Applied Natural Language Processing

Dr Jeff Mitchell, University of Sussex

Autumn 2025

Where are we?

Last time

- Pre-processing text documents
  - segmentation
  - tokenization
  - Zipf's Law
  - normalization
  - punctuation and stopword removal
  - stemming and lemmatization

This time

- Document classification
  - Feature extraction
  - Word list classifiers
- Evaluation
  - accuracy and error rate
  - the confusion matrix
  - precision, recall and F1-score

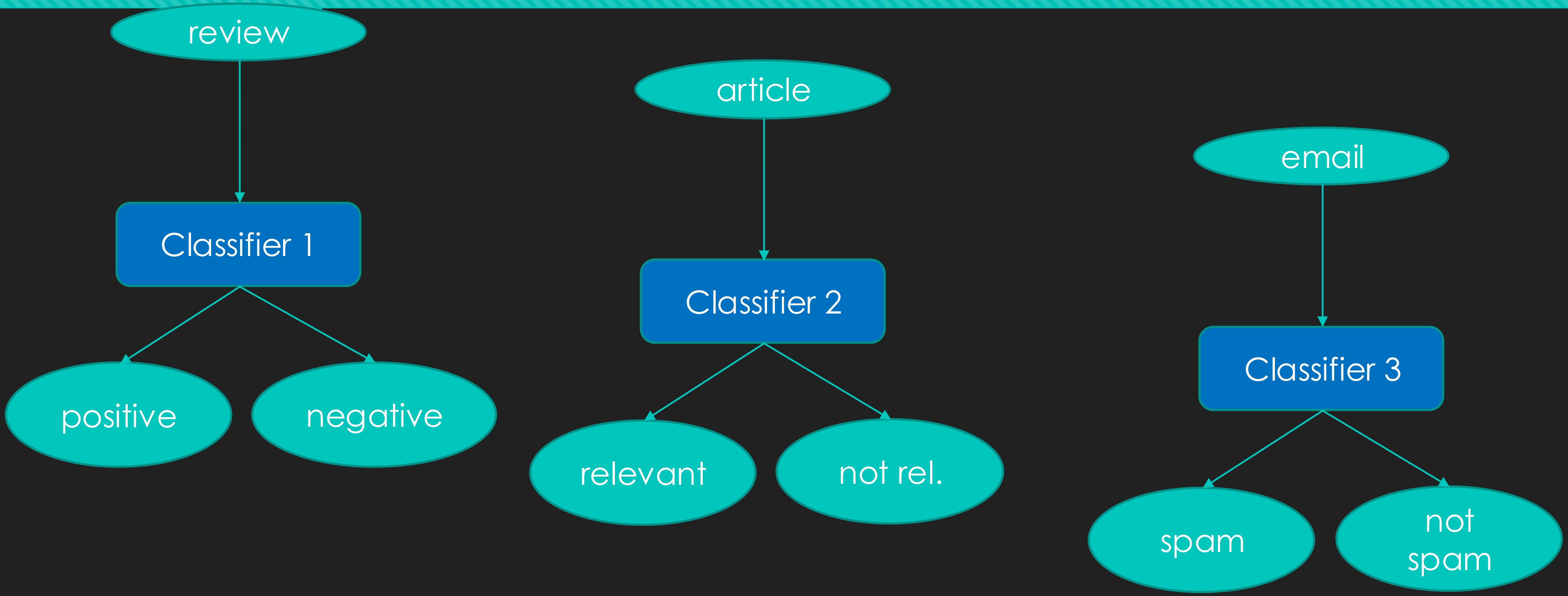
Introduction to the document classification scenario

Part 1

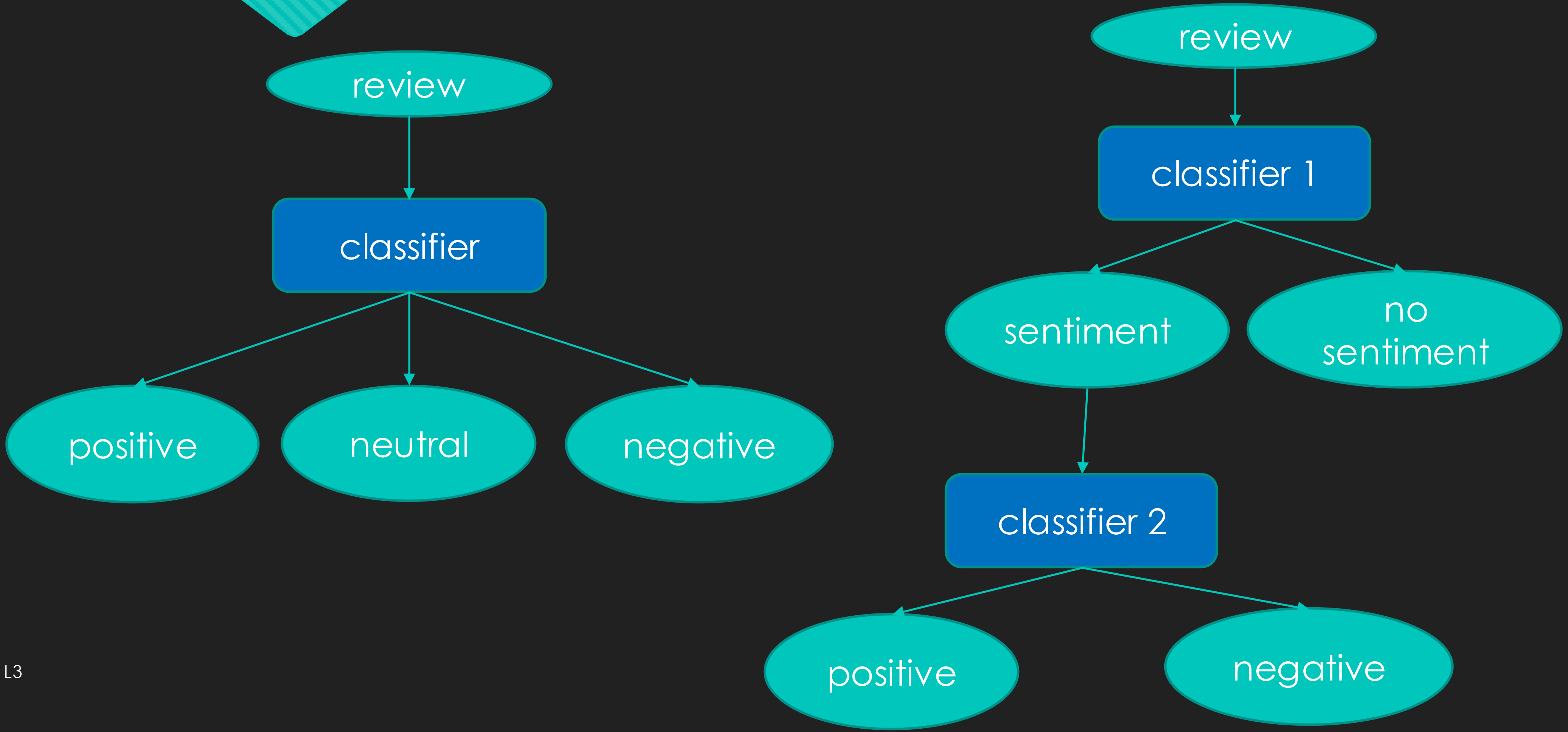
What do the following scenarios have in common?

- Given a corpus of reviews of a new restaurant, determining how many are positive and how many are negative about the restaurant.
- Given a corpus of recently published scientific articles, identifying all of the ones which are about Natural Language Engineering.
- Identifying and filtering spam emails.
- Identifying and blocking posts to a social media website which contain swear words.
- Determining whether tweeters are for or against a particular political campaign (or politician).

Binary classification



Multi-class classification



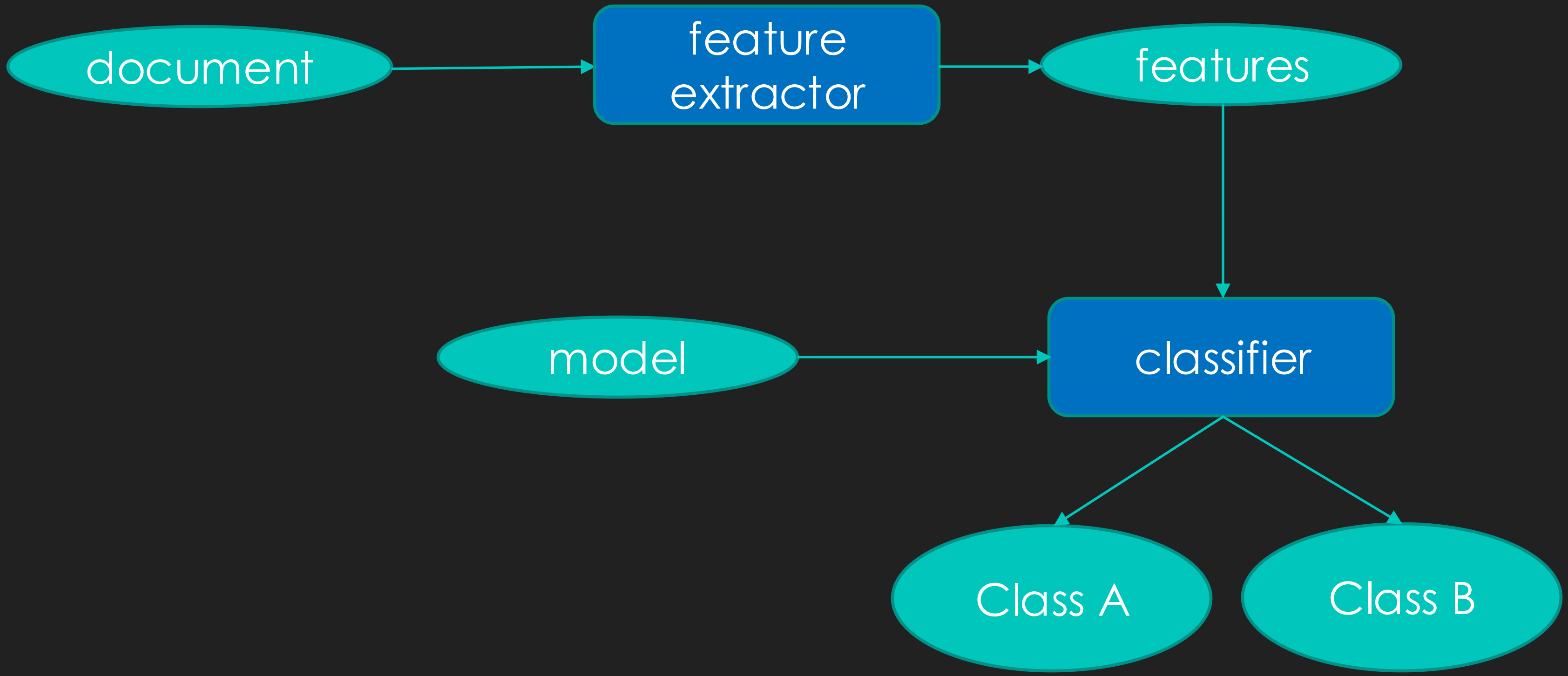
General issues

- Specifying what makes a document in one class or another
  - Can we *learn* from positive and negative examples?
- Adapting to dynamics of data source
  - Class specifications may change over time
  - e.g., words particularly indicative of positive or negative sentiment in a particular domain
- Unbalanced classes
  - One class may be much larger than the other (e.g., many more irrelevant documents than relevant documents)
- Is one decision per document appropriate?

Beyond classification

- Identification of parts of a document which are relevant or where sentiment is expressed
- Identification of specific entities within a document that sentiment (or other classification) may be applied to:
  - Different films referred to in a review of latest film releases
  - Comparisons with other products/entities:
    - "The One Max is disappointing compared to the excellent iPhone 5"
- Identification of specific features of entities that sentiment (or other classification) may be applied to:
  - "I don't like iPads but the battery life is particularly impressive"

General Architecture for Document Classification



1. Feature extractor turns document into a set or **vector** of features

2. Classifier consults a model of what features to expect in different classes and decides the **most likely** class accordingly

Questions

- What are the features that we need to extract from documents in order to be able to classify them?
- Where does the model come from?

Feature extraction

Words as features

- Words within documents provide excellent evidence of being in a particular class
  - ***excellent*** – evidence that a review is positive
  - ***bitcoin*** – evidence that an email is spam
  - ***lemma*** – evidence that an article is relevant to NLE
- **KEY IDEA**: Just treat a document as a **bag-of-words.**
  - Ignore order and grammatical structure
  - Preprocessing critical – what constitutes a word?
  - Sometimes ignore frequency too – although then strictly it would be a set-of-words

Bag-of-words representation

The cats chase the mice.

The mice chase the cats.

Feature representation

- May be binary-valued:
  - D1 = {**cats**:*True*, **mice**: *True*, **chase**: *True*, **the**: *True*}
- Or numeric-valued:
  - D1 = {**cats**:*1*, **mice**: *1*, **chase**: *1*, **the**: *2*}
- Might be stored with sparse vector representation such as python dictionaries (as above)
  - default value for keys not present is False / zero

```
value = D1.get(word,0)
```

Vectors and matrices

- Document features may be stored in fixed length arrays / matrices

	a	bat	car	cat	…	mice	…	yelp	zebra
D1	0	0	0	1	…	1	…	0	0
D2	1	1	0	0	…	0	…	0	0
D3	1	0	1	1	…	0	…	1	1

- Length of each row vector = |V|
- Inherent sparsity (lots of zeros)
- Python dictionaries with defaults are much more space efficient data structure

Classification models

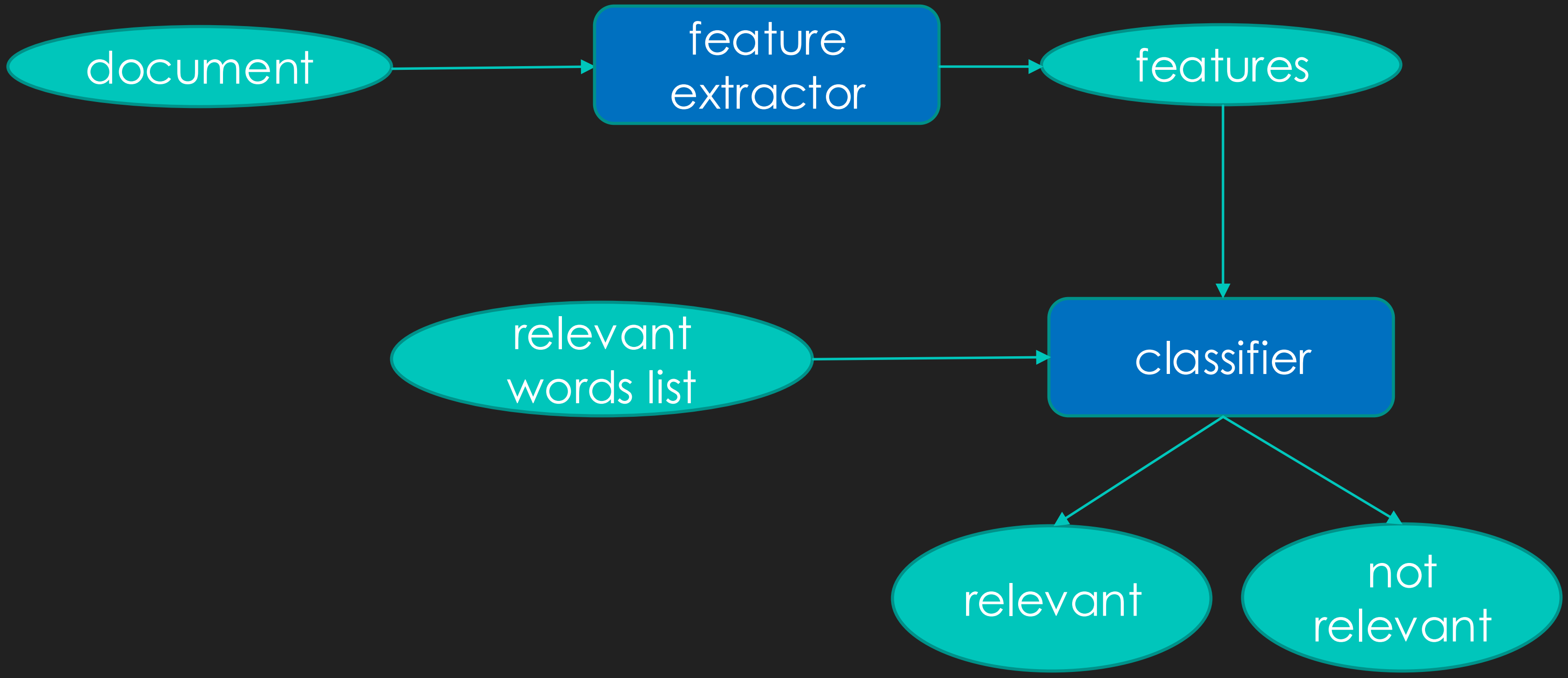
Models for document classification

Approaches we will consider:

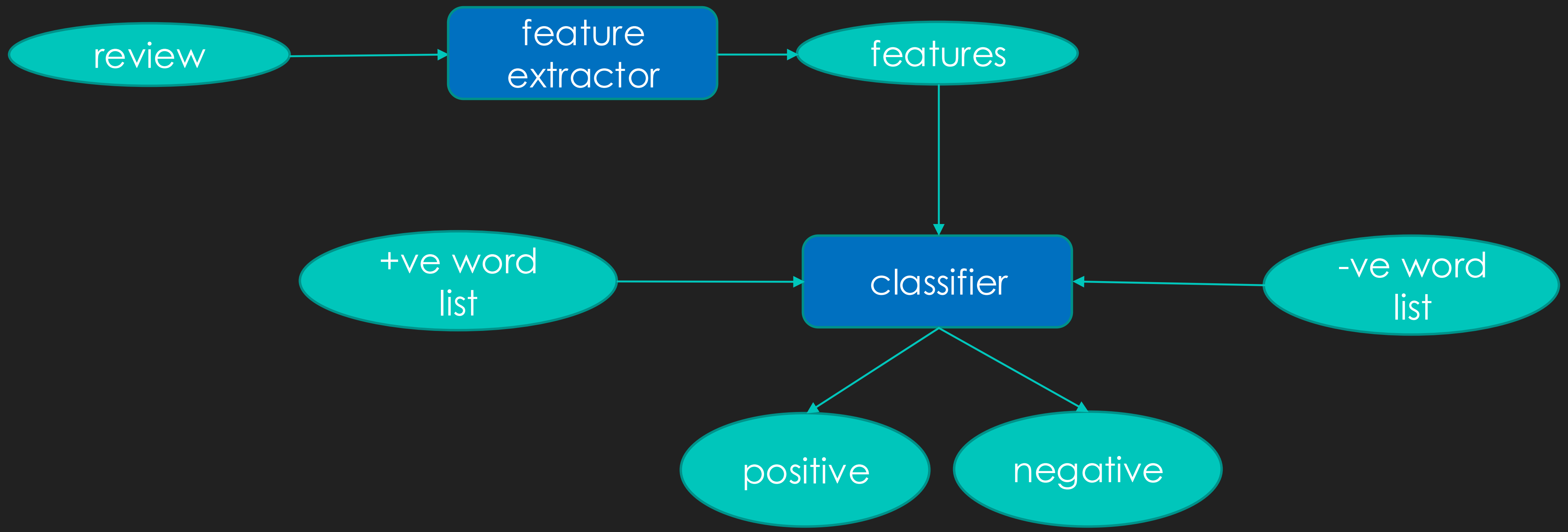
- Hand-crafted vocabulary lists
- Automatically derived vocabulary lists
- Naïve Bayes

Word list classifiers

Using vocabulary lists for relevancy classification



Using vocabulary lists for sentiment classification



Stop and think

- What words might you expect to occur in a positive review about a movie?
- What words might you expect to occur in a negative review about a movie?

Scoring documents

- A document is treated as a **bag-of-words**, *d*
- Number of occurrences of word *w* in *d* is denoted:

$$count(d, w)$$

- Generalise to total occurrences of words from list *L* in *d*:

$$count(d, L) = \sum_{w \in L} count(d, w)$$

Example: Scoring documents for sentiment analysis

- Positive sentiment word list: L_+
- Negative sentiment word list: L_-
- Class decision for document, *d,* is given by:

$$class(d) = \begin{cases} positive & \text{if } count(d, L_+) > count(d, L_-) \\ negative & \text{if } count(d, L_-) > count(d, L_+) \end{cases}$$

Options

- Different ways of scoring words occurring in documents.

1. Frequency (seen in example)
2. Uniform score for occurrence.
3. Weighted by 'importance' of word.
   - But where do these weights come from?

- Differences in linguistic processing.
  - Should you use:
    - Case normalization?
    - Number normalization?
    - Lemmatisation?

$$score(d,w) = count(d,w)$$

$$score(d,w) = \begin{cases} 1 & \text{if } w \text{ occurs in } d \\ 0 & \text{otherwise} \end{cases}$$

$$score(d,w) = \begin{cases} imp(w) & \text{if } w \text{ occurs in } d \\ 0 & \text{otherwise} \end{cases}$$

$$score(d,w) = imp(w) \times count(d,w)$$

Problems with hand-crafted lists

- Degree of variation in the way people express themselves
- Hard to build comprehensive lists
- Hard to build balanced lists
- Variation in vocabulary across different domains
- Relevance of words not always obvious

Automatic derivation of wordlists

- Use a sample of **labelled** documents.
  - Need to ensure that both (all) classes are represented in sample
  - Class **imbalance** can be a big problem
- Partition this into **training** and **testing** sets
  - Common splits are 50:50, 80:20 and 90:10
- Use training sample of documents to build a model
  - i.e., select words for word lists
- Use testing sample of documents to find out how well the model works
- Proper use of training and testing data is essential to avoid **over-fitting** the data
  - Want to know how well the model works on unseen data NOT how well it works on data that has already been labelled

supervised learning

Supervised learning is great but …

- Relies on labelled data
- This can be expensive to produce
  - Generally requires humans to hand-label it
- The more complex the model being built, the more labelled training data it will require
- Models in NLE are usually highly domain dependent
- **Domain adaptation** is a big challenge

Selection of words for word lists

- **Most frequent terms**
  - Positive word list – most frequently occurring words in positive documents
  - Negative word list – most frequently occurring words in negative documents
- **Greatest frequency difference**
  - A word's score for class c is its total frequency in documents labelled as c minus total frequency in documents labelled as not c.
  - Word list for class c is the words with highest scores for class c

Tuning hyperparameters

- Length of vocabulary lists is an example of a **hyperparameter**
- Tempting to try lots of different possibilities until one gives good results (on the testing data)
  - May do well just because a certain word in test data was or wasn't included on the list
  - May not generalize to truly unseen data
  - This is **over-fitting**

- Empirically determine the optimal value of hyperparameters on held-out *development* set
  - Requires partitioning labelled data into **training, testing** and **development** sets
  - Learn more about other methods for hyperparameter tuning in the *Machine Learning* module

Evaluating classifiers

Part 2

Evaluating classifiers

- Why?
  - We want to know how well our classifier will do on unseen documents
  - We want to be able to choose the best classifier (or parameter settings) in a particular scenario
- Need a labelled dataset that has **not been used in training**
  - Evaluate on a held-out **testing** set
  - Good practice to compare different parameter settings on a separate held-out **development** set, before calculating final performance on the testing set.
- Here we focus on binary classification
  - Common scenario
  - May further distinguish one class (e.g., “positive”) as being the one of interest
  - Metrics considered do generalize to multi-class cases.

Accuracy and error rate

- Accuracy is the proportion of items in the test set that are classified correctly.

$$accuracy = \frac{|\{i|prediction(i) = label(i)\}|}{|\{i\}|}$$

- Error rate is the proportion of items in the test set that are classified incorrectly.

$$error\ rate = \frac{|\{i|prediction(i) \neq label(i)\}|}{|\{i\}|}$$

- Note that: $error\ rate = 1 - accuracy$

Stop and think

- Are accuracy and error rate the most appropriate way of assessing/comparing classifiers?
- Can you think of a scenario where using accuracy or error rate might be misleading?

Confusion Matrix

	Predicted Class		
True Class		+ve	-ve
	+ve	True Positives (**TP**)	False Negatives (**FN**)
	-ve	False Positives (**FP**)	True Negatives (**TN**)

Total number of data items:

$$N = TP + FP + TN + FN$$

Accuracy: the proportion classified correctly

$$Accuracy = \frac{TP + TN}{N}$$

Error rate: the proportion classified incorrectly

$$Error\ rate = \frac{FN + FP}{N}$$

Example 1

	Predicted Class			
True Class		+ve	-ve	total
	+ve	**TP=45**	**FN=5**	50
	-ve	**FP=15**	**TN=35**	50
	total	60	40	100

$$Accuracy = \frac{45 + 35}{100} = 0.8$$

$$Error\ rate = \frac{5 + 15}{100} = 0.2$$

Stop and think

- Is this an example of a balanced or an unbalanced class distribution?

Example 2

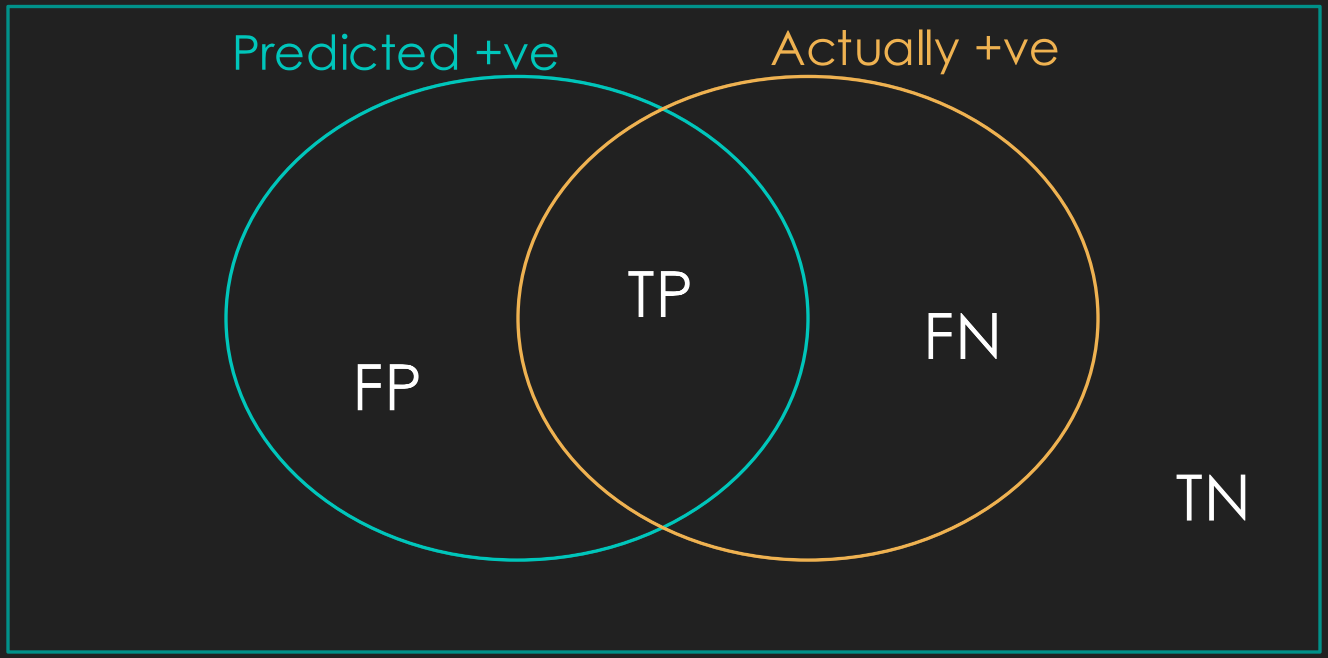
		Predicted Class		
True Class		+ve	-ve	total
	+ve	**TP=3**	**FN=7**	10
	-ve	**FP=3**	**TN=87**	90
	total	6	94	100

$$Accuracy = \frac{3+87}{100} = 0.9$$

$$Error\ rate = \frac{7+3}{100} = 0.1$$

But how good is this classifier at predicting the positive class?

Precision and recall



Recall is the proportion of actually +ve documents that are predicted correctly

$$Recall = \frac{TP}{TP + FN}$$

Precision is the proportion of +ve predictions that are correct

$$Precision = \frac{TP}{TP + FP}$$

Example 1 (again)

	Predicted Class		
True Class	+ve	-ve	total
+ve	**TP=45**	**FN=5**	50
-ve	**FP=15**	**TN=35**	50
total	60	40	100

For the positive class:

Recall= $\frac{TP}{TP+FN}$

Recall $= \frac{45}{45+5} = 0.9$

Precision= $\frac{TP}{TP+FP}$

Precision= $\frac{45}{45+15} = 0.75$

Example 2 (Again)

	Predicted Class			
True Class		+ve	-ve	total
	+ve	**TP=3**	**FN=7**	10
	-ve	**FP=3**	**TN=87**	90
	total	6	94	100

For the positive class:

Recall= $\frac{TP}{TP+FN}$

Recall= $-$

Precision= $\frac{TP}{TP+FP}$

Precision= $-$

F1-Score

- In general, we want high precision AND high recall
- We combine precision (*P*) and recall (*R*) scores using the F1-score:

$$F1 = \frac{2PR}{P + R}$$

- This is the harmonic mean (and is always closer to the lower of two values)

In example 1, *R*= 0.9 and *P*=0.75

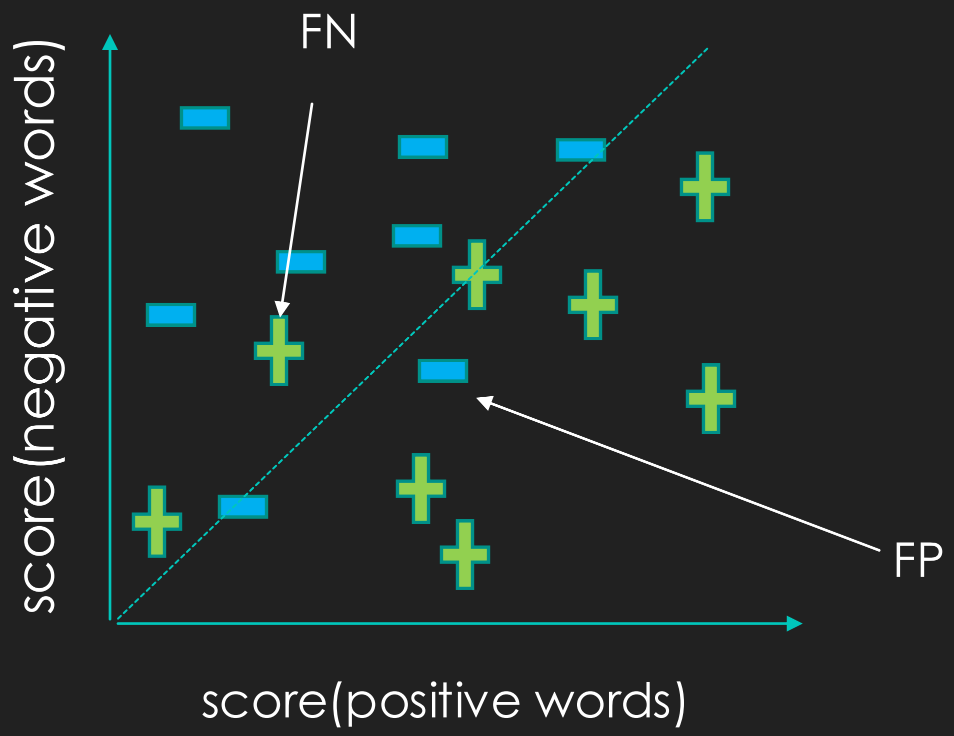
$$F1 = \frac{2 \times 0.9 \times 0.75}{0.9 + 0.75} = 0.818$$

In example 2 *R*= 0.3 and *P*=0.5

$$F1 = \frac{2 \times 0.3 \times 0.5}{0.3 + 0.5} = 0.375$$

Trading Off Precision and Recall

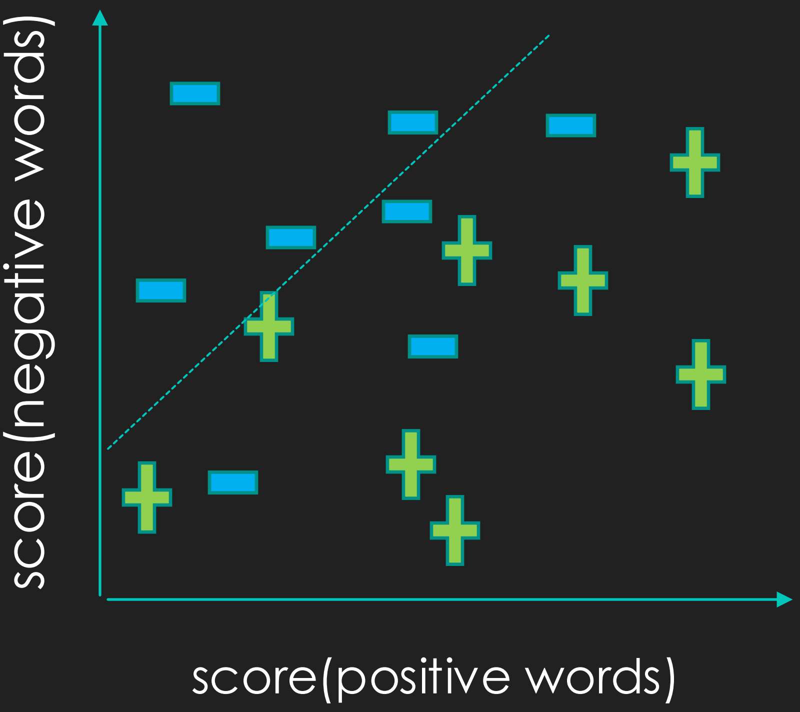
For any given classifier, we can change the **decision boundary**, making it more or less likely to classify an item as positive.



- The standard decision rule for a wordlist classifier is **score(positive words) > score(negative words)**
- Everything below the boundary of the graph is classified as positive, everything above is classified as negative
- If the true labels are as shown, we get some FN and some FP
- Affecting precision and recall

Trading Off Precision and Recall

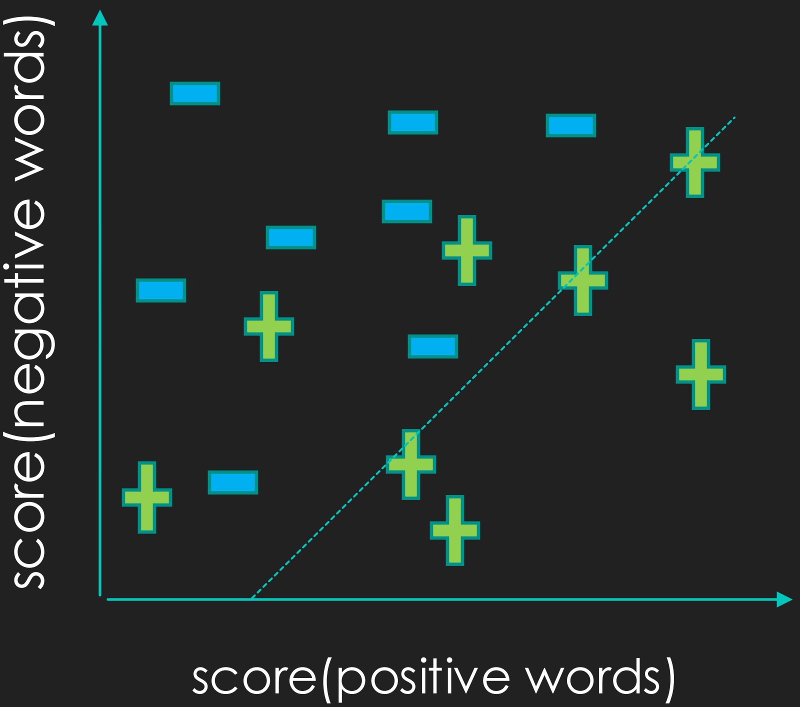
- For any given classifier, we can change the **decision boundary**, making it more or less likely to classify an item as positive.



- Moving the boundary to the left reduces the number of FN
- Recall → 1

Trading Off Precision and Recall

- For any given classifier, we can change the **decision boundary**, making it more or less likely to classify an item as positive.



- Moving the boundary to the right reduces the number of FP
- Precision → 1

Making progress

- This week you should complete **all** of the exercises in **all 2 notebooks** for week 3 on Document Classification:

  - Part 1: Lab_3_1.ipynb
  - Part 2: Lab_3_2.ipynb

Keywords Check

binary classification	
bag-of-words	
supervised learning	
over-fitting	
hyperparameter	
accuracy	
error rate	
confusion matrix	
precision	
recall	
F1 Score	

More Python

Classes in Python

- Classes in python support object-oriented programming paradigm
- Intuitively, classes can be thought of as a user-defined type
- The class defines the attributes that each object needs to have and the methods which can be called on it
- For example, a PERSON class might have
  - attributes such as name, age and birthplace
  - methods such as is_eligible()
- You instantiate or construct an object belonging to a particular class and call methods on particular objects

PERSON example

```python
class Person():
    def __init__(self,name,age,birthplace):
        self.name=name
        self.age=age
        self.birthplace=birthplace

    def is_eligible(self,):
        return self.age>17 and self.birthplace.lower()=="brighton"
```

```python
people=[]
person1=Person("john",12,"brighton")
people.append(person1)
person2=Person("diana",19,"brighton")
people.append(person2)
person3=Person("bob",20,"london")
people.append(person3)
```

```python
for p in people:
    if p.is_eligible():
        print(p.name)
```

```
diana
```

- Note the use of 'self' to refer to this particular instance of a class
- Note the use of __init__() which is automatically called when an instance is created

Inheritance

- Note that the Student class **inherits** from the Person class
- All attributes and methods in Person are available in Student UNLESS they are explicitly **over-ridden** (like __init__)

```
class Student(Person):
    def __init__(self,name,age,birthplace):
        self.name=name
        self.age=age
        self.birthplace=birthplace
        self.scores=[]

    def add_score(self,score):
        self.scores.append(score)

    def average_score(self):
        if len(self.scores)>0:
            return sum(self.scores)/len(self.scores)
        else:
            return 0
```

```
person1=Student("diana",19,"brighton")
person1.add_score(45)
person1.add_score(60)
print("The average score for {} is {}".format(person1.name,person1.average_score()))
print("Eligibility of {}: {}".format(person1.name,person1.is_eligible()))
```

```
The average score for diana is 52.5
Eligibility of diana: True
```