Introduction to NLP

We will cover *practical* NLP:

- how to process text
- how to do useful things with it

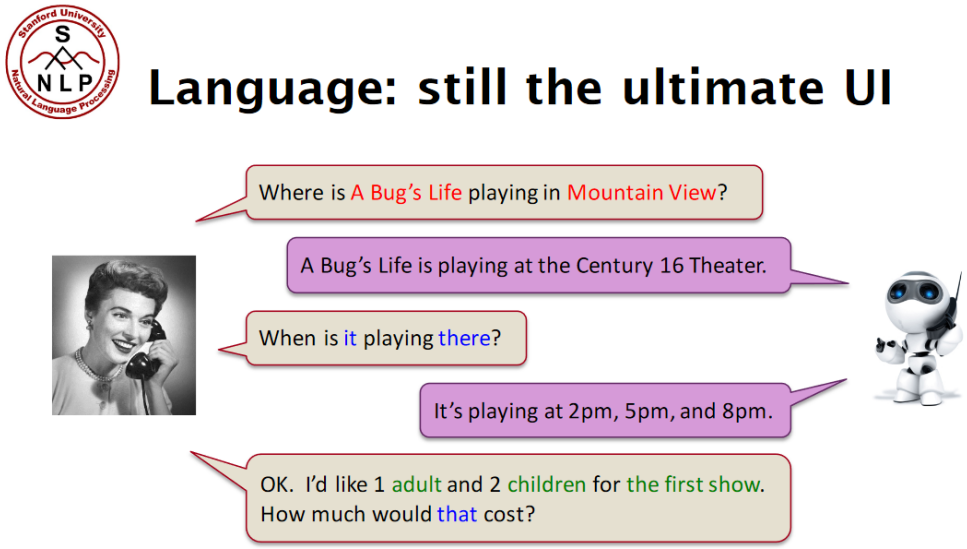
... we will not cover *theoretical* NLP:

- grammars, syntax, linguistic models
- (sorry Noam)

To run this notebook, you'll need need to have the IMDb dataset extracted in the `aclImdb` subfolder. This can be downloaded from: https://ai.stanford.edu/~amaas/data/sentiment/

Then run this in this folder: `tar -xf aclImdb_v1.tar.gz`

Why NLP?



[1] CS224N

Why is NLP hard?

San Jose cops kill man with knife

Ex-college football player, 23, shot 9 times allegedly charged police at fiancee's home

Why is NLP hard?

McDonald's Fries the Holy Grail for Potato Farmers

Wednesday, September 23, 2009
Associated Press

Print

KIMBERLY, Idaho — From the fields of Idaho to tasting rooms in suburban Chicago, potato farmers, researchers and industry representatives are in the midst of an elusive hunt: finding a new spud for McDonald's french fries.

Garden-path sentence (wiki)

Why is NLP hard?

Home | Recent | Popular | Features | National | Crime | Entertainment | Politics | Business | Technol

« BACK TO ENTERTAINMENT / ARTS TOP

print

Violinist linked to JAL crash blossoms

By May Masangkay

The requested article has expired, and is no longer available.
Any related articles, and user comments are shown below.

Crash Blossoms

Why is NLP hard?

- ambiguous
- complex
- context dependent

- requires prior knowledge
- requires a world model
- is used for interaction with people (amusing, insulting)

Famous NLP successes

- GPT-3
- Google Translate
- AI Dungeon
- DALL-E

Words as data

Let's start with some very short reviews, and our goal is to classify whether they are positive (1) or negative (0).

In [1]:
```
sentences = [
    'We had a good time.',
    'This was a good-looking place.',
    'A once in a lifetime experience.',
    'Horrible.',
    'One of the worst experiences of my life.',
    "Everything went terribly, I'll never return."
]
y = [1, 1, 1, 0, 0, 0]
```

- We need to transform `sentence` into a matrix of features (X)
- How would you do this?

Many ways to skin a cat!

- Memorize the sentences

In [2]:
```
model = {sentences[i]: y[i] for i in range(len(sentences))}
match = [model[sentence] == y[i] for i, sentence in enumerate(sentences)]
print(f'{sum(match) / len(sentences):3.1%} accuracy.')
```

```
100.0% accuracy.
```

What if we try a new sentence..

In [3]:
```
sentences_test = [
    'This was fun!',
    'Abysmal service.'
]
y_test = [1, 0]
```

In [4]:
```
try:
    match = [model[sentence] == y[i] for i, sentence in enumerate(sentences_test
    print(f'{sum(match) / len(sentences_test):3.1%} accuracy.')
except KeyError as err:
    print(f'Error!')
```

```
Error!

---------------------------------------------------------------------------
---
KeyError                                  Traceback (most recent call
last)
<ipython-input-5-1d2b9ac74b7f> in <module>
----> 1 match = [model[sentence] == y[i] for i, sentence in
enumerate(sentences_test)]
      2 print(f'{sum(match) / len(sentences_test):3.1%} accuracy.')

<ipython-input-5-1d2b9ac74b7f> in <listcomp>(.0)
----> 1 match = [model[sentence] == y[i] for i, sentence in
enumerate(sentences_test)]
      2 print(f'{sum(match) / len(sentences_test):3.1%} accuracy.')

KeyError: 'This was fun!'
```

Yikes!

Our model failed at out of sample data. Sentences are too specific - learning that a specific sentence is positive or negative does not help us generalize to other sentences. Clearly we need a new approach!

What if we picked a more generic, broadly applicable symbol? Characters come to mind!

Many ways to skin a cat!

- One feature for every letter in the alphabet

In [5]:
```
# utility functions for creating a graph
import numpy as np

from sklearn import tree

# used to display trees
import pydotplus
from IPython.display import Image

%matplotlib inline

def create_graph(mdl, cmap=None, feature_names=None):
    # cmap is a colormap
    # e.g. cmap = matplotlib.cm.coolwarm( np.linspace(0.0, 1.0, 256, dtype=float
    tree_graph = tree.export_graphviz(mdl, out_file=None,
                                      feature_names=feature_names,
                                      filled=True, rounded=True)
```

```
    graph = pydotplus.graphviz.graph_from_dot_data(tree_graph)

    # get colormap
    if cmap:
        # remove transparency
        if cmap.shape[1]==4:
            cmap = cmap[:,0:2]

        nodes = graph.get_node_list()
        for node in nodes:
            if node.get_label():
                # get number of samples in group 1 and group 2
                num_samples = [int(ii) for ii in node.get_label().split('value =

                # proportion that is class 2
                cm_value = float(num_samples[1]) / float(sum(num_samples))
                # convert to (R, G, B, alpha) tuple
                cm_value = matplotlib.cm.coolwarm(cm_value)
                cm_value = [int(np.ceil(255*x)) for x in cm_value]
                color = '#{:02x}{:02x}{:02x}'.format(cm_value[0], cm_value[1], c
                node.set_fillcolor(color)

    Image(graph.create_png())
    return graph
```

Prepare the dataset

In [6]:

```
# utility function for preparing the data
def prepare_data(sentences, alphabet='abcdefghijklmnopqrstuvwxyz'):
    return [[a in sentence for a in alphabet] for sentence in sentences]

X = prepare_data(sentences)
print(X[0])
```

```
[True, False, False, True, True, False, True, True, True, False, False, False, T
rue, False, True, False, False, False, False, True, False, False, False, False,
False, False]
```

Train the model

In [7]:

```
mdl = tree.DecisionTreeClassifier(max_depth=3)

print(X[0])
# fit the model to the data - trying to predict y from X
mdl = mdl.fit(X, y)
```

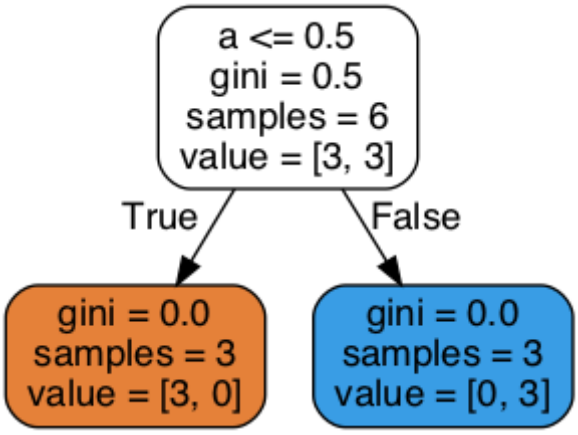
```
[True, False, False, True, True, False, True, True, True, False, False, False, T
rue, False, True, False, False, False, False, True, False, False, False, False,
False, False]
```

Trained model

In [8]:

```
graph = create_graph(mdl, feature_names=list('abcdefghijklmnopqrstuvwxyz'))
Image(graph.create_png())
```

Out[8]:



In [9]:

```
print([(y[i], sentence) for i, sentence in enumerate(sentences)])
```

```
[(1, 'We had a good time.'), (1, 'This was a good-looking place.'), (1, 'A once
in a lifetime experience.'), (0, 'Horrible.'), (0, 'One of the worst experiences
of my life.'), (0, "Everything went terribly, I'll never return.")]
```

Our model has learned that the letter "a" is always present in positive sentences, and never in negative sentences.

Try our model on new data

In [10]:

```
X_test = prepare_data(sentences_test)
pred_test = mdl.predict(X_test)

match = [pred_test[i] == y_test[i] for i, sentence in enumerate(sentences_test)]
print(f'{sum(match) / len(sentences_test):3.1%} accuracy.')
print([(pred_test[i], sentence) for i, sentence in enumerate(sentences_test)])
```

```
50.0% accuracy.
[(1, 'This was fun!'), (1, 'Abysmal service.')]
```

That's... bad. Our model clearly does not generalize. It turns out reviews with the letter "a" may be negative!

NLP has many levels

- Symbol/Character
- Lexeme
- Word
- Sentence

NLP has many levels

- ~Symbol/Character~ - too low level
- ~Lexeme~ - we don't know what this means
- Word

- ~Sentence~ - too high level

Words as data

- Before: alphabet = [a, b, c, d, ...]
- Now: alphabet: [good, bad, movie, ...]

In [11]:
```
import itertools

alphabet = sorted(list(set(
    [word for sentence in sentences
     for word in sentence.split(' ')]
)))
```

In [12]:
```
print(alphabet)
```

```
['A', 'Everything', 'Horrible.', "I'll", 'One', 'This', 'We', 'a', 'experienc
e.', 'experiences', 'good', 'good-looking', 'had', 'in', 'life.', 'lifetime', 'm
y', 'never', 'of', 'once', 'place.', 'return.', 'terribly,', 'the', 'time.', 'wa
s', 'went', 'worst']
```

The above list.. isn't great.

- punctuation included in words
- "experience" and "experiences" separate
- "A" and "a" are different
- "good-looking" vs "good"

> Hrm, that's annoying.

This should be the motto for NLP.

Words as data

- Focus on the word "good-looking".
- Should we split this into two words, "good" and "looking"?
- What about `I'll` ? Strictly speaking, it's two words: `I` and `will` .

Tokenization

- In NLP, we don't work with "words", we work with "tokens"
- *Tokens* are groups of symbols with some semantic meaning
- Tokenization is the process of converting text into individual tokens

Tokenizers

- spaCy's English tokenizer
- nltk's English tokenizer
- ... etc

Trying it out

We'll work with spaCy's tokenizer.

In [13]:
```
# download the English model we will use - only needs to be run once
# this model is small (~7 MB)
!python -m spacy download en_core_web_sm
```

```
Collecting en-core-web-sm==3.0.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_cor
e_web_sm-3.0.0/en_core_web_sm-3.0.0-py3-none-any.whl (13.7 MB)
     |████████████████████████████████| 13.7 MB 3.3 MB/s eta 0:00:01
| 1.4 MB 3.3 MB/s eta 0:00:04
Requirement already satisfied: spacy<3.1.0,>=3.0.0 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from en-core-web-sm==3.0.0)
(3.0.3)
Requirement already satisfied: numpy>=1.15.0 in /Users/alistairewj/miniconda3/en
vs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-core-we
b-sm==3.0.0) (1.20.1)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /Users/alistairewj/minic
onda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en
-core-web-sm==3.0.0) (3.0.5)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in /Users/alistairewj/minicond
a3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-co
re-web-sm==3.0.0) (0.7.4)
Requirement already satisfied: typing-extensions>=3.7.4 in /Users/alistairewj/mi
niconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0-
>en-core-web-sm==3.0.0) (3.7.4.3)
Requirement already satisfied: catalogue<2.1.0,>=2.0.1 in /Users/alistairewj/min
iconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->
en-core-web-sm==3.0.0) (2.0.1)
Requirement already satisfied: typer<0.4.0,>=0.3.0 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-c
ore-web-sm==3.0.0) (0.3.2)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.0 in /Users/alistairewj/
miniconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.
0->en-core-web-sm==3.0.0) (3.0.1)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-c
ore-web-sm==3.0.0) (2.0.5)
Requirement already satisfied: setuptools in /Users/alistairewj/miniconda3/envs/
nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-core-web-s
m==3.0.0) (49.6.0.post20210108)
Requirement already satisfied: thinc<8.1.0,>=8.0.0 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-c
ore-web-sm==3.0.0) (8.0.1)
Requirement already satisfied: pydantic<1.8.0,>=1.7.1 in /Users/alistairewj/mini
conda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->e
n-core-web-sm==3.0.0) (1.7.3)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /Users/alistairewj/m
iniconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0
->en-core-web-sm==3.0.0) (1.0.5)
Requirement already satisfied: importlib-metadata>=0.20 in /Users/alistairewj/mi
niconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0-
>en-core-web-sm==3.0.0) (3.7.2)
Requirement already satisfied: srsly<3.0.0,>=2.4.0 in /Users/alistairewj/minicon
```

```
da3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-c
ore-web-sm==3.0.0) (2.4.0)
Requirement already satisfied: pathy in /Users/alistairewj/miniconda3/envs/nlp_l
ecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.
0.0) (0.4.0)
Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in /Users/alistairewj/minico
nda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-
core-web-sm==3.0.0) (0.8.2)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-c
ore-web-sm==3.0.0) (4.59.0)
Requirement already satisfied: jinja2 in /Users/alistairewj/miniconda3/envs/nlp_
lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==
3.0.0) (2.11.3)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /Users/alistairewj/min
iconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->
en-core-web-sm==3.0.0) (2.25.1)
Requirement already satisfied: packaging>=20.0 in /Users/alistairewj/miniconda3/
envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-core-
web-sm==3.0.0) (20.9)
Requirement already satisfied: zipp>=0.5 in /Users/alistairewj/miniconda3/envs/n
lp_lecture/lib/python3.7/site-packages (from importlib-metadata>=0.20->spacy<3.
1.0,>=3.0.0->en-core-web-sm==3.0.0) (3.4.1)
Requirement already satisfied: pyparsing>=2.0.2 in /Users/alistairewj/miniconda
3/envs/nlp_lecture/lib/python3.7/site-packages (from packaging>=20.0->spacy<3.1.
0,>=3.0.0->en-core-web-sm==3.0.0) (2.4.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/alistairewj/minic
onda3/envs/nlp_lecture/lib/python3.7/site-packages (from requests<3.0.0,>=2.13.0
->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (1.26.3)
Requirement already satisfied: idna<3,>=2.5 in /Users/alistairewj/miniconda3/env
s/nlp_lecture/lib/python3.7/site-packages (from requests<3.0.0,>=2.13.0->spacy<
3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in /Users/alistairewj/miniconda
3/envs/nlp_lecture/lib/python3.7/site-packages (from requests<3.0.0,>=2.13.0->sp
acy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (4.0.0)
Requirement already satisfied: certifi>=2017.4.17 in /Users/alistairewj/minicond
a3/envs/nlp_lecture/lib/python3.7/site-packages (from requests<3.0.0,>=2.13.0->s
pacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (2020.12.5)
Requirement already satisfied: click<7.2.0,>=7.1.1 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from typer<0.4.0,>=0.3.0->spac
y<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (7.1.2)
Requirement already satisfied: MarkupSafe>=0.23 in /Users/alistairewj/miniconda
3/envs/nlp_lecture/lib/python3.7/site-packages (from jinja2->spacy<3.1.0,>=3.0.0
->en-core-web-sm==3.0.0) (1.1.1)
Requirement already satisfied: smart-open<4.0.0,>=2.2.0 in /Users/alistairewj/mi
niconda3/envs/nlp_lecture/lib/python3.7/site-packages (from pathy->spacy<3.1.0,>
=3.0.0->en-core-web-sm==3.0.0) (2.2.1)
Requirement already satisfied: boto3 in /Users/alistairewj/miniconda3/envs/nlp_l
ecture/lib/python3.7/site-packages (from smart-open<4.0.0,>=2.2.0->pathy->spacy<
3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (1.17.22)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /Users/alistairewj/mini
conda3/envs/nlp_lecture/lib/python3.7/site-packages (from boto3->smart-open<4.0.
0,>=2.2.0->pathy->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (0.10.0)
Requirement already satisfied: botocore<1.21.0,>=1.20.22 in /Users/alistairewj/m
iniconda3/envs/nlp_lecture/lib/python3.7/site-packages (from boto3->smart-open<
4.0.0,>=2.2.0->pathy->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (1.20.22)
Requirement already satisfied: s3transfer<0.4.0,>=0.3.0 in /Users/alistairewj/mi
niconda3/envs/nlp_lecture/lib/python3.7/site-packages (from boto3->smart-open<4.
0.0,>=2.2.0->pathy->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (0.3.4)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /Users/alistairew
j/miniconda3/envs/nlp_lecture/lib/python3.7/site-packages (from botocore<1.21.0,
>=1.20.22->boto3->smart-open<4.0.0,>=2.2.0->pathy->spacy<3.1.0,>=3.0.0->en-core-
web-sm==3.0.0) (2.8.1)
Requirement already satisfied: six>=1.5 in /Users/alistairewj/miniconda3/envs/nl
p_lecture/lib/python3.7/site-packages (from python-dateutil<3.0.0,>=2.1->botocor
```

```
e<1.21.0,>=1.20.22->boto3->smart-open<4.0.0,>=2.2.0->pathy->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (1.15.0)
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

In [14]:
```
import spacy
nlp = spacy.load("en_core_web_sm") # load an English model we already downloaded
```

In [15]:
```
for sentence in sentences:
    doc = nlp(sentence)
    print(sentence, end=': ')
    print([token for token in doc])
```

```
We had a good time.: [We, had, a, good, time, .]
This was a good-looking place.: [This, was, a, good, -, looking, place, .]
A once in a lifetime experience.: [A, once, in, a, lifetime, experience, .]
Horrible.: [Horrible, .]
One of the worst experiences of my life.: [One, of, the, worst, experiences, of, my, life, .]
Everything went terribly, I'll never return.: [Everything, went, terribly, ,, I, 'll, never, return, .]
```

Trying it out

We can try other sentences to explore the tokenizer's quirks.

In [16]:
```
sentence = "To be or not to be; that is the question. Or at least it was Hamlet's question."
doc = nlp(sentence)
print([token for token in doc])
```

```
[To, be, or, not, to, be, ;, that, is, the, question, ., Or, at, least, it, was, Hamlet, 's, question, .]
```

IMDb

Let's try out spaCy's tokenizer on a large dataset from IMDb curated by researchers [1].

- 50,000 reviews of movies on IMDb
- 25,000 positive reviews (>=7), 25,000 negative reviews (<=4)
- Dataset commonly used for sentiment classification

> [1] Maas A, Daly RE, Pham PT, Huang D, Ng AY, Potts C. Learning word vectors for sentiment analysis. In Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies 2011 Jun (pp. 142-150).

In [17]:
```
import os
from pathlib import Path
from tqdm.notebook import tqdm

def load_data(path):
```

```
    corpus = {}
    files = os.listdir(path)
    for fn in tqdm(files):
        filename = path / fn
        with open(filename, 'r') as fp:
            text = ''.join(fp.readlines())
        corpus[filename.stem] = text

    return corpus
```

In [18]:

```
print('Loading positive documents (training set).')
pos = load_data(Path('aclImdb/train/pos'))
print('Loading negative documents (training set).')
neg = load_data(Path('aclImdb/train/neg'))

print('Loading positive documents (test set).')
pos_test = load_data(Path('aclImdb/test/pos'))
print('Loading negative documents (test set).')
neg_test = load_data(Path('aclImdb/test/neg'))
```

```
Loading positive documents (training set).

Loading negative documents (training set).

Loading positive documents (test set).

Loading negative documents (test set).
```

In [19]:

```
def get_numpy_arrays(pos, neg):
    # create numpy arrays of the datasets
    X_pos = np.array(list(pos.values()))
    X_neg = np.array(list(neg.values()))
    X = np.hstack([X_pos, X_neg])
    n_pos, n_neg = X_pos.shape[0], X_neg.shape[0]
    y = np.concatenate([np.ones(n_pos), np.zeros(n_neg)])
    return X, y

# only use ~1000 examples for speed
X_train, y_train = get_numpy_arrays(
    {k: v for k, v in pos.items() if k.startswith('2')},
    {k: v for k, v in neg.items() if k.startswith('2')}
)
X_test, y_test = get_numpy_arrays(
    {k: v for k, v in pos_test.items() if k.startswith('2')},
    {k: v for k, v in neg_test.items() if k.startswith('2')}
)
```

Example review

In [20]:

```
print(pos['10_9'])
```

```
I'm a male, not given to women's movies, but this is really a well done special
story. I have no personal love for Jane Fonda as a person but she does one Hell
of a fine job, while DeNiro is his usual superb self. Everything is so well don
e: acting, directing, visuals, settings, photography, casting. If you can enjoy
a story of real people and real love - this is a winner.
```

Stanley and Iris

Text classification

- Recall, the goal of this exercise was to *classify text*
  - classify e-mails as spam or not spam
  - tweets as inappropriate or appropriate
- We are trying to create a matrix `X` to classify targets `y`

Tokenize the review

In [21]:
```
review = pos['10_9']
doc = nlp(review)
print([token for token in doc])
```

```
[I, 'm, a, male, ,, not, given, to, women, 's, movies, ,, but, this, is, really,
a, well, done, special, story, ., I, have, no, personal, love, for, Jane, Fonda,
as, a, person, but, she, does, one, Hell, of, a, fine, job, ,, while, DeNiro, i
```

```
s, his, usual, superb, self, ., Everything, is, so, well, done, :, acting, ,, di
recting, ,, visuals, ,, settings, ,, photography, ,, casting, ., If, you, can, e
njoy, a, story, of, real, people, and, real, love, -, this, is, a, winner, .]
```

- `'m` and `'s` are given their own token
- punctuation marks are individual tokens
- whitespaces are individual tokens

As you can see, tokens *do not have to be words*!

... how do we turn this into `X` ?

Creating the data matrix

- Recall when we used the alphabet
- Each column was a letter
- Each row was our sentence
  - `X[i,j] == 0` -> letter is in the document
  - `X[i,j] == 1` -> letter is not in the document

In [22]:
```
columns = list('abcdefghijklmnopqrstuvwxyz')
print(sentences[0])
print(columns)
print(X[0])
```

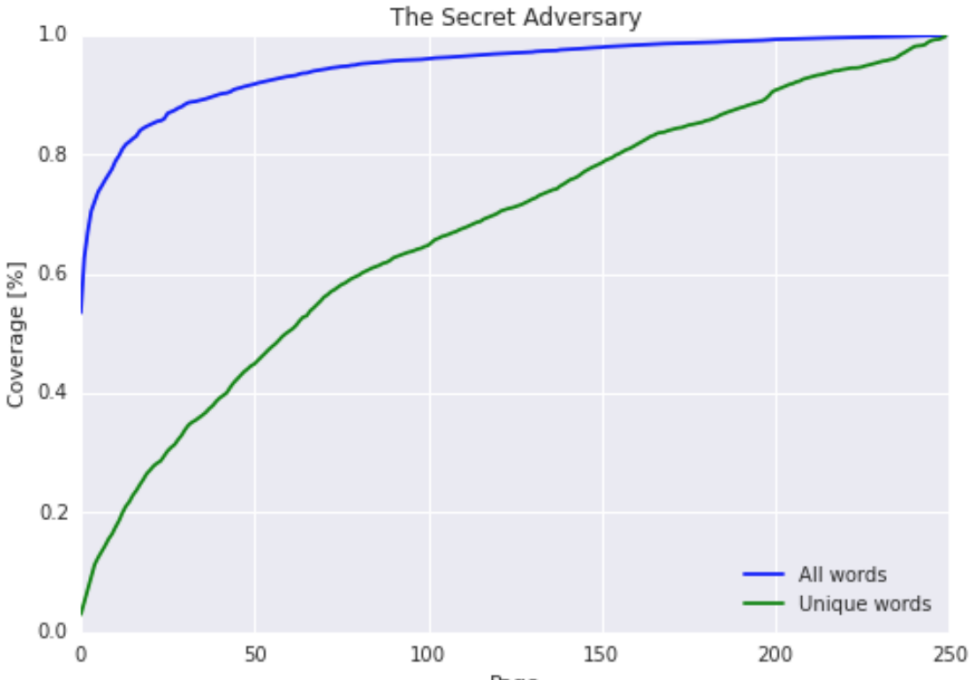
```
We had a good time.
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
[True, False, False, True, True, False, True, True, True, False, False, False, T
rue, False, True, False, False, False, False, True, False, False, False, False,
False, False]
```

- We are using tokens instead of letters
- Which tokens should we include as columns?

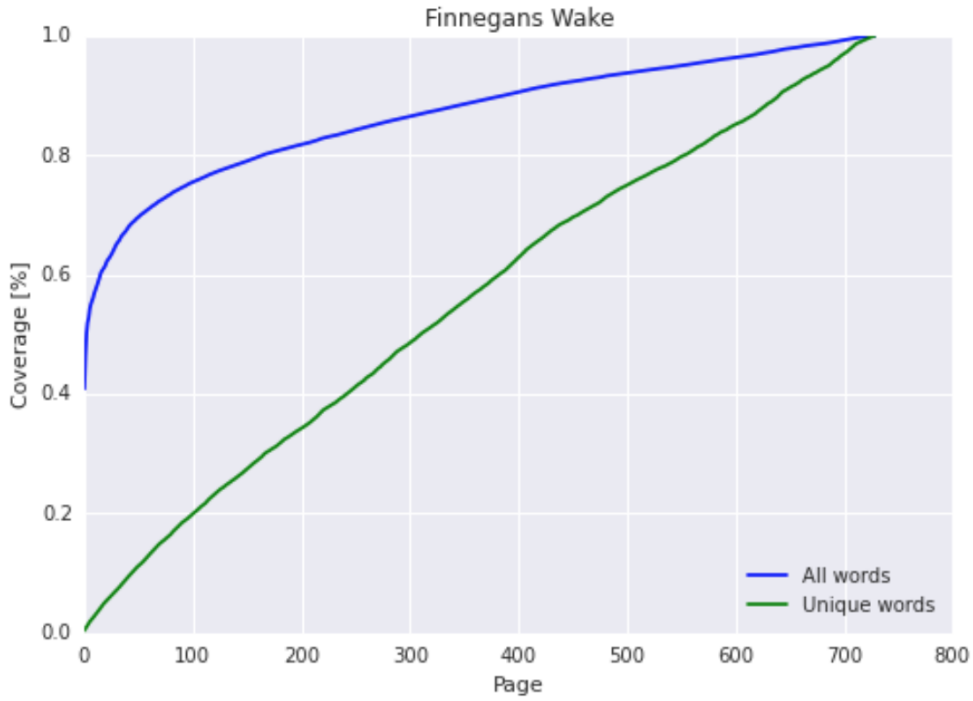
Vocabulary

- The set of tokens we choose is our *vocabulary*.
- Most vocabularies consist of the most common tokens.

Common words in books



- 250 pages, 75,208 words.
- You will know 90% of words after 40 pages which are 16.00% of the book.
- At that page, you will know 39.21% of unique words.



- 729 pages, 218,793 words.
- You will know 90% of words after 387 pages which are 53.09% of the book.
- At that page, you will know 60.64% of unique words.

We can grab the 10 most frequently used tokens in the IMDb dataset.

In [23]:
```
from collections import Counter

token_freq = Counter()
```

```
for corpus in [pos, neg]:
    for text in tqdm(corpus.values(), total=len(corpus)):
        # only call the tokenizer for speed purposes
        doc = nlp.tokenizer(text)

        for token in doc:
            # use 'orth', the integer version of the string
            # will convert back to string version later
            token_freq[token.orth] += 1

# convert back to string
token_freq = Counter({nlp.vocab.strings[k]: v for k, v in token_freq.items()})
```

What do you think the top 10 tokens are?

In [24]:
```
token_freq.most_common(10)
```

Out[24]:
```
[('the', 289838),
 (',', 275296),
 ('.', 236702),
 ('and', 156484),
 ('a', 156282),
 ('of', 144056),
 ('to', 133886),
```

```
('is', 109095),
('in', 87676),
('I', 77546)]
```

Stop-words

- The tokens "the", ",", etc have little meaning on their own
- We call these "stop-words".
- Let's remove them for now.

Why are these stop-words not useful for our current approach?

We've destroyed the document structure! The existence of punctuation is almost always meaningless.

Using sklearn for modelling

- Pipeline
- CountVectorizer
- User Guide on text feature extraction

First we will define a few utility functions.

In [25]:

```
from spacy.tokens import Token
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn import metrics

def train_pipe_and_predict(pipe, X_train, y_train, X_test):
    """Convenience function for training pipeline."""
    print('Building model.')
    pipe.fit(X_train, y_train)

    print('Getting test predictions.')

    # Predicting with a test dataset
    pred = pipe.predict(X_test)

    return pred

def print_results(y_test, pred):
    # Model Accuracy
    print(f"Accuracy: {metrics.accuracy_score(y_test, pred):3.1%}")
    print(f"Precision: {metrics.precision_score(y_test, pred):3.1%}")
    print(f"Recall: {metrics.recall_score(y_test, pred):3.1%}")

def introspect_text(pipe, text):
    print(text, end='\n\n')
    vectorizer = pipe.named_steps['vectorizer']
    freq = vectorizer.transform([text])[0].toarray()
    features = vectorizer.get_feature_names()

    # get the prediction
    pred = pipe.predict([text])
    if pred[0] == 0:
```

```
        print('Prediction: NEGATIVE.')
    else:
        print('Prediction: POSITIVE.')
    for i, feat in enumerate(features):
        print(f'{feat:10s} {freq[0][i]:3d}')
```

In [26]:
```
def spacy_tokenizer(text):
    return [token.text for token in nlp.tokenizer(text)]
```

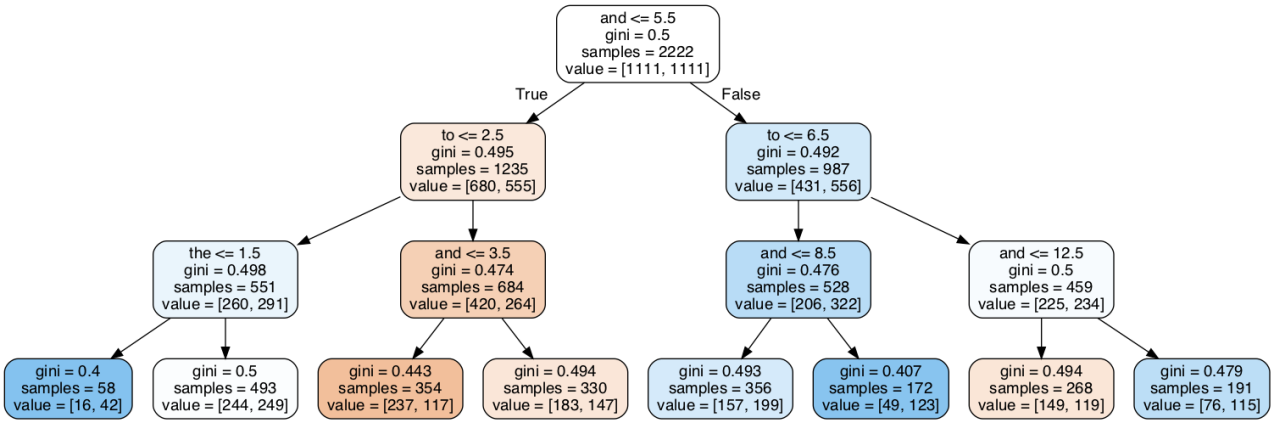
In [27]:
```
pipe = Pipeline([('vectorizer', CountVectorizer(tokenizer=spacy_tokenizer, max_f
                 ('classifier', tree.DecisionTreeClassifier(max_depth=3))])

# pipeline will be updated in-place!
pred = train_pipe_and_predict(pipe, X_train, y_train, X_test)
```

```
Building model.
Getting test predictions.
```

In [28]:
```
# Model Accuracy
print_results(y_test, pred)

vectorizer = pipe.named_steps['vectorizer']
classifier = pipe.named_steps['classifier']

graph = create_graph(classifier, feature_names=vectorizer.get_feature_names())
Image(graph.create_png())
```

```
Accuracy: 56.3%
Precision: 55.6%
Recall: 62.3%
```

Out[28]:



In [29]:
```
text = pos['10_9']
print(text, end='\n\n')
freq = vectorizer.transform([text])[0].toarray()
features = vectorizer.get_feature_names()
for i, feat in enumerate(features):
    print(f'{feat:10s} {freq[0][i]:3d}')
```

```
I'm a male, not given to women's movies, but this is really a well done special
story. I have no personal love for Jane Fonda as a person but she does one Hell
of a fine job, while DeNiro is his usual superb self. Everything is so well don
e: acting, directing, visuals, settings, photography, casting. If you can enjoy
a story of real people and real love - this is a winner.
```

```
,           8
.           4
a           6
and         1
in          0
is          4
it          0
of          2
the         0
to          1
```

Stop words

- We can try to remove the stop words and see how the model improves
- Also remove punctuation, for the same reason

In [30]:
```
def spacy_tokenizer(text):
    return [token.text for token in nlp.tokenizer(text)
            if not token.is_punct and not token.is_stop]
```

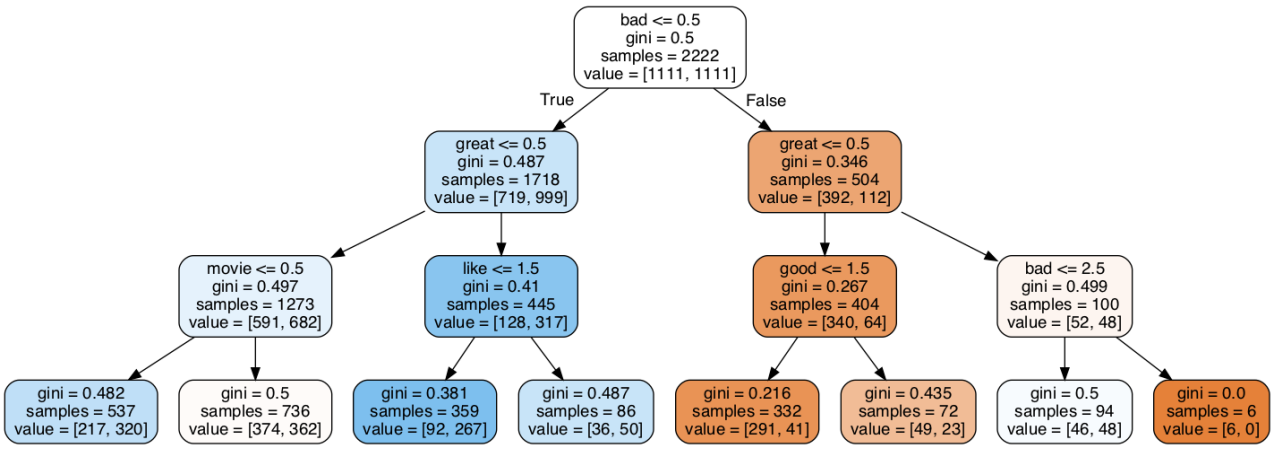
In [31]:
```
pipe = Pipeline([('vectorizer', CountVectorizer(tokenizer=spacy_tokenizer, max_f
                 ('classifier', tree.DecisionTreeClassifier(max_depth=3))])

pred = train_pipe_and_predict(pipe, X_train, y_train, X_test)
```

```
Building model.
Getting test predictions.
```

In [32]:
```
print_results(y_test, pred)

vectorizer = pipe.named_steps['vectorizer']
classifier = pipe.named_steps['classifier']

graph = create_graph(classifier, feature_names=vectorizer.get_feature_names())
Image(graph.create_png())
```

```
Accuracy: 63.0%
Precision: 63.2%
Recall: 62.2%
```

Out[32]:



In [33]:
```
text = pos['10_9']
introspect_text(pipe, text)
```

```
I'm a male, not given to women's movies, but this is really a well done special
story. I have no personal love for Jane Fonda as a person but she does one Hell
of a fine job, while DeNiro is his usual superb self. Everything is so well don
e: acting, directing, visuals, settings, photography, casting. If you can enjoy
a story of real people and real love - this is a winner.

Prediction: POSITIVE.
/><br        0
bad          0
film         0
good         0
great        0
like         0
movie        0
people       1
story        2
time         0
```

Increase the vocabulary size

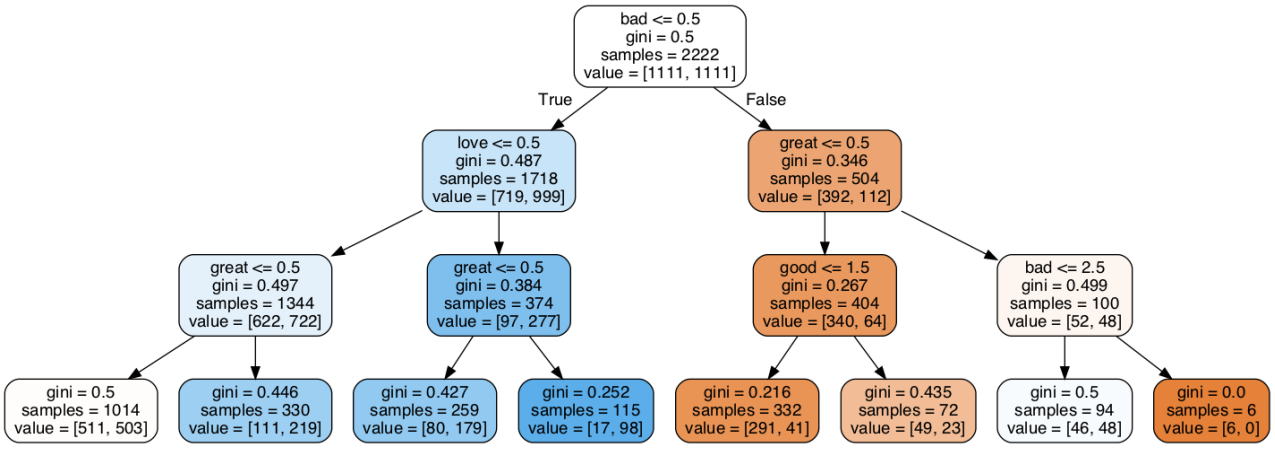
In [34]:
```
pipe = Pipeline([('vectorizer', CountVectorizer(tokenizer=spacy_tokenizer, max_f
                 ('classifier', tree.DecisionTreeClassifier(max_depth=3))])

pred = train_pipe_and_predict(pipe, X_train, y_train, X_test)
```

```
Building model.
Getting test predictions.
```

In [35]:
```
print_results(y_test, pred)

vectorizer = pipe.named_steps['vectorizer']
classifier = pipe.named_steps['classifier']

graph = create_graph(classifier, feature_names=vectorizer.get_feature_names())
Image(graph.create_png())
```

```
Accuracy: 63.2%
Precision: 68.9%
Recall: 48.2%
```

Out[35]:



In [36]:
```
text = pos['10_9']
introspect_text(pipe, text)
```

```
I'm a male, not given to women's movies, but this is really a well done special
```

```
story. I have no personal love for Jane Fonda as a person but she does one Hell
of a fine job, while DeNiro is his usual superb self. Everything is so well don
e: acting, directing, visuals, settings, photography, casting. If you can enjoy
a story of real people and real love - this is a winner.

Prediction: POSITIVE.
/><br         0
/>the         0
<             0
acting        1
bad           0
br            0
film          0
films         0
good          0
great         0
like          0
love          2
movie         0
movies        1
people        1
story         2
think         0
time          0
watch         0
way           0
```

Notice the "movie" and "movies" - we have (functionally) the same word which is appearing twice. This is unideal - we would like to combine these together as the information we want to learn from a dataset would be the same for both.

Stemming

- Many words have similar meaning but distinct morphology
  - this was entertaining // i was entertained
- Removing the end of the word is effective in grouping like terms
  - movies -> movie
  - entertaining -> entertain
- This process is called "stemming"

Stemming implementations

- Porter Stemmer: Most common, non-aggressive
- Snowball Stemmer: Improvement over Porter Stemmer

Snowball Stemmer implementation is on GitHub.

```
define Step_2 as (
    [substring] R1 among (
        'tional'  (<-'tion')
        'enci'    (<-'ence')
        'anci'    (<-'ance')
```

In [37]:

```
from nltk.stem.snowball import SnowballStemmer
```

```
stemmer = SnowballStemmer(language='english')

def spacy_tokenizer(text):
    return [stemmer.stem(token.text) for token in nlp.tokenizer(text) if not tok
```

In [38]:

```
# Create pipeline using Bag of Words
pipe = Pipeline([('vectorizer', CountVectorizer(tokenizer=spacy_tokenizer, max_f
                 ('classifier', tree.DecisionTreeClassifier(max_depth=3))])

pred = train_pipe_and_predict(pipe, X_train, y_train, X_test)
```

```
Building model.
Getting test predictions.
```

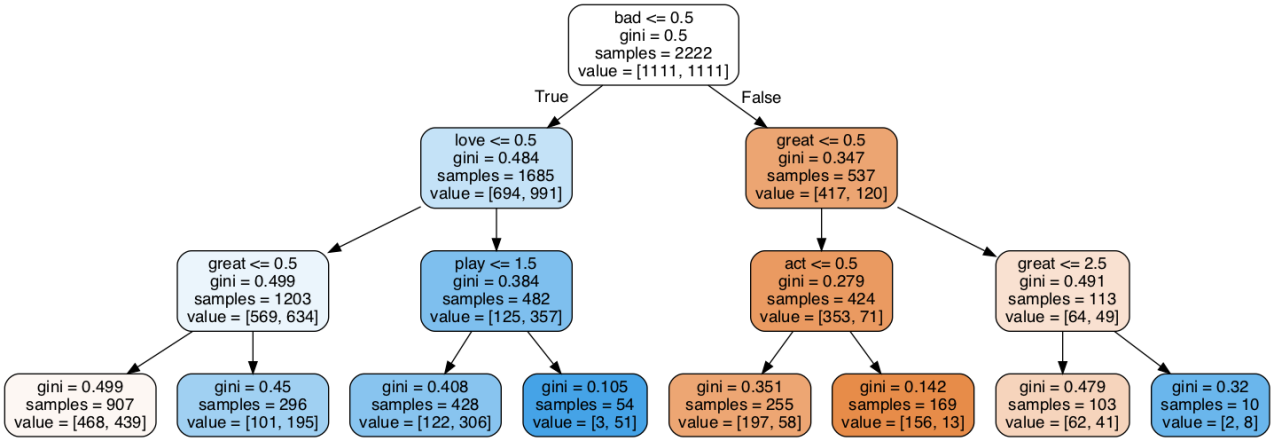
Current best (stop word removal): 63.2%.

In [39]:

```
print_results(y_test, pred)

graph = create_graph(pipe.named_steps['classifier'], feature_names=pipe.named_st
Image(graph.create_png())
```

```
Accuracy: 65.8%
Precision: 72.8%
Recall: 50.2%
```

Out[39]:



In [40]:

```
text = pos['10_9']
introspect_text(pipe, text)
```

```
I'm a male, not given to women's movies, but this is really a well done special
story. I have no personal love for Jane Fonda as a person but she does one Hell
of a fine job, while DeNiro is his usual superb self. Everything is so well don
e: acting, directing, visuals, settings, photography, casting. If you can enjoy
a story of real people and real love - this is a winner.

Prediction: POSITIVE.
/><br         0
act           1
bad           0
charact       0
end           0
film          0
good          0
great         0
like          0
look          0
love          2
```

```
movi          1
peopl         1
play          0
scene         0
stori         2
think         0
time          0
watch         0
way           0
```

- No longer dictionary words
- words like character and characters are combined into charact
- words like play and playing are also combined

Lemmatization

- With stemming, we applied an algorithm to simplify the text
- With lemmatization, we use *knowledge* - a lexicon
  - swing == swung

In [41]:
```
text = pos['10_9']
doc = nlp(text)
for token in doc[:4]:
    print(f'{token.text:15s} -> {token.lemma_:15s}')
```

```
I               -> I
'm              -> be
a               -> a
male            -> male
```

In [42]:
```
def spacy_tokenizer(text):
    # note we call nlp(text) now
    # lemmatizing is slow compared to tokenizing!
    return [token.lemma_ for token in nlp(text) if not token.is_punct and not to
```

In [43]:
```
# Create pipeline using Bag of Words
pipe = Pipeline([('vectorizer', CountVectorizer(tokenizer=spacy_tokenizer, max_f
                 ('classifier', tree.DecisionTreeClassifier(max_depth=3))])

pred = train_pipe_and_predict(pipe, X_train, y_train, X_test)
```

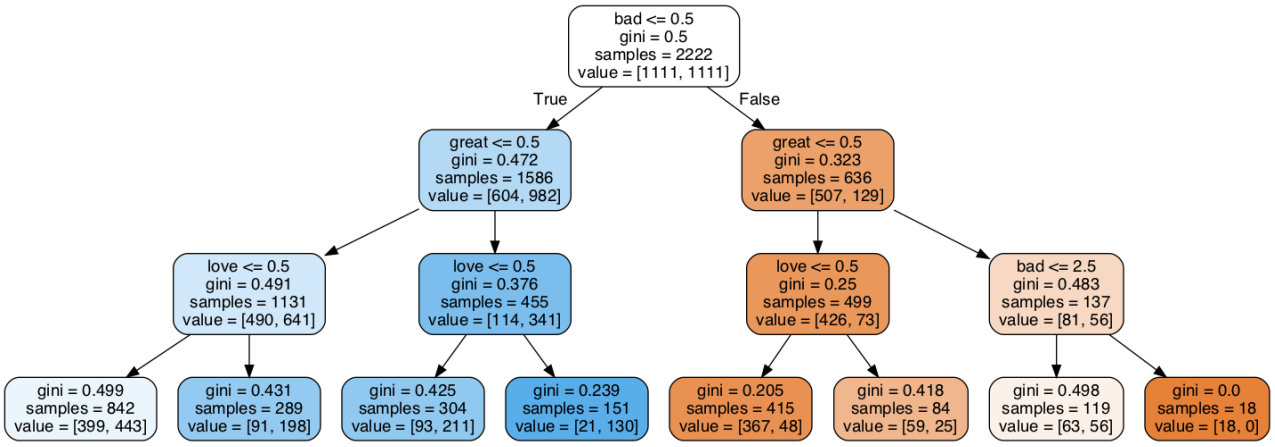
```
Building model.
Getting test predictions.
```

Current best (stop word removal + stemming): 65.8%

In [44]:
```
print_results(y_test, pred)

graph = create_graph(pipe.named_steps['classifier'], feature_names=pipe.named_st
Image(graph.create_png())
```

```
Accuracy: 68.2%
Precision: 63.0%
Recall: 88.1%
```

Out[44]:



What else could we do to improve performance?

I'm going to skin a cat.

There's more than one way to skin a cat.

> I'm going to **skin a cat**
>
> More than one way to **skin a cat**

- Context matters
- Words take on new meaning when used with other, specific words

Word combinations have meaning

- "hidden gem"

- So far we have one feature per token
- What if we had a feature for pairs of tokens?
  - -> token bigrams

In [45]:
```
# Create pipeline using Bag of Words
pipe = Pipeline([('vectorizer',
                  CountVectorizer(
                      tokenizer=spacy_tokenizer,
                      max_features=20,
```

```
                        ngram_range=(1, 2)
                    )),
                   ('classifier', tree.DecisionTreeClassifier(max_depth=3))])

pred = train_pipe_and_predict(pipe, X_train, y_train, X_test)
```

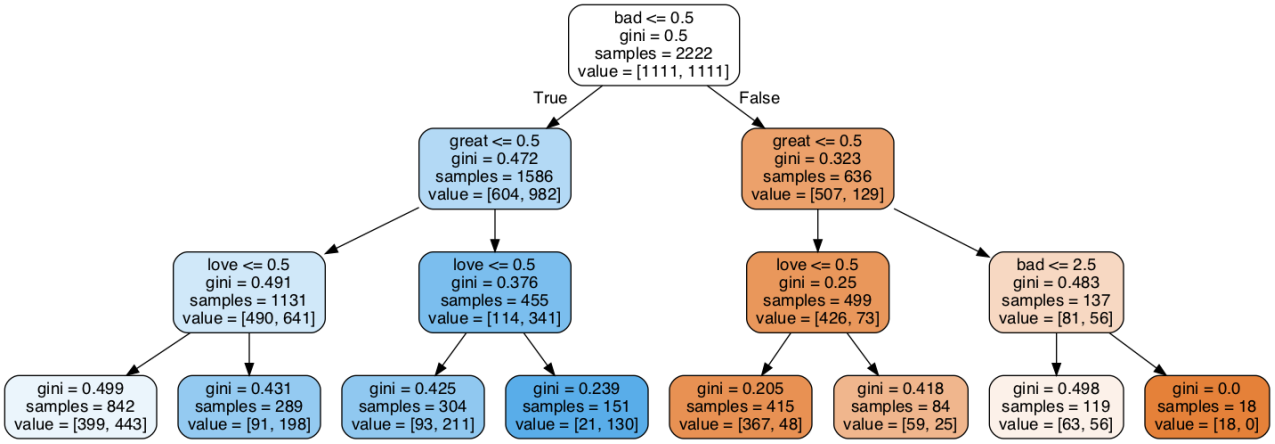
```
Building model.
Getting test predictions.
```

In [46]:

```
print_results(y_test, pred)

graph = create_graph(pipe.named_steps['classifier'], feature_names=pipe.named_st
Image(graph.create_png())
```

```
Accuracy: 68.2%
Precision: 63.0%
Recall: 88.1%
```

Out[46]:



What else could we do to improve performance?

... honestly, whatever we can think of!

Let's look at an example from a recent paper.

Named Entity Recognition

Classify the individual tokens in a text.

In [47]:

```
text = "Alistair Johnson saw me last Monday the 24th."
targets = ['NAME', 'NAME', 'O', 'O', 'O', 'DATE', 'DATE', 'DATE']

for i, token in enumerate(text.split(' ')):
    print(f'{token:15s}{targets[i]}')
```

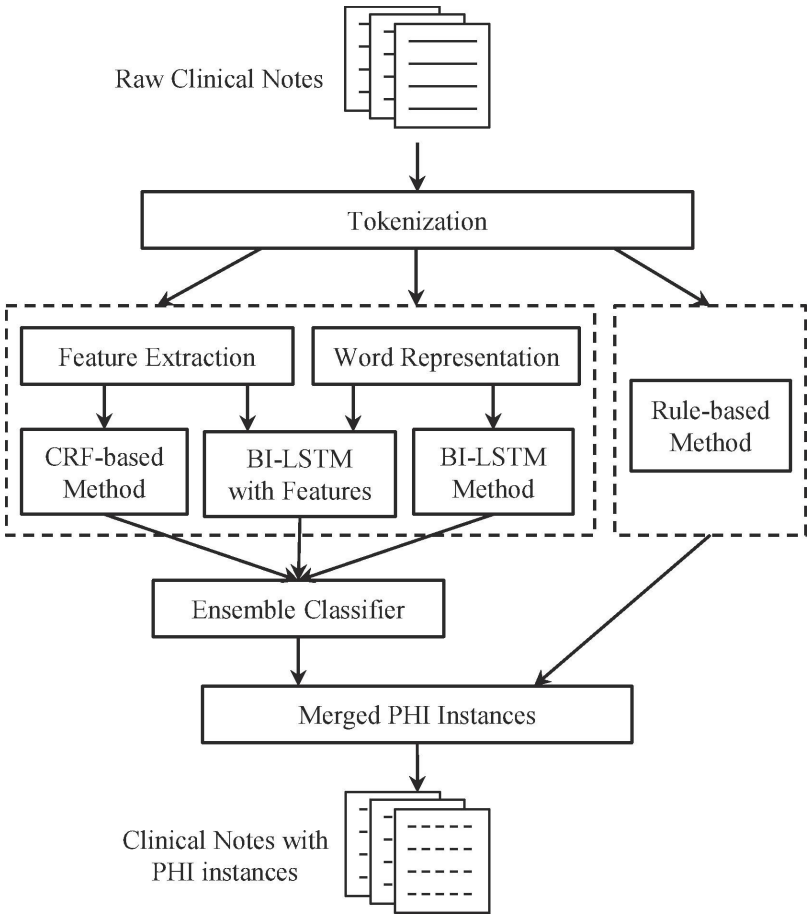
```
Alistair        NAME
Johnson         NAME
saw             O
me              O
last            O
Monday          DATE
```

```
the             DATE
24th.           DATE
```

One interesting NLP task is *deidentification*: label words in a document as names, dates, etc. This task is extremely important for healthcare applications, and research data usually needs to be deidentified before use.

In [48]:
```
for i, token in enumerate(text.split(' ')):
    print(f'{token:15s} -> x -> f(x) ~= {targets[i]}')
```

```
Alistair        -> x -> f(x) ~= NAME
Johnson         -> x -> f(x) ~= NAME
saw             -> x -> f(x) ~= O
me              -> x -> f(x) ~= O
last            -> x -> f(x) ~= O
Monday          -> x -> f(x) ~= DATE
the             -> x -> f(x) ~= DATE
24th.           -> x -> f(x) ~= DATE
```



[1] Lee HJ, Wu Y, Zhang Y, Xu J, Xu H, Roberts K. A hybrid approach to automatic de-identification of psychiatric notes. Journal of biomedical informatics. 2017 Nov 1;75:S19-27. https://doi.org/10.1016/j.jbi.2017.06.006

What type of features can we build?

- **Bag-of-words**: unigrams, bigrams and trigrams of words within a window of [-2, 2].

Like we did before, except now it's word specific features.

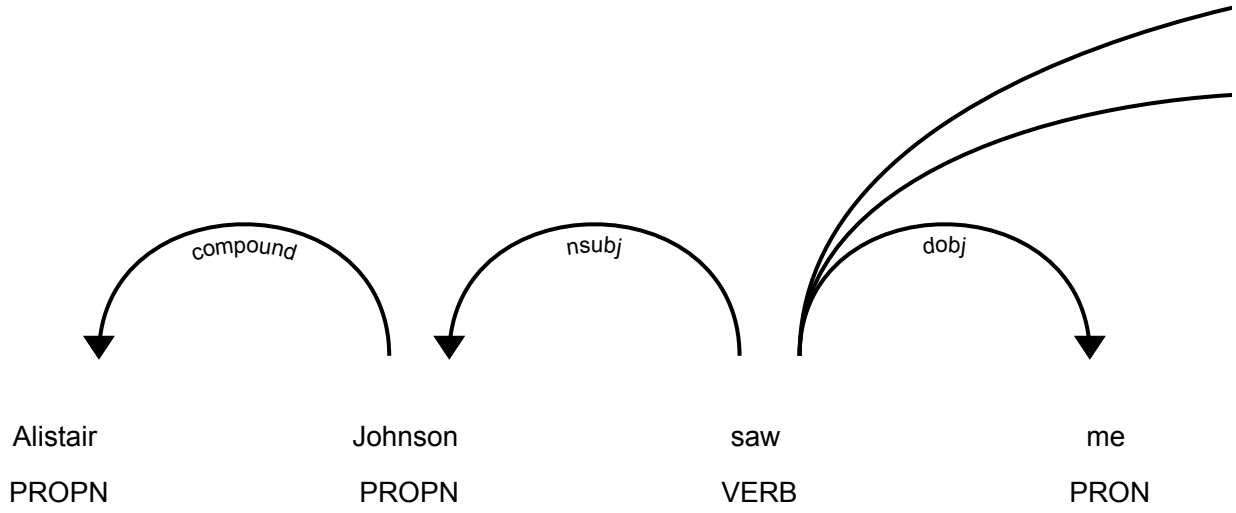
In [49]:
```
unigrams = ['Monday', 'Tuesday']
print(f'{"Entity":15s} -> [{", ".join(unigrams)}, ...]')
tokens = text.split(' ')
for i, token in enumerate(tokens):
    start, stop = max(i-2, 0), min(i+2+1, len(tokens))
    feature = [gram in tokens[start:stop] for gram in unigrams]

    print(f'{token:15s} -> {", ".join(map(str, feature))}, ...')
```

```
Entity          -> [Monday, Tuesday, ...]
Alistair        -> False, False, ...
Johnson         -> False, False, ...
saw             -> False, False, ...
me              -> True, False, ...
last            -> True, False, ...
Monday          -> True, False, ...
the             -> True, False, ...
24th.           -> True, False, ...
```

If we look at the word "me" - it is two words before Monday, so the feature is True (1). Contrast this with "saw", which is too many words away. This is a very rudimentary way of encoding context into the feature for a word.

- **Bag-of-words**: unigrams, bigrams and trigrams of words within a window of [-2, 2].
- **Part-of-speech (POS) tags**: unigrams, bigrams and trigrams of POS tags within a window of [−2, 2]. The Stanford POS Tagger [42] was used for POS tagging.

In [50]:
```
# reload spacy model for dependency parsing
nlp = spacy.load('en_core_web_sm')
doc = nlp(text)

from spacy import displacy
displacy.render(doc, style='dep', jupyter=True)
```



The above is a dependency parse of the sentence:

- Notice how there are words which is a root: "saw"
- The `nsubj` of "saw" in "Johnson"
- "Johnson" is a proper noun (`PROPN`).

Our model is described online: https://spacy.io/models/en#en_core_web_sm

Though we can use `spacy.explain` to figure out what a tag means.

spacy.explain

spaCy has nifty functionality for understanding some of its outputs:

In [51]:
```
for token in doc:
    print(f'{token.text:15s}{token.tag_:5s}{spacy.explain(token.tag_)}')
```

```
Alistair       NNP  noun, proper singular
Johnson        NNP  noun, proper singular
saw            VBD  verb, past tense
me             PRP  pronoun, personal
last           JJ   adjective
Monday         NNP  noun, proper singular
the            DT   determiner
24th           NN   noun, singular or mass
.              .    punctuation mark, sentence closer
```

- **Bag-of-words**: unigrams, bigrams and trigrams of words within a window of [-2, 2].
- **Part-of-speech (POS) tags**: unigrams, bigrams and trigrams of POS tags within a window of [−2, 2]. The Stanford POS Tagger [42] was used for POS tagging.

- **Combinations of words and POS tags**: combining current word with the unigrams, bigrams and trigrams of POS tags within a window of [−1, 1], i.e. w0p−1, w0p0, w0p1, w0p−1p0, w0p0p1, w0p−1p1, w0p−1p0p1, where w0, p−1, p0 and p1 denote current word, last, current and next POS tags respectively.
- **Sentence information**: number of words in current sentence, whether there is an end mark at the end of current sentence such as '.', '?' and '!', whether there is any bracket unmatched in current sentence.
- **Affixes**: prefixes and suffixes of length from 1 to 5.
- **Orthographical features**: whether the word is upper case, contains uppercase characters, contains punctuation marks, contains digits, etc.
- **Word shapes**: mapping any or consecutive uppercase character(s), lowercase character(s), digit(s) and other character(s) in current word to ' A', ' a', ' #' and ' -' respectively. For instance, the word shapes of "Hospital" are "Aaaaaaaa" and "Aa".

Focus on a specific feature

- **Word shapes**: mapping any or consecutive uppercase character(s), lowercase character(s), digit(s) and other character(s) in current word to ' A', ' a', ' #' and ' -' respectively. For instance, the word shapes of "Hospital" are "Aaaaaaaa" and "Aa".

In [52]:
```
word_shapes = []
for token in tokens:
    shape = ''
    for letter in token:
        if letter in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':
            shape += 'A'
        elif letter in 'abcdefghijklmnopqrstuvwxtz':
            shape += 'a'
        elif letter in '0123456789':
            shape += '#'
        else:
            shape += '-'
    word_shapes.append(shape)
    print(f'{token:15s} {shape}')
```

```
Alistair        Aaaaaaaa
Johnson         Aaaaaaa
saw             aaa
me              aa
last            aaaa
Monday          Aaaaa-
the             aaa
24th.           ##aa-
```

Note the mistake - forgot a 'y' in one of the if statements.

There must be a simpler way!

Regular expressions (regex)

- Regex: match patterns in text

  - Concise and powerful pattern matching language
  - Supported by many computer languages, including SQL
- Challenges
  - Brittle
  - Hard to write, can get complex to be correct
  - Hard to read

In [53]:
```
import re

text_modified = str(text)
print(text_modified)
```

```
Alistair Johnson saw me last Monday the 24th.
```

Replace capital letters with 'A'.

In [54]:
```
text_modified = re.sub('[A-Z]', 'A', text_modified)
print(text_modified)
```

```
Alistair Aohnson saw me last Aonday the 24th.
```

Replace lower case letters with 'a'.

In [55]:
```
text_modified = re.sub('[a-z]', 'a', text_modified)
print(text_modified)
```

```
Aaaaaaaa Aaaaaaa aaa aa aaaa Aaaaaa aaa 24aa.
```

Replace numbers with '#'.

In [56]:
```
text_modified = re.sub('[0-9]', '#', text_modified)
print(text_modified)
```

```
Aaaaaaaa Aaaaaaa aaa aa aaaa Aaaaaa aaa ##aa.
```

Replace the rest with '-'.

In [57]:
```
text_modified = re.sub('[^Aa# ]', '-', text_modified)
print(text_modified)
```

```
Aaaaaaaa Aaaaaaa aaa aa aaaa Aaaaaa aaa ##aa-
```

Features extracted in this *one* paper

- **Bag-of-words**: unigrams, bigrams and trigrams of words within a window of [-2, 2].
- **Part-of-speech (POS) tags**: unigrams, bigrams and trigrams of POS tags within a window of [−2, 2]. The Stanford POS Tagger [42] was used for POS tagging.
- **Combinations of words and POS tags**: combining current word with the unigrams, bigrams and trigrams of POS tags within a window of [−1, 1], i.e. w0p−1, w0p0, w0p1, w0p−1p0, w0p0p1, w0p−1p1, w0p−1p0p1, where w0, p−1, p0 and p1 denote current word, last, current and next POS tags respectively.

- **Sentence information**: number of words in current sentence, whether there is an end mark at the end of current sentence such as ′.′, ′?′ and ′!′, whether there is any bracket unmatched in current sentence.
- **Affixes**: prefixes and suffixes of length from 1 to 5.
- **Orthographical features**: whether the word is upper case, contains uppercase characters, contains punctuation marks, contains digits, etc.
- **Word shapes**: mapping any or consecutive uppercase character(s), lowercase character(s), digit(s) and other character(s) in current word to ′ A′, ′ a′, ′ #′ and ′ -′ respectively. For instance, the word shapes of "Hospital" are "Aaaaaaaa" and "Aa".
- **Section information**: twenty-nine section headers (see the supplementary file) were collected manually such as "History of Present Illness"; we check which section current word belongs to.
- **General NER information**: the Stanford Named Entity Recognizer [43] was used to generate the NER tags of current word, include: person, date, organization, location, and number tags, etc.

Features extracted in this *one* paper

Happily, many NLP libraries have these features built in!

For example, named entity recognition.

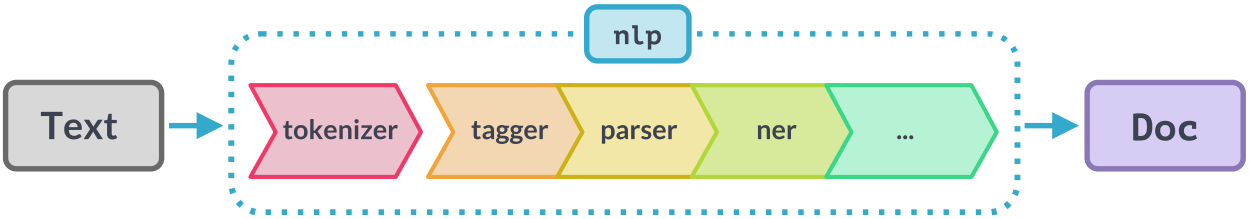
In [58]:
```
displacy.render(doc, style='ent', jupyter=True)
```

Alistair Johnson **PERSON** saw me last Monday the 24th **DATE** .

Recap

- Tokenize: split text into individual meaningful units
- Pre-process: remove stop words, remove punctuation
- Stemming: remove the suffix of words to merge similar words together
- Lemmatize: use a lexicon to merge similar words together
- Extract features: do whatever you think makes sense!
  - spaCy linguistic features

spaCy pipeline



https://spacy.io/usage/processing-pipelines

How do I use this knowledge?

Look at the data.

In [59]:
```
n = 0
for k, v in pos.items():
    print('='*10)
    print(f'{k}\n---')
    print(v)
    print('='*10)
    n += 1
    if n >= 5:
        break
```

```
==========
4715_9
---
For a movie that gets no respect there sure are a lot of memorable quotes listed
for this gem. Imagine a movie where Joe Piscopo is actually funny! Maureen Stapl
eton is a scene stealer. The Moroni character is an absolute scream. Watch for A
lan "The Skipper" Hale jr. as a police Sgt.
==========
==========
12390_8
---
Bizarre horror movie filled with famous faces but stolen by Cristina Raines (lat
er of TV's "Flamingo Road") as a pretty but somewhat unstable model with a gummy
smile who is slated to pay for her attempted suicides by guarding the Gateway to
Hell! The scenes with Raines modeling are very well captured, the mood music is
perfect, Deborah Raffin is charming as Cristina's pal, but when Raines moves int
o a creepy Brooklyn Heights brownstone (inhabited by a blind priest on the top f
loor), things really start cooking. The neighbors, including a fantastically wic
ked Burgess Meredith and kinky couple Sylvia Miles & Beverly D'Angelo, are a dia
bolical lot, and Eli Wallach is great fun as a wily police detective. The movie
is nearly a cross-pollination of "Rosemary's Baby" and "The Exorcist"--but what
a combination! Based on the best-seller by Jeffrey Konvitz, "The Sentinel" is en
tertainingly spooky, full of shocks brought off well by director Michael Winner,
who mounts a thoughtfully downbeat ending with skill. ***1/2 from ****
==========
==========
8329_7
---
A solid, if unremarkable film. Matthau, as Einstein, was wonderful. My favorite
part, and the only thing that would make me go out of my way to see this again,
was the wonderful scene with the physicists playing badmitton, I loved the sweat
ers and the conversation while they waited for Robbins to retrieve the birdie.
==========
==========
9063_8
---
It's a strange feeling to sit alone in a theater occupied by parents and their r
ollicking kids. I felt like instead of a movie ticket, I should have been given
a NAMBLA membership.<br /><br />Based upon Thomas Rockwell's respected Book, How
To Eat Fried Worms starts like any children's story: moving to a new town. The n
ew kid, fifth grader Billy Forrester was once popular, but has to start anew. Ma
king friends is never easy, especially when the only prospect is Poindexter Ada
m. Or Erica, who at 4 1/2 feet, is a giant.<br /><br />Further complicating thin
gs is Joe the bully. His freckled face and sleeveless shirts are daunting. He an
tagonizes kids with the Death Ring: a Crackerjack ring that is rumored to kill y
ou if you're punched with it. But not immediately. No, the death ring unleashes
a poison that kills you in the eight grade.<br /><br />Joe and his axis of evil
```

```
welcome Billy by smuggling a handful of slimy worms into his thermos. Once disco
vered, Billy plays it cool, swearing that he eats worms all the time. Then he th
rows them at Joe's face. Ewww! To win them over, Billy reluctantly bets that he
can eat 10 worms. Fried, boiled, marinated in hot sauce, squashed and spread on
a peanut butter sandwich. Each meal is dubbed an exotic name like the "Radioacti
ve Slime Delight," in which the kids finally live out their dream of microwaving
a living organism.<br /><br />If you've ever met me, you'll know that I have an
uncontrollably hearty laugh. I felt like a creep erupting at a toddler whining t
hat his "dilly dick" hurts. But Fried Worms is wonderfully disgusting. Like a G-
rated Farrelly brothers film, it is both vomitous and delightful.<br /><br />Wri
ter/director Bob Dolman is also a savvy storyteller. To raise the stakes the wor
ms must be consumed by 7 pm. In addition Billy holds a dark secret: he has an ul
tra-sensitive stomach.<br /><br />Dolman also has a keen sense of perspective. W
ith such accuracy, he draws on children's insecurities and tendency to exaggerat
e mundane dilemmas.<br /><br />If you were to hyperbolize this movie the way kid
s do their quandaries, you will see that it is essentially about war. Freedom-fi
ghter and freedom-hater use pubescent boys as pawns in proxy wars, only to learn
a valuable lesson in unity. International leaders can learn a thing or two about
global peacekeeping from Fried Worms.<br /><br />At the end of the film, I was c
omforted when two chaperoning mothers behind me, looked at each other with befud
dlement and agreed, "That was a great movie." Great, now I won't have to registe
r myself in any lawful databases.
==========
==========
3092_10
---
You probably all already know this by now, but 5 additional episodes never aired
can be viewed on ABC.com I've watched a lot of television over the years and thi
s is possibly my favorite show, ever. It's a crime that this beautifully written
and acted show was canceled. The actors that played Laura, Whit, Carlos, Mae, Da
mian, Anya and omg, Steven Caseman - are all incredible and so natural in those
roles. Even the kids are great. Wonderful show. So sad that it's gone. Of course
I wonder about the reasons it was canceled. There is no way I'll let myself beli
eve that Ms. Moynahan's pregnancy had anything to do with it. It was in the perf
ect time slot in this market. I've watched all the episodes again on ABC.com - I
hope they all come out on DVD some day. Thanks for reading.
==========
```

We already have some ideas:

- need to preprocess out `<br></br>`
- many proper names mentioned which would be movie specific, perhaps we should group them?
- sometimes users specifically state the review in the text (undesirable for our model to fixate on that)

Moving beyond one token per feature

Let's pretend we have 3 token features.

In [60]:

```
text = 'The queen is doing a much better job than the king.'
columns = ['queen', 'king', 'bishop']

features = [int(c in text) for c in columns]
print(columns)
print(features)
```

```
['queen', 'king', 'bishop']
[1, 1, 0]
```

- We have a world model, and we know queen/king are similar concepts
- ... but in the features, they are totally separate
- we don't *share* knowledge between "queen" and "king"

Distributional hypothesis

- Does language have a distributional structure? [1]

structure:

> A set of phonemes or a set of data is structured in respect to some feature, to the extent that we can form in terms of that feature some organized system of statements which describes the members of the set and their interrelations (at least up to some limit of complexity).

distributional:

> The distribution of an element will be understood as the sum of all its environments. An environment of an element A is an existing array of its co-occurrents, i.e. the other elements, each in a particular position, with which A occurs to yield an utterance.

-> you're known by the company you keep

[1] Harris ZS. Distributional structure. Word. 1954 Aug 1;10(2-3):146-62. PDF.

Distributional hypothesis

We could manually create these features.

- "The queen of England" -> feature 1: England is mentioned nearby
- "The king of Spain" -> feature 2: Spain is mentioned nearby
- "The queen of Russia" -> feature 3: some proper country is mentioned nearby
- "Ascended the throne as queen" -> feature 4: some regal stuff is nearby

... this would get exhausting.

word2vec

Learn these features, rather than manually create them.

1. Take a whole bunch of text
2. Try to predict a word's *context* form the word itself ("skip-grams")
3. Use the model's representation of words (tokens)

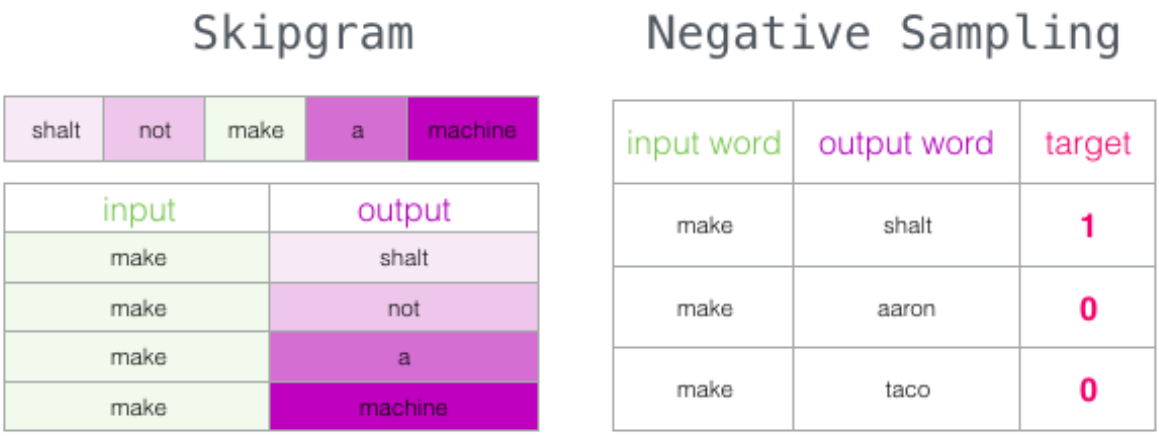
(target word, context word)

Window Size	Text	Skip-grams
2	[The **wide** road shimmered] in the hot sun.	wide, the wide, road wide, shimmered
	The [wide road **shimmered** in the] hot sun.	shimmered, wide shimmered, road shimmered, in shimmered, the
	The wide road shimmered in [the hot **sun**].	sun, the sun, hot
3	[The **wide** road shimmered in] the hot sun.	wide, the wide, road wide, shimmered wide, in
	[The wide road **shimmered** in the hot] sun.	shimmered, the shimmered, wide shimmered, road shimmered, in shimmered, the shimmered, hot
	The wide road shimmered [in the hot **sun**].	sun, in sun, the sun, hot

[1] https://www.tensorflow.org/tutorials/text/word2vec

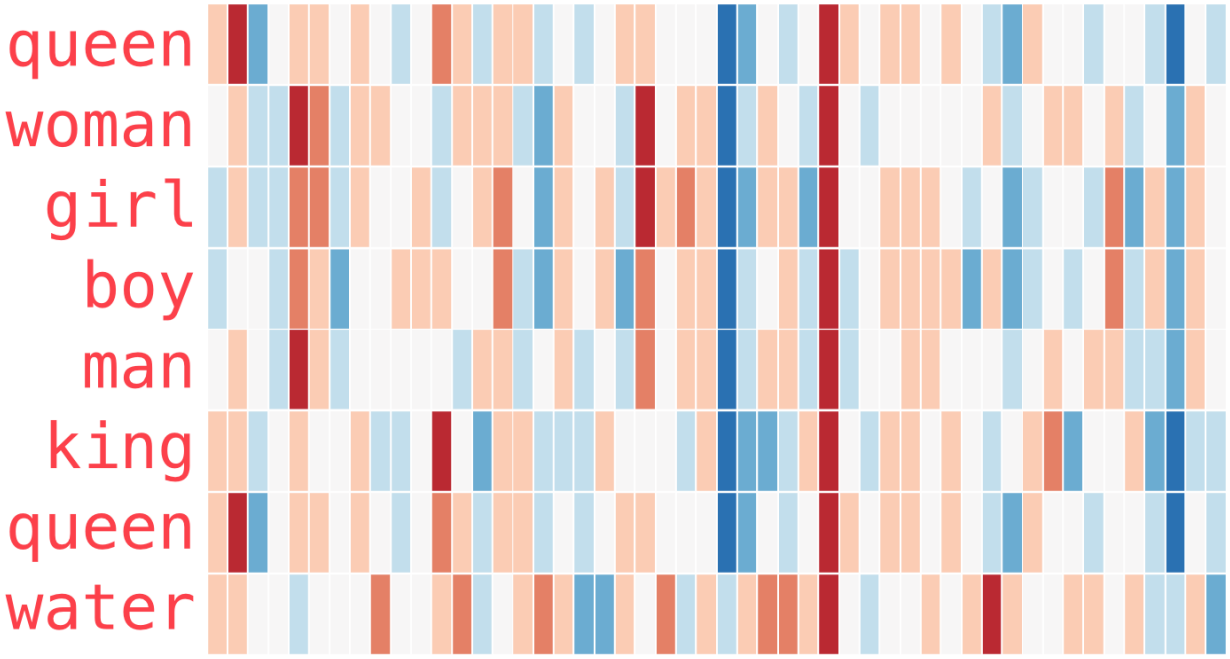
This works great in theory - except we're going to end up with 10,000s of outputs for 10,000s of inputs.

Instead of predicting the *word*, let's just predict yes/no "is this word a neighbour?".



[1] https://jalammar.github.io/illustrated-word2vec/

We also need to create negative samples - and to do that, we pair our word ("make") with randomly generated words from our vocabulary.



[1] https://jalammar.github.io/illustrated-word2vec/

The result is fantastically useful - instead of 10,000 features (0/1 for each token), we now have an *embedding*, which is a denser representation of the words along dimensions related to their use in sentences.

Download the word vectors

In [61]:
```
# 47.1 MB.
!python -m spacy download en_core_web_md
```

```
Collecting en-core-web-md==3.0.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_cor
e_web_md-3.0.0/en_core_web_md-3.0.0-py3-none-any.whl (47.1 MB)
     |████████████████████████████████| 47.1 MB 13.3 MB/s eta 0:00:01
Requirement already satisfied: spacy<3.1.0,>=3.0.0 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from en-core-web-md==3.0.0)
(3.0.3)
Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in /Users/alistairewj/minico
nda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-
core-web-md==3.0.0) (0.8.2)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /Users/alistairewj/min
iconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->
en-core-web-md==3.0.0) (2.25.1)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /Users/alistairewj/m
iniconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0
->en-core-web-md==3.0.0) (1.0.5)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-c
ore-web-md==3.0.0) (4.59.0)
Requirement already satisfied: typer<0.4.0,>=0.3.0 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-c
ore-web-md==3.0.0) (0.3.2)
Requirement already satisfied: catalogue<2.1.0,>=2.0.1 in /Users/alistairewj/min
iconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->
en-core-web-md==3.0.0) (2.0.1)
Requirement already satisfied: packaging>=20.0 in /Users/alistairewj/miniconda3/
```

```
envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-core-
web-md==3.0.0) (20.9)
Requirement already satisfied: pathy in /Users/alistairewj/miniconda3/envs/nlp_l
ecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-core-web-md==3.
0.0) (0.4.0)
Requirement already satisfied: thinc<8.1.0,>=8.0.0 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-c
ore-web-md==3.0.0) (8.0.1)
Requirement already satisfied: pydantic<1.8.0,>=1.7.1 in /Users/alistairewj/mini
conda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->e
n-core-web-md==3.0.0) (1.7.3)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in /Users/alistairewj/minicond
a3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-co
re-web-md==3.0.0) (0.7.4)
Requirement already satisfied: jinja2 in /Users/alistairewj/miniconda3/envs/nlp_
lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-core-web-md==
3.0.0) (2.11.3)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.0 in /Users/alistairewj/
miniconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.
0->en-core-web-md==3.0.0) (3.0.1)
Requirement already satisfied: setuptools in /Users/alistairewj/miniconda3/envs/
nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-core-web-m
d==3.0.0) (49.6.0.post20210108)
Requirement already satisfied: importlib-metadata>=0.20 in /Users/alistairewj/mi
niconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0-
>en-core-web-md==3.0.0) (3.7.2)
Requirement already satisfied: numpy>=1.15.0 in /Users/alistairewj/miniconda3/en
vs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-core-we
b-md==3.0.0) (1.20.1)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-c
ore-web-md==3.0.0) (2.0.5)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /Users/alistairewj/minic
onda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en
-core-web-md==3.0.0) (3.0.5)
Requirement already satisfied: typing-extensions>=3.7.4 in /Users/alistairewj/mi
niconda3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0-
>en-core-web-md==3.0.0) (3.7.4.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.0 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from spacy<3.1.0,>=3.0.0->en-c
ore-web-md==3.0.0) (2.4.0)
Requirement already satisfied: zipp>=0.5 in /Users/alistairewj/miniconda3/envs/n
lp_lecture/lib/python3.7/site-packages (from importlib-metadata>=0.20->spacy<3.
1.0,>=3.0.0->en-core-web-md==3.0.0) (3.4.1)
Requirement already satisfied: pyparsing>=2.0.2 in /Users/alistairewj/miniconda
3/envs/nlp_lecture/lib/python3.7/site-packages (from packaging>=20.0->spacy<3.1.
0,>=3.0.0->en-core-web-md==3.0.0) (2.4.7)
Requirement already satisfied: idna<3,>=2.5 in /Users/alistairewj/miniconda3/env
s/nlp_lecture/lib/python3.7/site-packages (from requests<3.0.0,>=2.13.0->spacy<
3.1.0,>=3.0.0->en-core-web-md==3.0.0) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /Users/alistairewj/minicond
a3/envs/nlp_lecture/lib/python3.7/site-packages (from requests<3.0.0,>=2.13.0->s
pacy<3.1.0,>=3.0.0->en-core-web-md==3.0.0) (2020.12.5)
Requirement already satisfied: chardet<5,>=3.0.2 in /Users/alistairewj/miniconda
3/envs/nlp_lecture/lib/python3.7/site-packages (from requests<3.0.0,>=2.13.0->sp
acy<3.1.0,>=3.0.0->en-core-web-md==3.0.0) (4.0.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/alistairewj/minic
onda3/envs/nlp_lecture/lib/python3.7/site-packages (from requests<3.0.0,>=2.13.0
->spacy<3.1.0,>=3.0.0->en-core-web-md==3.0.0) (1.26.3)
Requirement already satisfied: click<7.2.0,>=7.1.1 in /Users/alistairewj/minicon
da3/envs/nlp_lecture/lib/python3.7/site-packages (from typer<0.4.0,>=0.3.0->spac
y<3.1.0,>=3.0.0->en-core-web-md==3.0.0) (7.1.2)
Requirement already satisfied: MarkupSafe>=0.23 in /Users/alistairewj/miniconda
3/envs/nlp_lecture/lib/python3.7/site-packages (from jinja2->spacy<3.1.0,>=3.0.0
->en-core-web-md==3.0.0) (1.1.1)
```

```
Requirement already satisfied: smart-open<4.0.0,>=2.2.0 in /Users/alistairewj/mi
niconda3/envs/nlp_lecture/lib/python3.7/site-packages (from pathy->spacy<3.1.0,>
=3.0.0->en-core-web-md==3.0.0) (2.2.1)
Requirement already satisfied: boto3 in /Users/alistairewj/miniconda3/envs/nlp_l
ecture/lib/python3.7/site-packages (from smart-open<4.0.0,>=2.2.0->pathy->spacy<
3.1.0,>=3.0.0->en-core-web-md==3.0.0) (1.17.22)
Requirement already satisfied: botocore<1.21.0,>=1.20.22 in /Users/alistairewj/m
iniconda3/envs/nlp_lecture/lib/python3.7/site-packages (from boto3->smart-open<
4.0.0,>=2.2.0->pathy->spacy<3.1.0,>=3.0.0->en-core-web-md==3.0.0) (1.20.22)
Requirement already satisfied: s3transfer<0.4.0,>=0.3.0 in /Users/alistairewj/mi
niconda3/envs/nlp_lecture/lib/python3.7/site-packages (from boto3->smart-open<4.
0.0,>=2.2.0->pathy->spacy<3.1.0,>=3.0.0->en-core-web-md==3.0.0) (0.3.4)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /Users/alistairewj/mini
conda3/envs/nlp_lecture/lib/python3.7/site-packages (from boto3->smart-open<4.0.
0,>=2.2.0->pathy->spacy<3.1.0,>=3.0.0->en-core-web-md==3.0.0) (0.10.0)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /Users/alistairew
j/miniconda3/envs/nlp_lecture/lib/python3.7/site-packages (from botocore<1.21.0,
>=1.20.22->boto3->smart-open<4.0.0,>=2.2.0->pathy->spacy<3.1.0,>=3.0.0->en-core-
web-md==3.0.0) (2.8.1)
Requirement already satisfied: six>=1.5 in /Users/alistairewj/miniconda3/envs/nl
p_lecture/lib/python3.7/site-packages (from python-dateutil<3.0.0,>=2.1->botocor
e<1.21.0,>=1.20.22->boto3->smart-open<4.0.0,>=2.2.0->pathy->spacy<3.1.0,>=3.0.0-
>en-core-web-md==3.0.0) (1.15.0)
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_md')
```

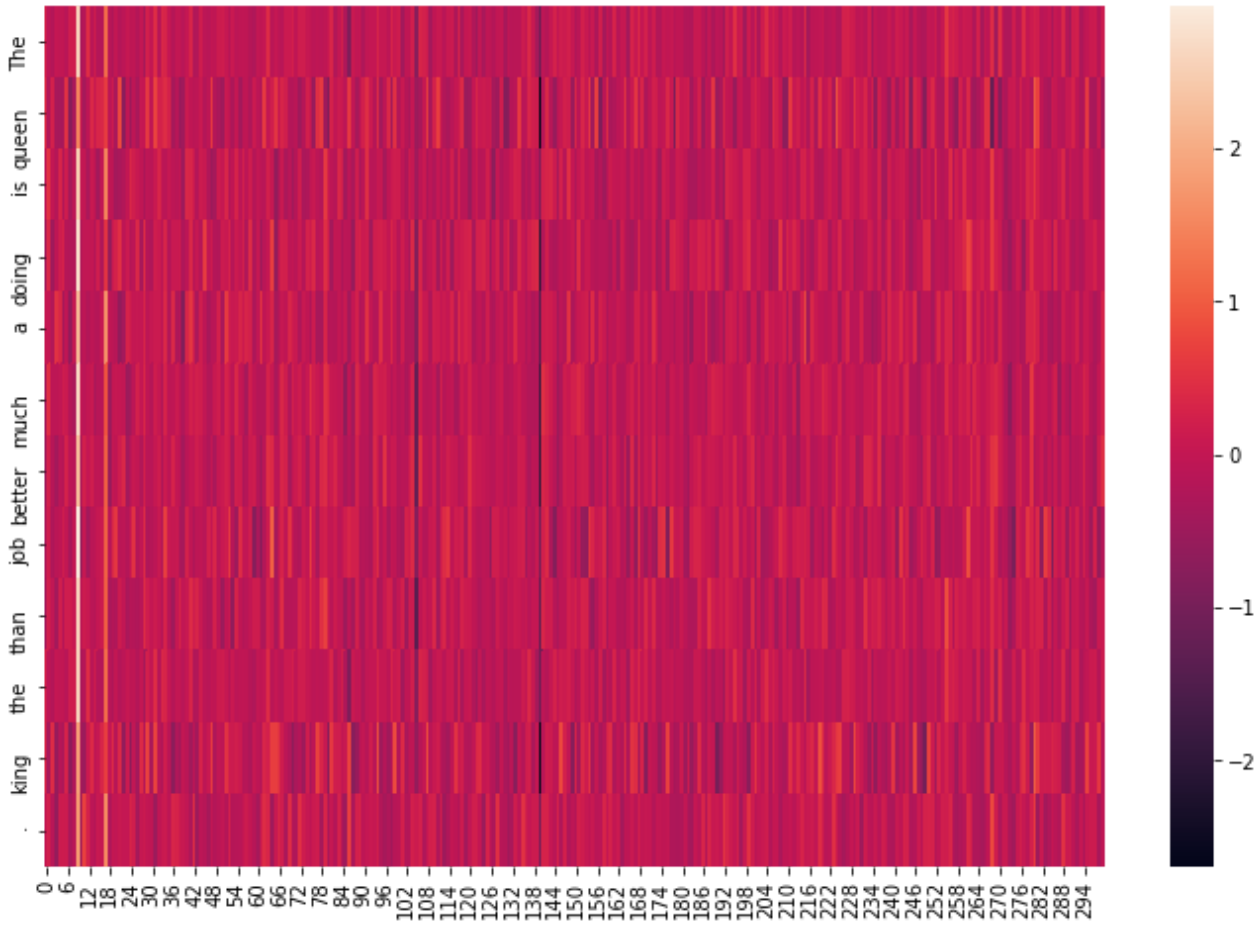
In [62]:

```
# only spacy models medium and larger have word vectors
nlp = spacy.load("en_core_web_md")
text = 'The queen is doing a much better job than the king.'
tokens = nlp(text)

vectors = []

for token in tokens:
    print(f'{token.text:10s}{token.has_vector} {token.vector_norm} {[f"{t:1.2f}"
    vectors.append(token.vector)

vectors = np.vstack(vectors)
```

```
The       True 4.709350109100342 ['0.27', '-0.06', '-0.19', '0.02', '-0.02'] ...
queen     True 6.829740524291992 ['0.41', '-0.23', '0.25', '-0.36', '-0.37'] ...
is        True 4.890305995941162 ['-0.08', '0.50', '0.00', '-0.17', '0.31'] ...
doing     True 5.571891784667969 ['-0.43', '0.36', '-0.45', '-0.07', '0.18'] ...
a         True 5.306695938110352 ['0.04', '0.02', '-0.21', '0.50', '0.36'] ...
much      True 5.0841450691223145 ['-0.41', '0.47', '-0.07', '-0.07', '-0.02']
...
better    True 5.107147693634033 ['-0.46', '0.21', '-0.15', '-0.40', '-0.10']
...
job       True 6.286815166473389 ['-0.29', '0.15', '-0.25', '-0.12', '-0.04']
...
than      True 5.180047988891602 ['-0.40', '0.19', '-0.02', '-0.40', '0.19'] ...
the       True 4.709350109100342 ['0.27', '-0.06', '-0.19', '0.02', '-0.02'] ...
king      True 7.141745567321777 ['0.32', '-0.35', '0.43', '-0.54', '-0.18'] ...
.         True 4.93163537979126 ['0.01', '0.21', '-0.13', '-0.59', '0.13'] ...
```

In [63]:

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=[12, 8])
sns.heatmap(vectors, yticklabels=[token.text for token in tokens])
plt.show()
```



Further reading on word vectors

- https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/
- https://jalammar.github.io/illustrated-word2vec/

What kind of tasks do people work on?

- Text classification
- Named Entity Recognition
- Part-of-speech tagging
- Question answering
- Entity Linking
- Relation extraction
- Topic modeling
- Language modeling
- Translation - human translation is a 33 billion dollar industry

Benchmarks

- GLUE
- SuperGLUE

What libraries do people use?

- scikit-learn
- nltk
- spaCy
- fastai
- PyTorch / HuggingFace

Fun datasets to try

- ICLR 2021 Abstracts with Scores
- IMDb dataset
- Jeopardy! questions
- List of datasets

Further reading

- http://web.stanford.edu/class/cs224n/
- https://github.com/fastai/course-nlp
- http://norvig.com/ngrams/
- The shallowness of Google translate - we still have some way to go!

Thanks!