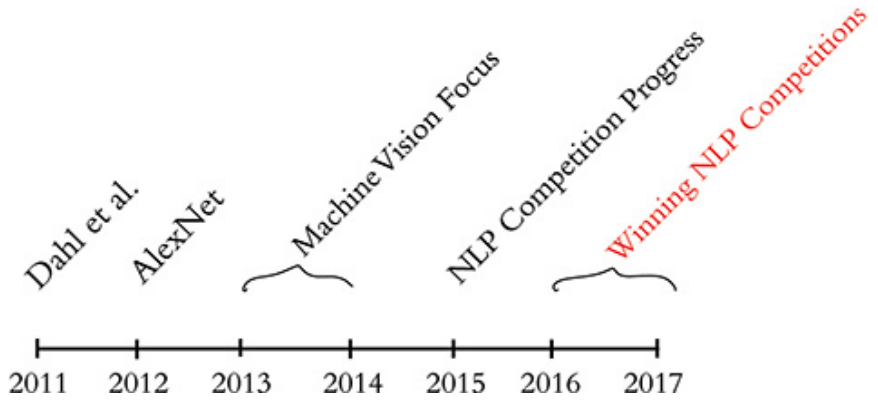
Source: Russell and Norvig (2016), *Artificial Intelligence: A Modern Approach*, Third Edition, p. 21.

NLP in 1966–1973 #2

> 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: Krohn (2019), *Deep Learning Illustrated*, Figure 2-3.

Lecture Outline

- Natural Language Processing
- **Car Crash Police Reports**
- Text Vectorisation
- Bag Of Words
- Limiting The Vocabulary
- Intelligently Limit The Vocabulary
- Interrogate The Model
- Word Embeddings
- Word Embeddings II
- Car Crash NLP Part II

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("NHTSA_NMVCCS_extract.parquet.gzip").exists():
    print("Downloading dataset")
    !wget https://github.com/JSchelldorfer/ActuarialDataScience/raw/master/12%20-%20NLP%

df = pd.read_parquet("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`:
  - `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

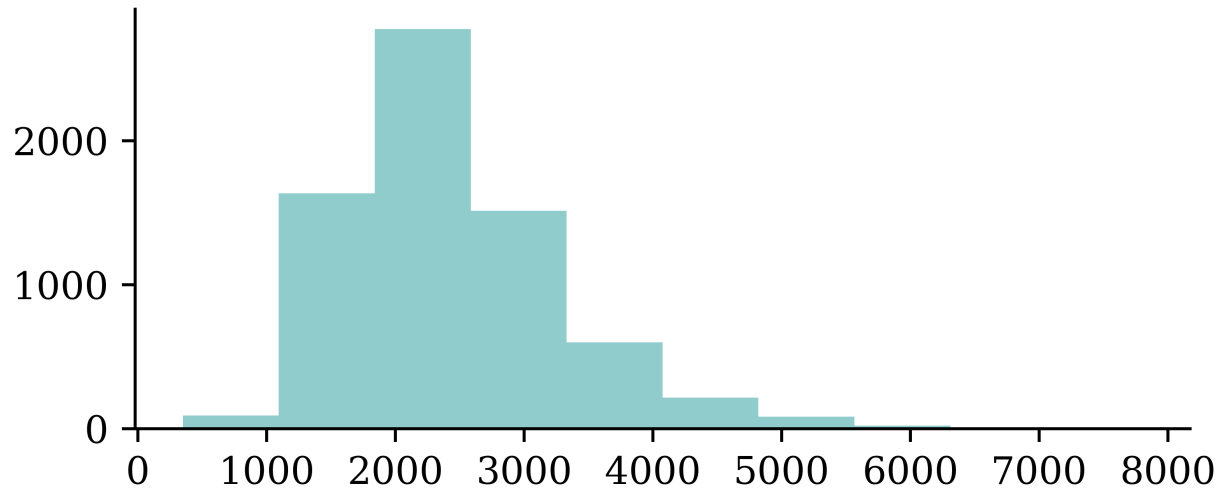
Source: JSchelldorfer's GitHub.

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: object
```

```
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 and felt rested. She was not injured in the crash.\r \r 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

Carriage returns

```
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.

```
# Replace every \r with \n
def replace_carriage_return(summary):
    return summary.replace("\r", "\n")

df["SUMMARY_EN"] = df["SUMMARY_EN"].map(replace_carriage_return)
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)
```

```
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
```

UNSW SYDNEY

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", "nin
    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"v
        summary = summary.replace(f"Vehicle #{word_numbers[car-1]}", new_num).replace(f"
        summary = summary.replace(f"Vehicle {car}", new_num).replace(f"vehicle {car}", n
        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
```

UNSW SYDNEY

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])
```

```
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]])
```

```
[0.25833533221396016, 0.6032621731830176, 0.1384024946030223]
```

UNSW SYDNEY

Lecture Outline

- Natural Language Processing
- Car Crash Police Reports
- **Text Vectorisation**
- Bag Of Words
- Limiting The Vocabulary
- Intelligently Limit The Vocabulary
- Interrogate The Model
- Word Embeddings
- Word Embeddings II
- Car Crash NLP Part II

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: object
```

```
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: object
```

UNSW SYDNEY

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
```

```
<3×13 sparse matrix of type '<class 'numpy.int64'>'
    with 21 stored elements in Compressed Sparse Row format>
```

```
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]])
```

UNSW SYDNEY

Encode new sentences to BoW

```
vect.transform([
    "first car hit second car in a crash",
    "ipad os 16 beta released",
])
```

```
<2×13 sparse matrix of type '<class 'numpy.int64'>'
    with 2 stored elements in Compressed Sparse Row format>
```

```
vect.transform([
    "first car hit second car in a crash",
    "ipad os 16 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']
```

UNSW SYDNEY

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
- Interrogate The Model
- Word Embeddings
- Word Embeddings II
- Car Crash NLP Part II

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_ct = vectorise_dataset(X_train, vect)
X_val_ct = vectorise_dataset(X_val, vect)
X_test_ct = 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	00ty
2532	0	0	0	0	0	0	0	0	0
6209	0	0	0	0	0	0	0	0	0
2561	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...
6882	0	0	0	0	0	0	0	0	0
206	0	0	0	0	0	0	0	0	0
6356	0	0	0	0	0	0	0	0	0

4169 rows × 18874 columns

Make a simple dense model

```
num_features = X_train_ct.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
```

```
model = build_model(num_features, num_cats)
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)

UNSW SYDNEY

Fit & evaluate the model

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

```
Epoch 5: early stopping
Restoring model weights from the end of the best epoch: 4.
CPU times: user 32.6 s, sys: 1.47 s, total: 34.1 s
Wall time: 4.07 s
```

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

```
[0.002541527384892106, 1.0, 1.0]
```

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

```
[2.776606559753418, 0.9453237652778625, 0.9949640035629272]
```

```
model.evaluate(X_test_ct, y_test, verbose=0)
```

```
[0.1902949959039688, 0.9374100565910339, 0.9971222877502441]
```

UNSW SYDNEY

Lecture Outline

- Natural Language Processing
- Car Crash Police Reports
- Text Vectorisation
- Bag Of Words
- **Limiting The Vocabulary**
- Intelligently Limit The Vocabulary
- Interrogate The Model
- Word Embeddings
- Word Embeddings II
- Car Crash NLP Part II

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']
```

Remove stop words

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

```
10
```

```
1 print(vocab)
```

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

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_ct = vectorise_dataset(X_train, vect)
X_val_ct = vectorise_dataset(X_val, vect)
X_test_ct = vectorise_dataset(X_test, vect)
```

UNSW SYDNEY

Check the input matrix

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

	10	105	113	12	15	150	16	17	18	180	...	yield	zoi
2532	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
2561	1	0	1	0	0	0	0	0	0	0	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6882	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
6356	0	0	0	0	0	0	0	0	0	0	...	0	0

4169 rows × 1008 columns

Make & inspect the model

```
num_features = X_train_ct.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)

UNSW SYDNEY

Fit & evaluate the model

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

```
Epoch 3: early stopping
Restoring model weights from the end of the best epoch: 2.
CPU times: user 1.31 s, sys: 144 ms, total: 1.45 s
Wall time: 871 ms
```

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

```
[0.1021684780716896, 0.9815303683280945, 0.9990405440330505]
```

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

```
[2.4335882663726807, 0.9381294846534729, 0.9942445755004883]
```

UNSW SYDNEY

Lecture Outline

- Natural Language Processing
- Car Crash Police Reports
- Text Vectorisation
- Bag Of Words
- Limiting The Vocabulary
- **Intelligently Limit The Vocabulary**
- Interrogate The Model
- Word Embeddings
- Word Embeddings II
- Car Crash NLP Part II

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']
```

UNSW SYDNEY

Install spacy

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

```
import spacy

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

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

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))
```

UNSW SYDNEY

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_ct = vectorise_dataset(X_train, vect, "SUMMARY_EN_LEMMA")
X_val_ct = vectorise_dataset(X_val, vect, "SUMMARY_EN_LEMMA")
X_test_ct = 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	abil
2532	0	0	0	0	0	0	0	0
6209	0	0	0	0	0	0	0	0
2561	0	0	0	0	1	1	0	0
...	...	...	...	...	...	...	...	...
6882	0	0	0	0	0	0	0	0
206	0	0	0	0	0	0	0	0
6356	0	0	0	0	0	0	0	0

4169 rows × 1008 columns

Make & inspect the model

```
num_features = X_train_ct.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)

UNSW SYDNEY

Fit & evaluate the model

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

```
Epoch 3: early stopping
Restoring model weights from the end of the best epoch: 2.
CPU times: user 1.45 s, sys: 174 ms, total: 1.62 s
Wall time: 1.02 s
```

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

```
[0.09055039286613464, 0.9851283431053162, 0.9990405440330505]
```

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

```
[3.8409152030944824, 0.9402877688407898, 0.9928057789802551]
```

UNSW SYDNEY

Lecture Outline

- Natural Language Processing
- Car Crash Police Reports
- Text Vectorisation
- Bag Of Words
- Limiting The Vocabulary
- Intelligently Limit The Vocabulary
- **Interrogate The Model**
- Word Embeddings
- Word Embeddings II
- Car Crash NLP Part II

Permutation importance algorithm

Taken directly from scikit-learn documentation:

- Inputs: fitted predictive model m, tabular dataset (training or validation) D.
- Compute the reference score s of the model m on data D (for instance the accuracy for a classifier or the R^2 for a regressor).
- For each feature j (column of D):
  - For each repetition k in $1, \ldots, K$:
    - Randomly shuffle column j of dataset D to generate a corrupted version of the data named $\tilde{D}_{k,j}$.
    - Compute the score $s_{k,j}$ of model m on corrupted data $\tilde{D}_{k,j}$.
  - Compute importance i_j for feature f_j defined as:

$$i_j = s - \frac{1}{K} \sum_{k=1}^{K} s_{k,j}$$

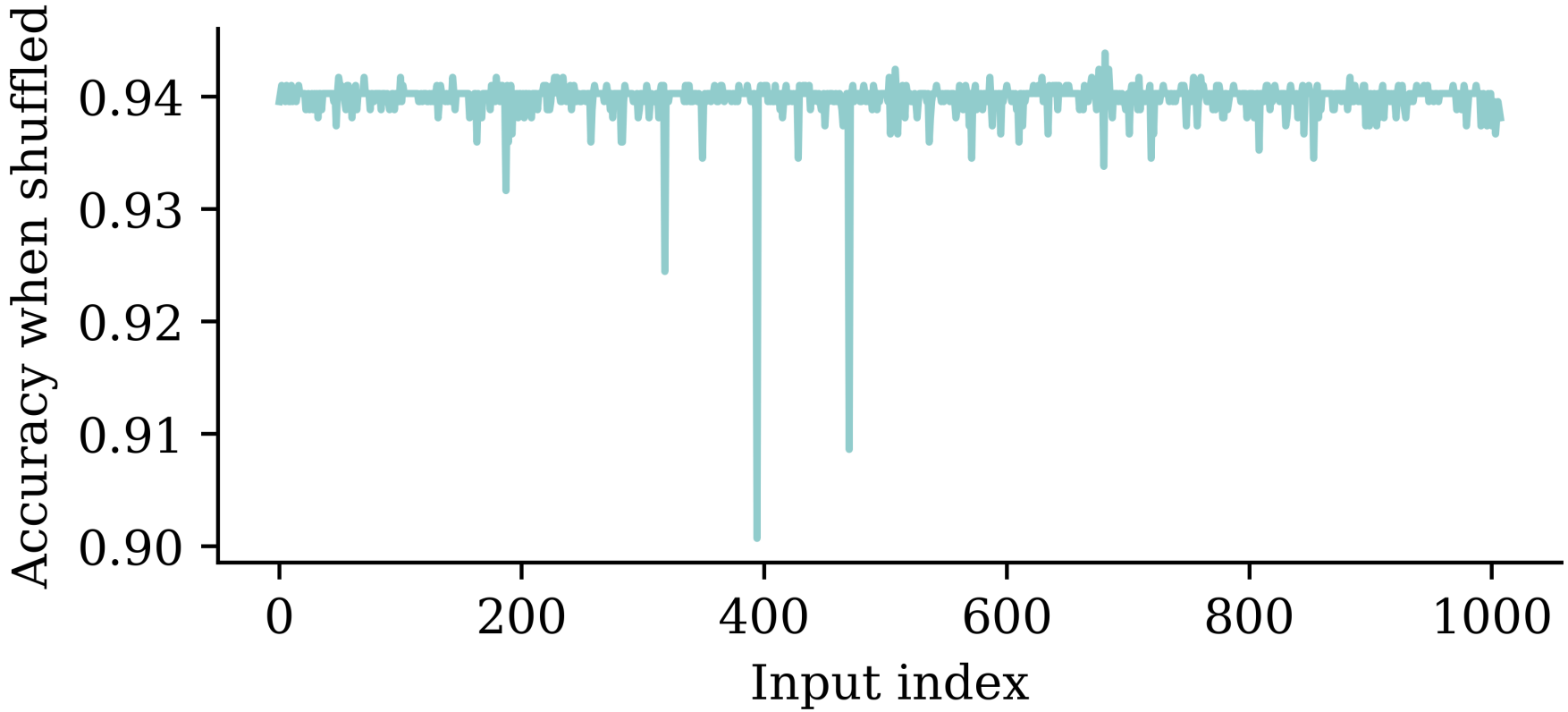
Source: scikit-learn documentation, permutation_importance function.

Find important inputs

```python
def permutation_test(model, X, y, num_reps=1, seed=42):
    """
    Run the permutation test for variable importance.
    Returns matrix of shape (X.shape[1], len(model.evaluate(X, y))).
    """
    rnd.seed(seed)
    scores = []

    for j in range(X.shape[1]):
        original_column = np.copy(X[:, j])
        col_scores = []

        for r in range(num_reps):
            rnd.shuffle(X[:,j])
            col_scores.append(model.evaluate(X, y, verbose=0))

        scores.append(np.mean(col_scores, axis=0))
        X[:,j] = original_column

    return np.array(scores)
```

Run the permutation test

```
1 perm_scores = permutation_test(model, X_val_ct, y_val)[:,1]
2 plt.plot(perm_scores);
3 plt.xlabel("Input index"); plt.ylabel("Accuracy when shuffled");
```



Find the most significant inputs

```
1 vocab = vect.get_feature_names_out()
2 input_cols = list(vocab) + weather_cols
3
4 best_input_inds = np.argsort(perm_scores)[:100]
5 best_inputs = [input_cols[idx] for idx in best_input_inds]
6
7 print(best_inputs)
```

```
['harmful', 'involve', 'event', 'contact', 'pre', 'stop', 'rear', 'motor', 'female',
'impact', 'single', 'occur', 'drive', 'code', 'male', 'continue', 'driver', 'direction',
'push', 'northbound', 'state', 'park', 'light', 'line', 'control', 'WEATHER4', 'reason',
'traffic', 'night', 'old', 'injure', 'right', 'apply', 'tow', 'rest', 'WEATHER1', 'year',
'morning', 'tire', 'work', 'south', 'westbound', 'intersection', 'second', 'shoulder',
'sign', 'edge', 'southbound', 'spin', 'seat', 'cross', 'encroachment', 'pressure', 'cause',
'error', 'clear', 'come', 'malibu', 'loss', 'local', 'hospital', 'correct', 'coupe', 'door',
'minivan', 'WEATHER8', 'turn', 'street', 'WEATHER5', 'undivided', 'trailer', 'travel',
'asphalt', 'ahead', 'afternoon', 'aggressive', 'undivide', 'color', 'collision', 'minor',
'miss', 'associative', 'alcohol', 'adverse', 'mph', 'near', 'chevrolet', 'past', 'physical',
'pickup', 'clockwise', 'WEATHER7', 'adjacent', 'position', 'edr', 'dodge', 'yield',
'directional', 'hard', 'departure']
```

UNSW SYDNEY

How about a simple decision tree?

```
1 from sklearn import tree
2 clf = tree.DecisionTreeClassifier(random_state=0, max_leaf_nodes=3)
3 clf.fit(X_train_ct[:, best_input_inds], y_train);
```

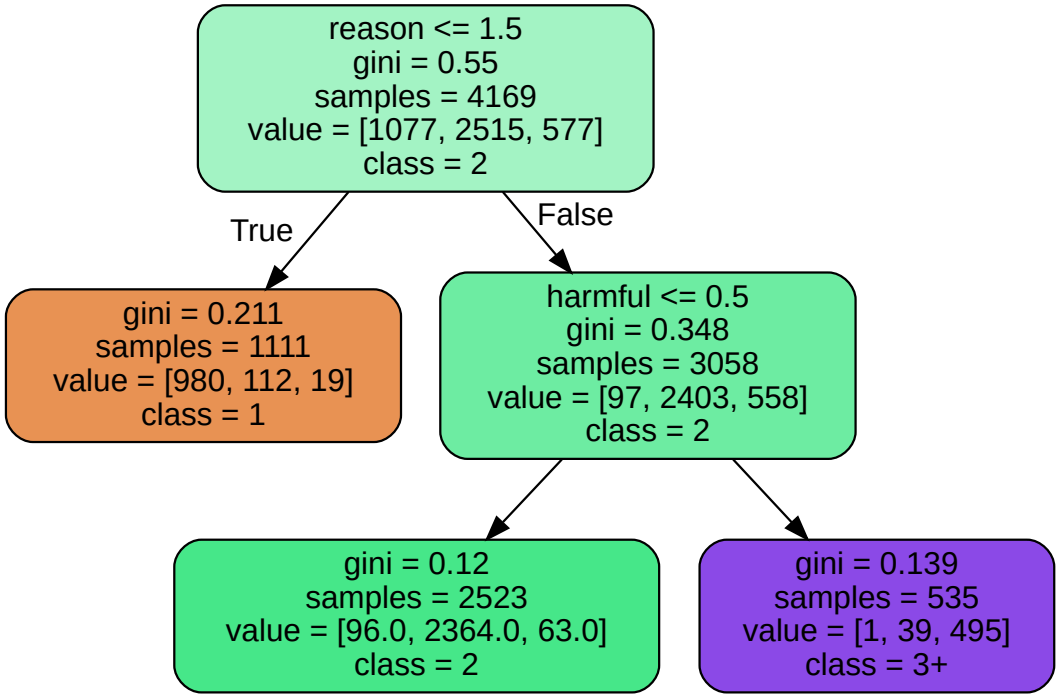
```
1 print(clf.score(X_train_ct[:, best_input_inds], y_train))
2 print(clf.score(X_val_ct[:, best_input_inds], y_val))
```

```
0.920844327176781
0.9330935251798561
```

UNSW SYDNEY

Decision tree

```
1 tree.plot_tree(clf, feature_names=best_inputs, filled=True);
```



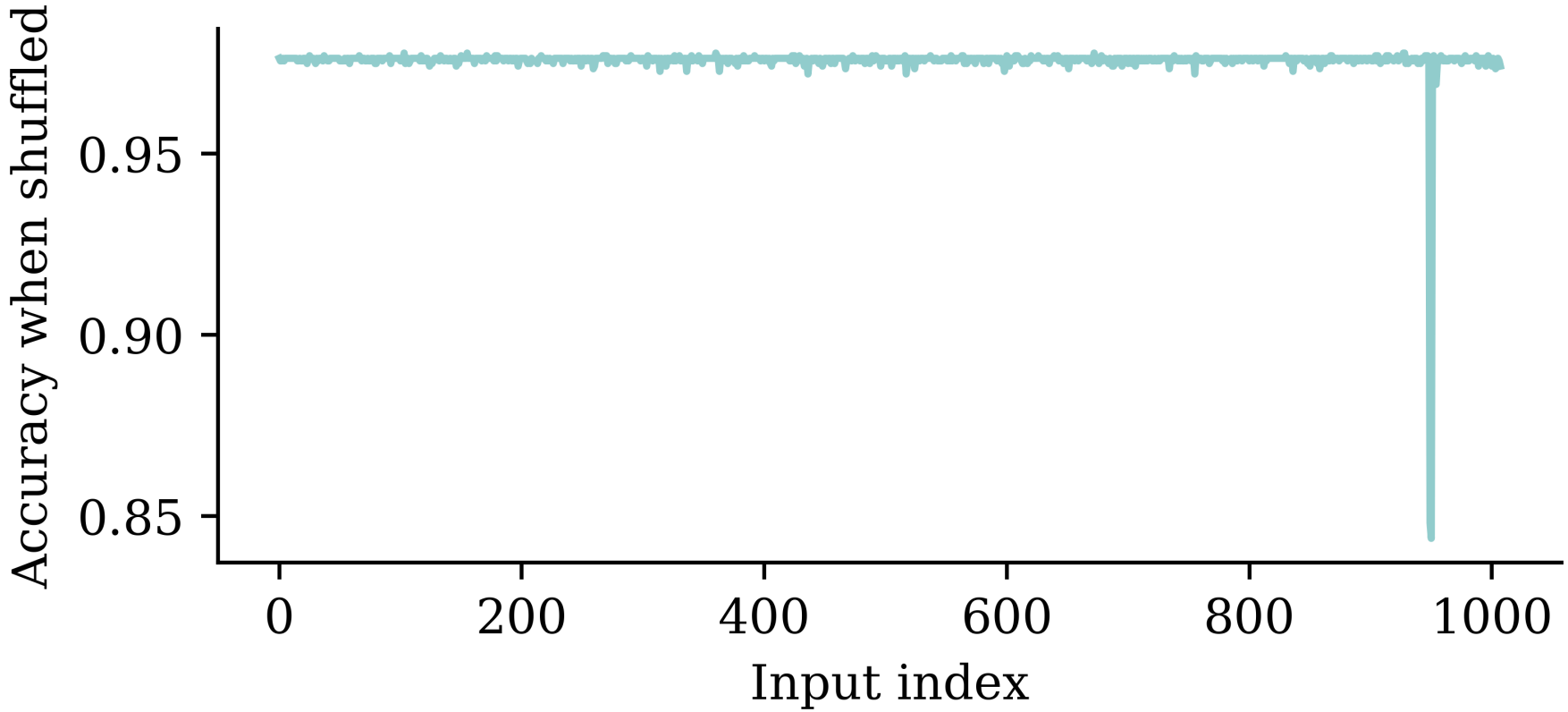

```
1 print(np.where(clf.feature_importances_ > 0)[0])
2 [best_inputs[ind] for ind in np.where(clf.feature_importances_ > 0)[0]]
```

```
[ 0 26]
['harmful', 'reason']
```

UNSW SYDNEY

Using the original dataset

```
1 perm_scores = permutation_test(model, X_val_ct, y_val)[:,1]
2 plt.plot(perm_scores);
3 plt.xlabel("Input index"); plt.ylabel("Accuracy when shuffled");
```



Find the most significant inputs

```
1 vocab = vect.get_feature_names_out()
2 input_cols = list(vocab) + weather_cols
3
4 best_input_inds = np.argsort(perm_scores)[:100]
5 best_inputs = [input_cols[idx] for idx in best_input_inds]
6
7 print(best_inputs)
```

```
['v3', 'v2', 'vehicle', 'lane', 'harmful', 'right', 'divided', 'south', 'motor', 'dry',
'event', 'left', 'parked', 'WEATHER4', 'related', 'stop', 'impact', 'v4', 'crash',
'direction', 'involved', 'internal', 'stated', 'barrier', 'dodge', 'asphalt', 'chevrolet',
'higher', 'forward', 'pre', 'precrash', 'pushed', 'corner', 'hand', 'prior', 'door',
'WEATHER8', 'factor', 'work', 'mph', 'year', 'critical', 'WEATHER1', 'WEATHER3', 'single',
'WEATHER5', 'straight', 'ahead', 'turning', 'honda', 'hours', 'type', 'daylight', 'possible',
'ford', 'male', 'facing', 'actions', 'consists', 'unknown', 'uphill', 'pick', 'stopped',
'point', 'alcohol', 'high', 'pull', 'proceeded', 'encroaching', 'morning', 'trailer',
'grand', 'associated', 'blood', 'meters', 'basis', 'experience', 'prescription', 'moved',
'small', 'steered', 'maneuver', 'medication', 'heart', 'rotate', 'old', 'pain', 'weekday',
'clear', 'seconds', 'civic', 'started', 'northbound', '2006', 'noon', 'miles', '44',
'injuries', 'vehicles', 'saw']
```

UNSW SYDNEY

How about a simple decision tree?

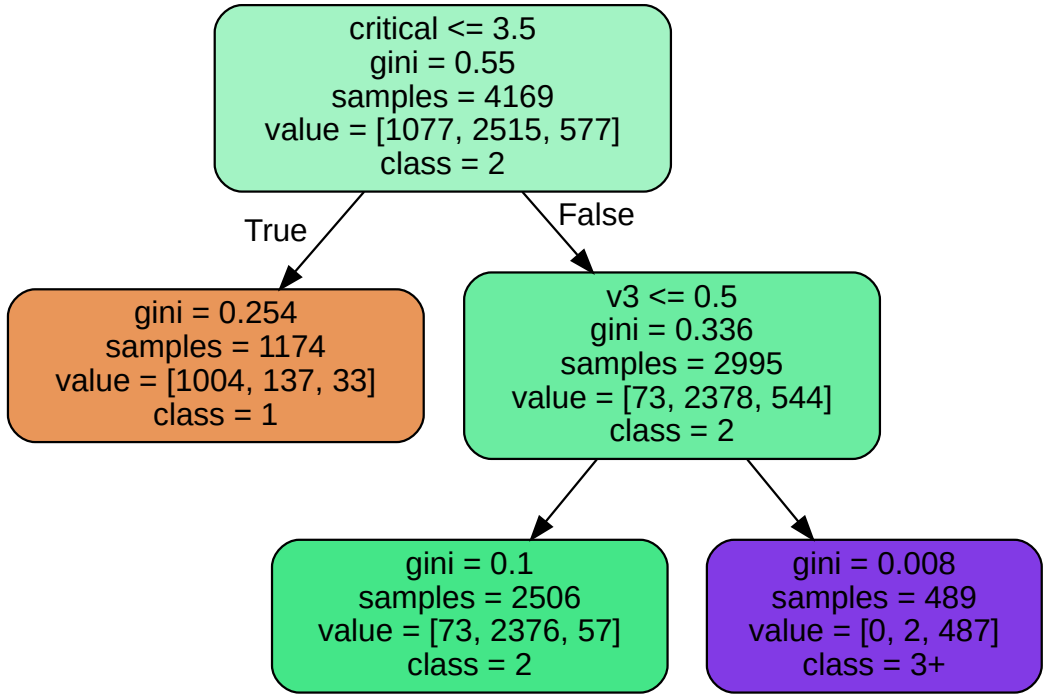
```
clf = tree.DecisionTreeClassifier(random_state=0, max_leaf_nodes=3)
clf.fit(X_train_ct[:, best_input_inds], y_train);
```

```
print(clf.score(X_train_ct[:, best_input_inds], y_train))
print(clf.score(X_val_ct[:, best_input_inds], y_val))
```

```
0.9275605660829935
0.939568345323741
```

Decision tree

```
1 tree.plot_tree(clf, feature_names=best_inputs, filled=True);
```




```
1 print(np.where(clf.feature_importances_ > 0)[0])
2 [best_inputs[ind] for ind in np.where(clf.feature_importances_ > 0)[0]]
```

```
[ 0 41]
['v3', 'critical']
```

UNSW SYDNEY

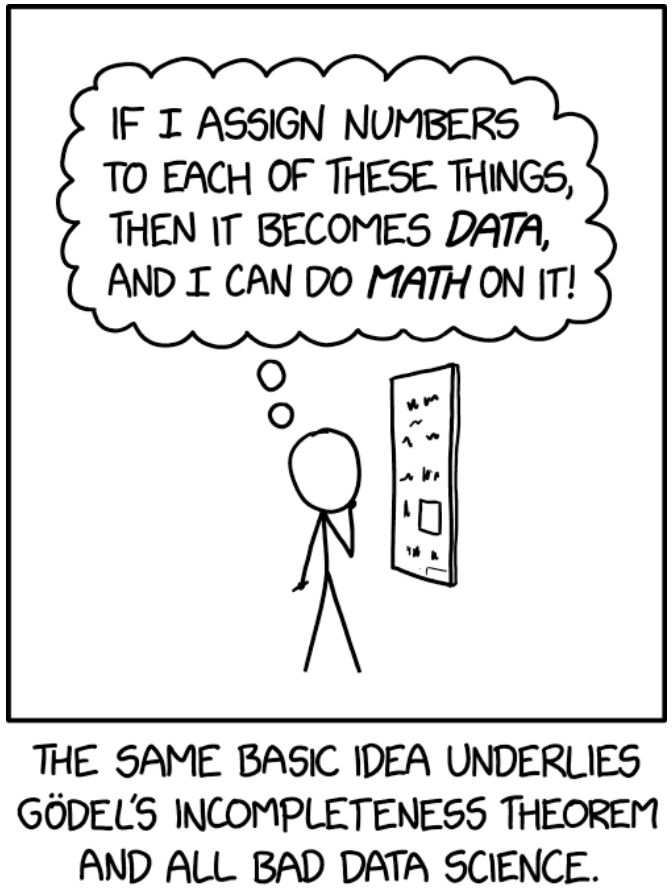
Lecture Outline

- Natural Language Processing
- Car Crash Police Reports
- Text Vectorisation
- Bag Of Words
- Limiting The Vocabulary
- Intelligently Limit The Vocabulary
- Interrogate The Model
- **Word Embeddings**
- Word Embeddings II
- Car Crash NLP Part II

Overview

Popular methods for converting text into numbers include:

- One-hot encoding
- Bag of words
- TF-IDF
- Word vectors (*transfer learning*)



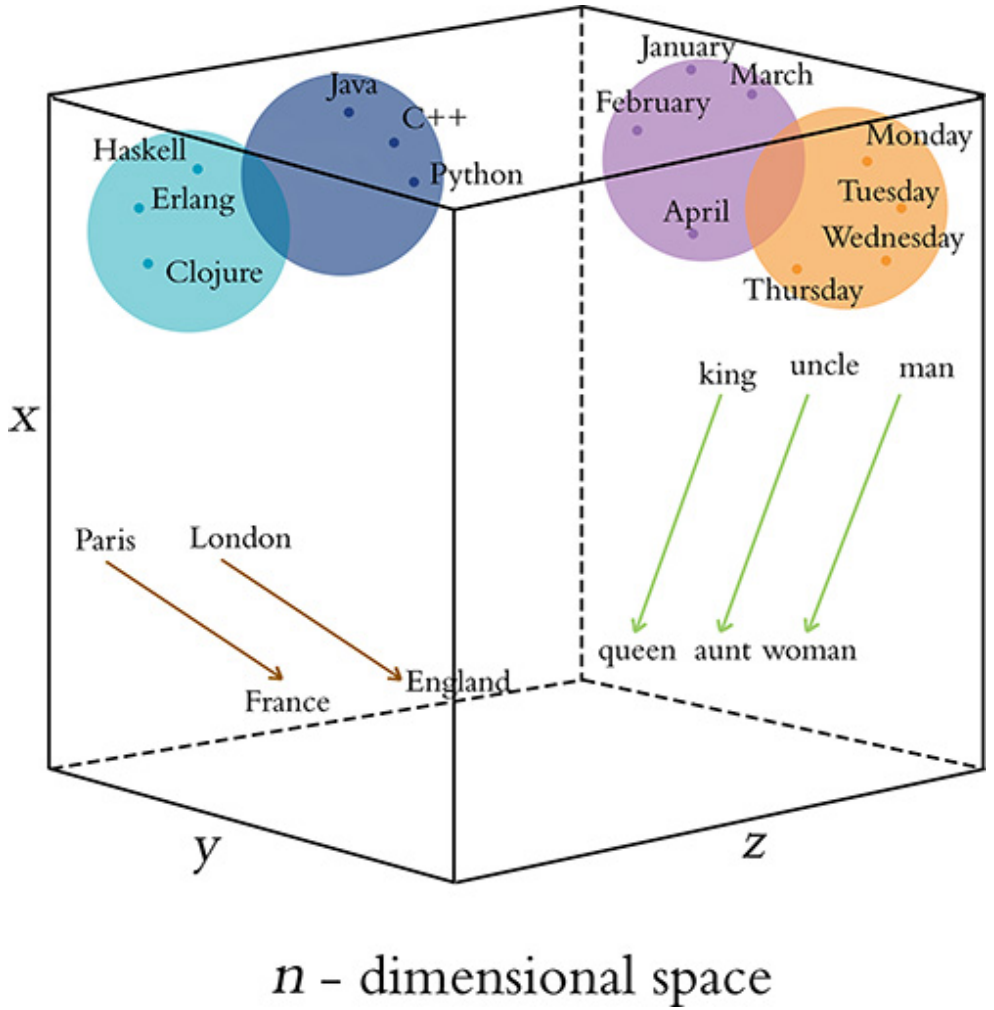
Assigning Numbers

Source: Randall Munroe (2022), xkcd #2610: Assigning Numbers.

Word Vectors

- One-hot representations capture word 'existence' only, whereas word vectors capture information about word meaning as well as location.
- This enables deep learning NLP models to automatically learn linguistic features.
- **Word2Vec** & **GloVe** are popular algorithms for generating word embeddings (i.e. word vectors).

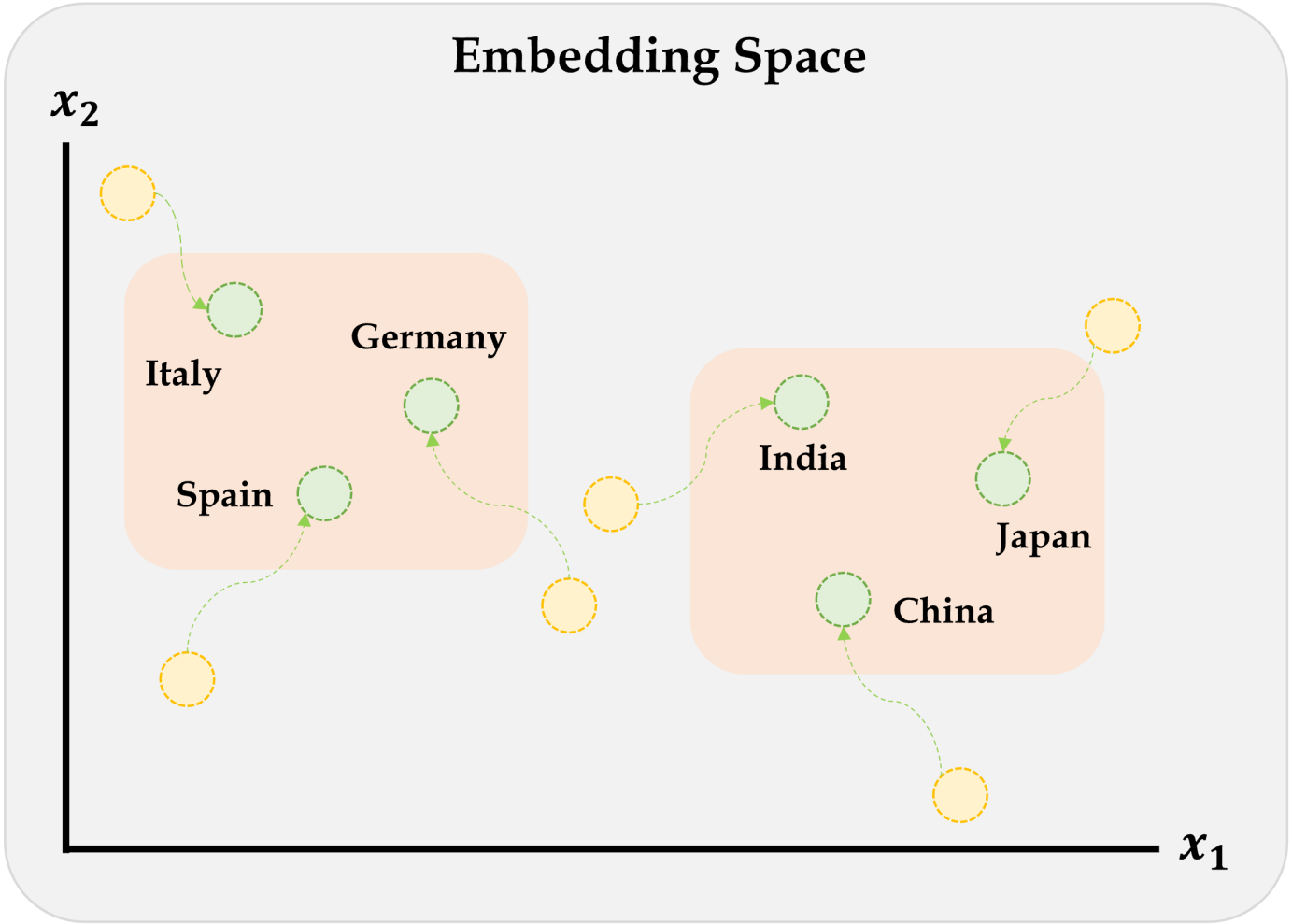
Word Vectors



Illustrative word vectors.

Source: Krohn (2019), *Deep Learning Illustrated*, Figure 2-6.

Remember this diagram?



Embeddings will gradually improve during training.

Source: Aurélien Géron (2019), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, Figure 13-4.

Word2Vec

Key idea: You're known by the company you keep.

Two algorithms are used to calculate embeddings:

- *Continuous bag of words*: uses the context words to predict the target word
- *Skip-gram*: uses the target word to predict the context words

Predictions are made using a neural network with one hidden layer. Through backpropagation, we update a set of "weights" which become the word vectors.

Paper: Mikolov et al. (2013), *Efficient estimation of word representations in vector space*, arXiv:1301.3781.

Word2Vec training methods

Continuous bag of words is a *center word prediction* task

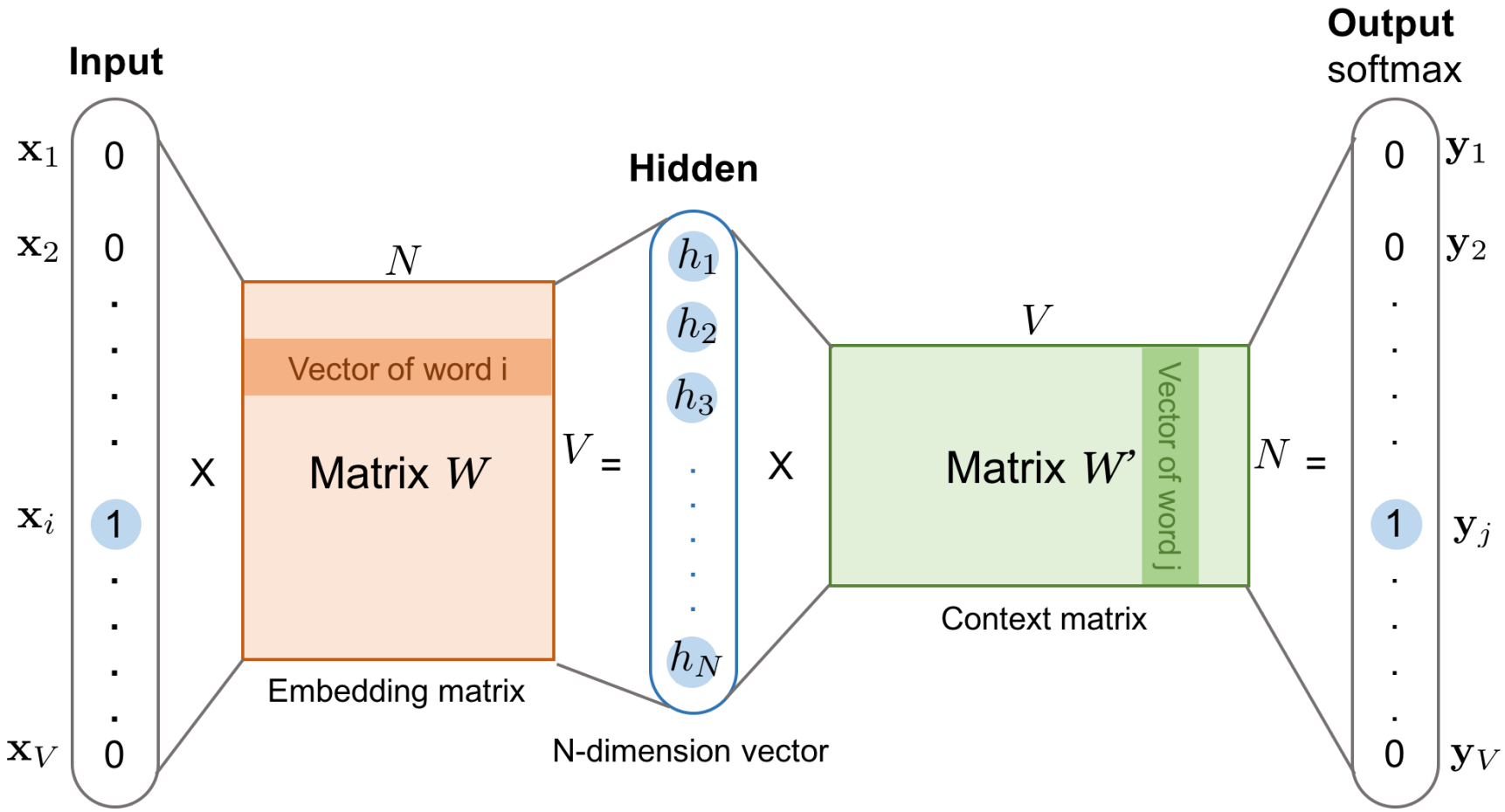
Skip-gram is a *neighbour word prediction* task

Suggested viewing

Computerphile (2019), Vectoring Words (Word Embeddings), YouTube (16 mins).

Source: Amit Chaudhary (2020), Self Supervised Representation Learning in NLP.

The skip-gram network

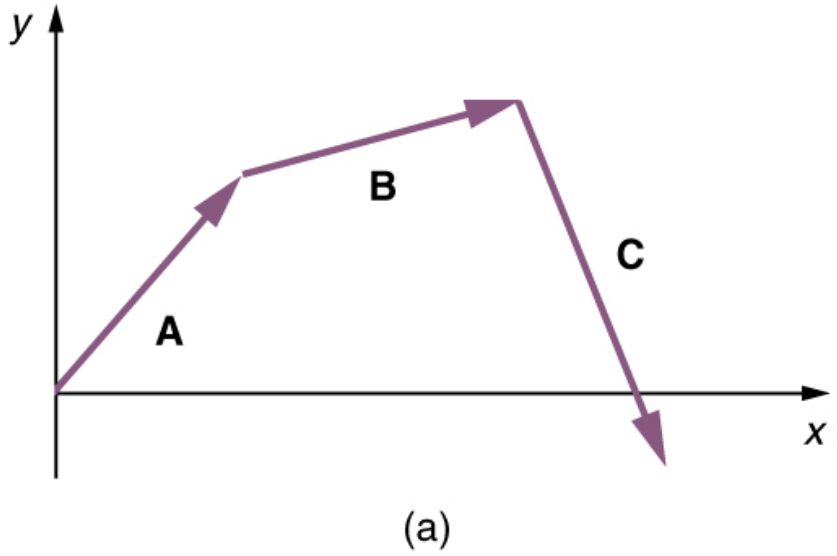


The skip-gram model. Both the input vector $\boldsymbol{x}$ and the output $\boldsymbol{y}$ are one-hot encoded word representations. The hidden layer is the word embedding of size N.

Source: Lilian Weng (2017), Learning Word Embedding, Blog post, Figure 1.

Word Vector Arithmetic

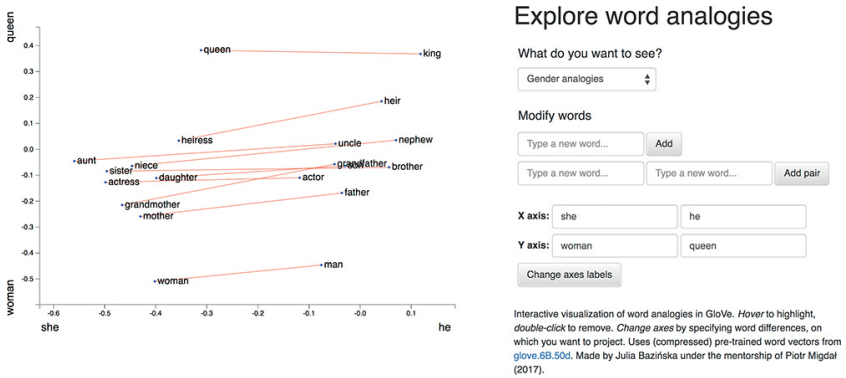
Relationships between words becomes vector math.



You remember vectors, right?

$$v_{king} - v_{man} + v_{woman} = v_{queen}$$
$$v_{bezos} - v_{amazon} + v_{tesla} = v_{musk}$$
$$v_{windows} - v_{microsoft} + v_{google} = v_{android}$$

Illustrative word vector arithmetic



Screenshot from Word2viz

Sources: PressBooks, College Physics: OpenStax, Chapter 17 Figure 9, and Krohn (2019), *Deep Learning Illustrated*, Figures 2-7 & 2-8.

Lecture Outline

- Natural Language Processing
- Car Crash Police Reports
- Text Vectorisation
- Bag Of Words
- Limiting The Vocabulary
- Intelligently Limit The Vocabulary
- Interrogate The Model
- Word Embeddings
- **Word Embeddings II**
- Car Crash NLP Part II

Pretrained word embeddings

```
!pip install gensim
```

Load word2vec embeddings trained on Google News:

```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
```

When run for the first time, that downloads a huge file:

```
gensim_dir = Path("~/gensim-data/").expanduser()
[str(p) for p in gensim_dir.iterdir()]
```

```
['/home/plaub/gensim-data/information.json',
 '/home/plaub/gensim-data/word2vec-google-news-300']
```

```
next(gensim_dir.glob("*/*.gz")).stat().st_size / 1024**3
```

```
1.6238203644752502
```

```
f"The size of the vocabulary is {len(wv)}"
```

```
'The size of the vocabulary is 3000000'
```

Treat `wv` like a dictionary

```
1 wv["pizza"]
```

```
array([-1.26e-01,  2.54e-02,  1.67e-01,  5.51e-01, -7.67e-02,  1.29e-01,
        1.03e-01, -3.95e-04,  1.22e-01,  4.32e-02,  1.73e-01, -6.84e-02,
        3.42e-01,  8.40e-02,  6.69e-02,  2.68e-01, -3.71e-02, -5.57e-02,
        1.81e-01,  1.90e-02, -5.08e-02,  9.03e-03,  1.77e-01,  6.49e-02,
       -6.25e-02, -9.42e-02, -9.72e-02,  4.00e-01,  1.15e-01,  1.03e-01,
       -1.87e-02, -2.70e-01,  1.81e-01,  1.25e-01, -3.17e-02, -5.49e-02,
        3.46e-01, -1.57e-02,  1.82e-05,  2.07e-01, -1.26e-01, -2.83e-01,
        2.00e-01,  8.35e-02, -4.74e-02, -3.11e-02, -2.62e-01,  1.70e-01,
       -2.03e-02,  1.53e-01, -1.21e-01,  3.75e-01, -5.69e-02, -4.76e-03,
       -1.95e-01, -2.03e-01,  3.01e-01, -1.01e-01, -3.18e-01, -9.03e-02,
       -1.19e-01,  1.95e-01, -8.79e-02,  1.58e-01,  1.52e-02, -1.60e-01,
       -3.30e-01, -4.67e-01,  1.69e-01,  2.23e-02,  1.55e-01,  1.08e-01,
       -3.56e-02,  9.13e-02, -8.69e-02, -1.20e-01, -3.09e-01, -2.61e-02,
       -7.23e-02, -4.80e-01,  3.78e-02, -1.36e-01, -1.03e-01, -2.91e-01,
       -1.93e-01, -4.22e-01, -1.06e-01,  3.55e-01,  1.67e-01, -3.63e-03,
       -7.42e-02, -3.22e-01, -7.52e-02, -8.25e-02, -2.91e-01, -1.26e-01,
        1.68e-02,  5.00e-02,  1.28e-01, -7.42e-02, -1.31e-01, -2.46e-01,
        6.49e-02,  1.53e-01,  2.60e-01, -1.05e-01,  3.57e-01, -4.30e-02,
       -1.58e-01,  8.20e-02, -5.98e-02, -2.34e-01, -3.22e-01, -1.26e-01,
```

```
1 len(wv["pizza"])
```

```
300
```

UNSW SYDNEY

Find nearby word vectors

```
1 wv.most_similar("Python")
```

```
[('Jython', 0.6152505874633789),
 ('Perl_Python', 0.5710949897766113),
 ('IronPython', 0.5704679489135742),
 ('scripting_languages', 0.5695090889930725),
 ('PHP_Perl', 0.5687724947929382),
 ('Java_Python', 0.5681070685386658),
 ('PHP', 0.5660915970802307),
 ('Python_Ruby', 0.5632461905479431),
 ('Visual_Basic', 0.5603480339050293),
 ('Perl', 0.5530891418457031)]
```

```
1 wv.similarity("Python", "Java")
```

```
0.46189708
```

```
1 wv.similarity("Python", "sport")
```

```
0.08406468
```

```
1 wv.similarity("Python", "R")
```

```
0.066954285
```

Fun fact: Gensim's `most_similar` uses Spotify's `annoy` library ("Approximate Nearest Neighbors Oh Yeah")

UNSW SYDNEY

What does ‘similarity’ mean?

The ‘similarity’ scores

```
wv.similarity("Sydney", "Melbourne")
```

```
0.8613987
```

are normally based on cosine distance.

```
x = wv["Sydney"]
y = wv["Melbourne"]
x.dot(y) / (np.linalg.norm(x) * np.linalg.norm(y))
```

```
0.86139864
```

```
wv.similarity("Sydney", "Aarhus")
```

```
0.19079602
```

Weng's GoT Word2Vec

In the GoT word embedding space, the top similar words to "king" and "queen" are:

```
1 model.most_similar('king')
```

```
('kings', 0.897245)
('baratheon', 0.809675)
('son', 0.763614)
('robert', 0.708522)
('lords', 0.698684)
('joffrey', 0.696455)
('prince', 0.695699)
('brother', 0.685239)
('aerys', 0.684527)
('stannis', 0.682932)
```

```
1 model.most_similar('queen')
```

```
('cersei', 0.942618)
('joffrey', 0.933756)
('margaery', 0.931099)
('sister', 0.928902)
('prince', 0.927364)
('uncle', 0.922507)
('varys', 0.918421)
('ned', 0.917492)
('melisandre', 0.915403)
('robb', 0.915272)
```

Source: Lilian Weng (2017), Learning Word Embedding, Blog post.

Combining word vectors

You can summarise a sentence by averaging the individual word vectors.

```
1 sv = (wv["Melbourne"] + wv["has"] + wv["better"] + wv["coffee"]) / 4
2 len(sv), sv[:5]
```

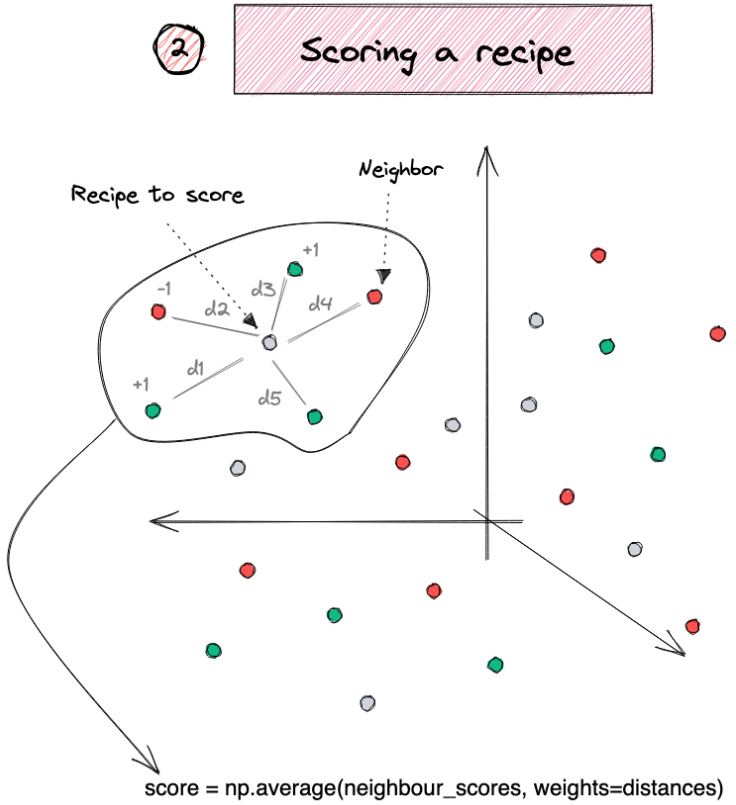
```
(300, array([-0.08, -0.11, -0.16,  0.24,  0.06], dtype=float32))
```

> As it turns out, averaging word embeddings is a surprisingly effective way to create word embeddings. It's not perfect (as you'll see), but it does a strong job of capturing what you might perceive to be complex relationships between words.

Source: Trask (2019), Grokking Deep Learning, Chapter 12.

Recipe recommender

Recipes are the average of the word vectors of the ingredients.



Nearest neighbours used to classify new recipes as potentially delicious.

Source: Duarte O.Carmo (2022), A recipe recommendation system, Blog post.

Analogies with word vectors

Obama is to America as ___ is to Australia.

$$\text{Obama} - \text{America} + \text{Australia} = ?$$

```
1 wv.most_similar(positive=["Obama", "Australia"], negative=["America"])
```

```
[('Mr_Rudd', 0.6151423454284668),
 ('Prime_Minister_Julia_Gillard', 0.6045385003089905),
 ('Prime_Minister_Kevin_Rudd', 0.5982581973075867),
 ('Kevin_Rudd', 0.5627648830413818),
 ('Ms_Gillard', 0.5517690777778625),
 ('Opposition_Leader_Kevin_Rudd', 0.5298037528991699),
 ('Mr_Beazley', 0.5259249210357666),
 ('Gillard', 0.5250653624534607),
 ('NARDA_GILMORE', 0.5203536748886108),
 ('Mr_Downer', 0.5150347948074341)]
```

Testing more associations

```
1 wv.most_similar(positive=["France", "London"], negative=["Paris"])
```

```
[('Britain', 0.7368935346603394),
 ('UK', 0.6637030839920044),
 ('England', 0.6119861602783203),
 ('United_Kingdom', 0.6067784428596497),
 ('Great_Britain', 0.5870823860168457),
 ('Britian', 0.5852951407432556),
 ('Scotland', 0.5410018563270569),
 ('British', 0.5318332314491272),
 ('Europe', 0.5307435989379883),
 ('East_Midlands', 0.5230222344398499)]
```

Quickly get to bad associations

```
1 wv.most_similar(positive=["King", "woman"], negative=["man"])
```

```
[('Queen', 0.5515626668930054),
 ('Oprah_BFF_Gayle', 0.47597548365592957),
 ('Geoffrey_Rush_Exit', 0.46460166573524475),
 ('Princess', 0.4533674716949463),
 ('Yvonne_Stickney', 0.4507041573524475),
 ('L._Bonauto', 0.4422135353088379),
 ('gal_pal_Gayle', 0.4408389925956726),
 ('Alveda_C.', 0.4402790665626526),
 ('Tupou_V.', 0.4373864233493805),
 ('K._Letourneau', 0.4351031482219696)]
```

```
1 wv.most_similar(positive=["computer_programmer", "woman"], negative=["man"])
```

```
[('homemaker', 0.5627118945121765),
 ('housewife', 0.5105047225952148),
 ('graphic_designer', 0.505180299282074),
 ('schoolteacher', 0.497949481010437),
 ('businesswoman', 0.493489146232605),
 ('paralegal', 0.49255111813545227),
 ('registered_nurse', 0.4907974898815155),
 ('saleswoman', 0.4881627559661865),
 ('electrical_engineer', 0.4797725975513458),
 ('mechanical_engineer', 0.4755399227142334)]
```

UNSW SYDNEY

Bias in NLP models

The Verge (2016), Twitter taught Microsoft's AI chatbot to be a racist a****** in less than a day.

... there are serious questions to answer, like how are we going to teach AI using public data without incorporating the worst traits of humanity? If we create bots that mirror their users, do we care if their users are human trash? There are plenty of examples of technology embodying — either accidentally or on purpose — the prejudices of society, and Tay's adventures on Twitter show that even big corporations like Microsoft forget to take any preventative measures against these problems.

The library cheats a little bit

```
wv.similar_by_vector(wv["computer_programmer"]-wv["man"]+wv["woman"])
```

```
[('computer_programmer', 0.910581111907959),
 ('homemaker', 0.5771316289901733),
 ('schoolteacher', 0.5500192046165466),
 ('graphic_designer', 0.5464698672294617),
 ('mechanical_engineer', 0.539836585521698),
 ('electrical_engineer', 0.5337055325508118),
 ('housewife', 0.5274525284767151),
 ('programmer', 0.5096209049224854),
 ('businesswoman', 0.5029540657997131),
 ('keypunch_operator', 0.4974639415740967)]
```

To get the 'nice' analogies, the `.most_similar` ignores the input words as possible answers.

```
# ignore (don't return) keys from the input
result = [
    (self.index_to_key[sim + clip_start], float(dists[sim]))
    for sim in best if (sim + clip_start) not in all_keys
]
```

Source: gensim, gensim/models/keyedvectors.py, lines 853–857.

Lecture Outline

- Natural Language Processing
- Car Crash Police Reports
- Text Vectorisation
- Bag Of Words
- Limiting The Vocabulary
- Intelligently Limit The Vocabulary
- Interrogate The Model
- Word Embeddings
- Word Embeddings II
- **Car Crash NLP Part II**

Dataset source: Dr Jürg Schelldorfer's GitHub.

Predict injury severity

```
features = df["SUMMARY_EN"]
target = LabelEncoder().fit_transform(df["INJSEVB"])

X_main, X_test, y_main, y_test = \
    train_test_split(features, target, test_size=0.2, random_state=1)
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,), (1390,), (1390,))
```

Using Keras `TextVectorization`

```
1  max_tokens = 1_000
2  vect = layers.TextVectorization(
3      max_tokens=max_tokens,
4      output_mode="tf_idf",
5      standardize="lower_and_strip_punctuation",
6  )
7
8  vect.adapt(X_train)
9  vocab = vect.get_vocabulary()
10
11 X_train_txt = vect(X_train)
12 X_val_txt = vect(X_val)
13 X_test_txt = vect(X_test)
14
15 print(vocab[:50])
```

```
['[UNK]', 'the', 'was', 'a', 'to', 'of', 'and', 'in', 'driver', 'for', 'this', 'vehicle',
'critical', 'lane', 'he', 'on', 'with', 'that', 'left', 'roadway', 'coded', 'she', 'event',
'crash', 'not', 'at', 'intersection', 'traveling', 'right', 'precrash', 'as', 'from', 'were',
'by', 'had', 'reason', 'his', 'side', 'is', 'front', 'her', 'traffic', 'an', 'it', 'two',
'speed', 'stated', 'one', 'occurred', 'no']
```

UNSW SYDNEY

The TF-IDF vectors

```
1 pd.DataFrame(X_train_txt, columns=vocab, index=X_train.index)
```

	[UNK]	the	was	a	to	
2532	121.857979	42.274662	10.395409	10.395409	11.785541	8.32
6209	72.596237	17.325682	10.395409	5.544218	4.159603	5.54
2561	124.450699	30.493198	15.246599	11.088436	9.012472	7.62
...	...	...	...	...	...	...
6882	75.188965	20.790817	4.851191	7.623300	9.012472	4.85
206	147.785202	27.028063	13.167518	6.237246	8.319205	4.85
6356	75.188965	15.246599	9.702381	8.316327	7.625938	5.54

4169 rows × 1000 columns

UNSW SYDNEY

Feed TF-IDF into an ANN

```
random.seed(42)
tfidf_model = keras.models.Sequential([
    layers.Input((X_train_txt.shape[1],)),
    layers.Dense(250, "relu"),
    layers.Dense(1, "sigmoid")
])

tfidf_model.compile("adam", "binary_crossentropy", metrics=["accuracy"])
tfidf_model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_8 (Dense)	(None, 250)	250,250
dense_9 (Dense)	(None, 1)	251

Total params: 250,501 (978.52 KB)
Trainable params: 250,501 (978.52 KB)
Non-trainable params: 0 (0.00 B)

UNSW SYDNEY

Fit & evaluate

```
es = keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True,
    monitor="val_accuracy", verbose=2)

if not Path("tfidf-model.keras").exists():
    tfidf_model.fit(X_train_txt, y_train, epochs=1_000, callbacks=es,
        validation_data=(X_val_txt, y_val), verbose=0)
    tfidf_model.save("tfidf-model.keras")
else:
    tfidf_model = keras.models.load_model("tfidf-model.keras")
```

```
tfidf_model.evaluate(X_train_txt, y_train, verbose=0, batch_size=1_000)
```

```
[0.11705566942691803, 0.9575437903404236]
```

```
tfidf_model.evaluate(X_val_txt, y_val, verbose=0, batch_size=1_000)
```

```
[0.3212660849094391, 0.8848921060562134]
```

UNSW SYDNEY

Keep text as sequence of tokens

```python
max_length = 500
max_tokens = 1_000
vect = layers.TextVectorization(
    max_tokens=max_tokens,
    output_sequence_length=max_length,
    standardize="lower_and_strip_punctuation",
)

vect.adapt(X_train)
vocab = vect.get_vocabulary()

X_train_txt = vect(X_train)
X_val_txt = vect(X_val)
X_test_txt = vect(X_test)

print(vocab[:50])
```

```
['', '[UNK]', 'the', 'was', 'a', 'to', 'of', 'and', 'in', 'driver', 'for', 'this', 'vehicle',
'critical', 'lane', 'he', 'on', 'with', 'that', 'left', 'roadway', 'coded', 'she', 'event',
'crash', 'not', 'at', 'intersection', 'traveling', 'right', 'precrash', 'as', 'from', 'were',
'by', 'had', 'reason', 'his', 'side', 'is', 'front', 'her', 'traffic', 'an', 'it', 'two',
'speed', 'stated', 'one', 'occurred']
```

UNSW SYDNEY

A sequence of integers

```
1 X_train_txt[0]
```

```
<tf.Tensor: shape=(500,), dtype=int64, numpy=
array([ 11,  24,  49,   8,   2, 253, 219,   6,   4, 165,   8,   2, 410,
         6,   4, 564, 971,  27,   2,  27, 568,   6,   4, 192,   1,  45,
        51, 208,  65, 235,  54,  14,  20, 867,  34,  43, 183,   1,  45,
        51, 208,  65, 235,  54,  14,  20, 178,  34,   4, 676,   1,  42,
       237,   2, 153, 192,  20,   3, 107,   7,  75,  17,   4, 612, 441,
       549,   2,  88,  46,   3, 207,  63, 185,  55,   2,  42, 243,   3,
       400,   7,  58,  33,  50, 172, 251,  84,  26,   2,  60,   6,   2,
        24,   1,   4, 402, 970,   1,   1,   3,  68,  26,   2,  27,  94,
       118,   8,  14, 101, 311,  10,   2, 237,   5, 422, 269,  44, 154,
        54,  19,   1,   4, 308, 342,   1,   3,  79,   8,  14,  45, 159,
         2, 121,  27, 190,  44, 598,   5, 325,  75,  70,   2, 105, 189,
       231,   1, 241,  81,  19,  31,   1, 193,   2,  54,  81,   9, 134,
         4, 174,  12,  17,   1, 390,   1, 159,   2,  27,  32,   2, 119,
         1,  68,   8,   2, 410,   6,   2,  27,   8,   1,   5,   2, 159,
       174,  12,   1, 168,   2,  27,   7,  69,   2,  40,   6,   1,  17,
        81,  40,  19, 246,  73,  83,  64,   5, 129,  56,   8,   2,  27,
         7,  33,  73,  71,  57,   5,  82,   2,   9,   6,   1,   4,   1,
        59, 382,   5, 113,   8, 276, 258,   1, 317, 928, 284,  10, 784,
```

UNSW SYDNEY

Feed LSTM a sequence of one-hots

```
random.seed(42)
inputs = keras.Input(shape=(max_length,), dtype="int64")
onehot = keras.ops.one_hot(inputs, max_tokens)
x = layers.Bidirectional(layers.LSTM(24))(onehot)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
one_hot_model = keras.Model(inputs, outputs)
one_hot_model.compile(optimizer="adam",
    loss="binary_crossentropy", metrics=["accuracy"])
one_hot_model.summary()
```

Model: "functional_8"

Layer (type)	Output Shape	Param #
input_layer_5 (InputLayer)	(None, 500)	0
one_hot (OneHot)	(None, 500, 1000)	0
bidirectional (Bidirectional)	(None, 48)	196,800
dropout (Dropout)	(None, 48)	0
dense_10 (Dense)	(None, 1)	49

Total params: 196,849 (768.94 KB)
Trainable params: 196,849 (768.94 KB)
Non-trainable params: 0 (0.00 B)

UNSW SYDNEY

Fit & evaluate

```
es = keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True,
    monitor="val_accuracy", verbose=2)

if not Path("one-hot-model.keras").exists():
    one_hot_model.fit(X_train_txt, y_train, epochs=1_000, callbacks=es,
        validation_data=(X_val_txt, y_val), verbose=0);
    one_hot_model.save("one-hot-model.keras")
else:
    one_hot_model = keras.models.load_model("one-hot-model.keras")
```

```
one_hot_model.evaluate(X_train_txt, y_train, verbose=0, batch_size=1_000)
```

```
[0.3969860076904297, 0.8304149508476257]
```

```
one_hot_model.evaluate(X_val_txt, y_val, verbose=0, batch_size=1_000)
```

```
[0.5179389715194702, 0.7496402859687805]
```

UNSW SYDNEY

Custom embeddings

```
inputs = keras.Input(shape=(max_length,), dtype="int64")
embedded = layers.Embedding(input_dim=max_tokens, output_dim=32,
        mask_zero=True)(inputs)
x = layers.Bidirectional(layers.LSTM(24))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
embed_lstm = keras.Model(inputs, outputs)
embed_lstm.compile("adam", "binary_crossentropy", metrics=["accuracy"])
embed_lstm.summary()
```

Model: "functional_10"

Layer (type)	Output Shape	Param #	Connected to
input_layer_6 (InputLayer)	(None, 500)	0	-
embedding (Embedding)	(None, 500, 32)	32,000	input_layer_6[0]…
not_equal (NotEqual)	(None, 500)	0	input_layer_6[0]…
bidirectional_1 (Bidirectional)	(None, 48)	10,944	embedding[0][0], not_equal[0][0]
dropout_1 (Dropout)	(None, 48)	0	bidirectional_1[…
dense_11 (Dense)	(None, 1)	49	dropout_1[0][0]

Total params: 42,993 (167.94 KB)
Trainable params: 42,993 (167.94 KB)

UNSW SYDNEY

Fit & evaluate

```
es = keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True,
    monitor="val_accuracy", verbose=2)

if not Path("embed-lstm.keras").exists():
    embed_lstm.fit(X_train_txt, y_train, epochs=1_000, callbacks=es,
        validation_data=(X_val_txt, y_val), verbose=0);
    embed_lstm.save("embed-lstm.keras")
else:
    embed_lstm = keras.models.load_model("embed-lstm.keras")
```

```
embed_lstm.evaluate(X_train_txt, y_train, verbose=0, batch_size=1_000)
```

```
[0.5322403311729431, 0.7265531420707703]
```

```
embed_lstm.evaluate(X_val_txt, y_val, verbose=0, batch_size=1_000)
```

```
[0.5577294826507568, 0.7043165564537048]
```

```
embed_lstm.evaluate(X_test_txt, y_test, verbose=0, batch_size=1_000)
```

```
[0.5559249520301819, 0.7100719213485718]
```

UNSW SYDNEY

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       : 3.11.9
IPython version      : 8.24.0

keras     : 3.3.3
matplotlib: 3.8.4
numpy     : 1.26.4
pandas    : 2.2.2
seaborn   : 0.13.2
scipy     : 1.11.0
torch     : 2.0.1
tensorflow: 2.16.1
tf_keras  : 2.16.0
```

UNSW SYDNEY

Glossary

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