CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computer Science

http://www.cit.ie

Natural Language Processing

Week2: Text Processing with Regular Expressions, *Unix tools and* Scikit-Learn

Dr. Haithem Afli

Haithem.afli@cit.ie

@AfliHaithem

2020/2021

Recap

1) Natural language understanding, is it still challenging?

2) Describe NLP technologies with three words!

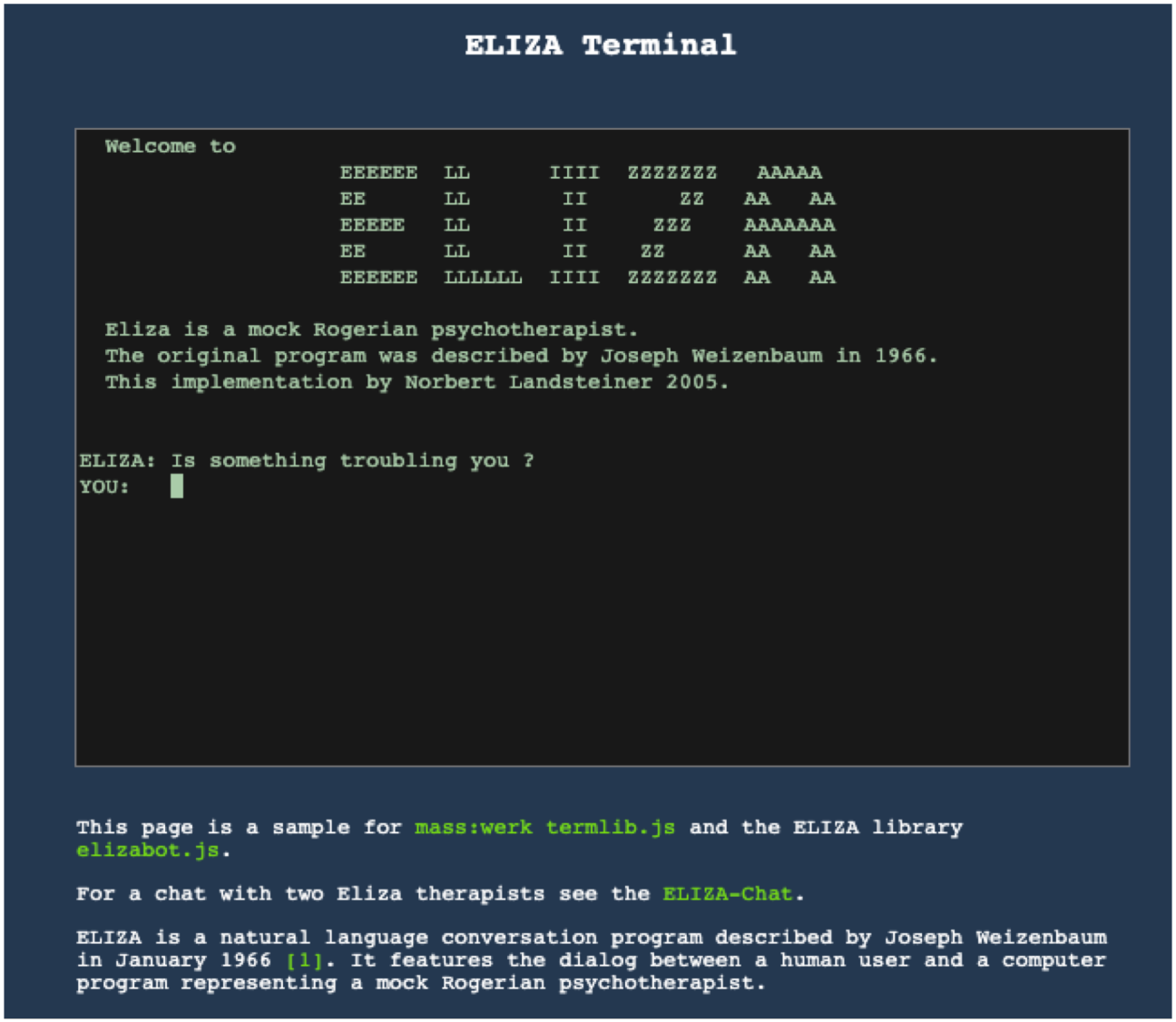
Go to https://www.menti.com/wxhysyji96 the digit code **46 08 69 4**

Or use the following QR Code

Outline

1. **Regular Expressions**
2. *Unix tools*
3. *Tokenization*
4. *Text processing with* Scikit-Learn

Please tell me
what's been bothering you.



Eliza

- Eliza is a mock Rogerian psychotherapist.
  - Rogerian psychotherapy, or person-centered therapy, is a form of talk-psychotherapy developed by psychologist Carl Rogers in the 1940s & 1950s.
- The original program was described by Joseph Weizenbaum in 1966.
  - Primitive NLP, using simple scripts and **regular expressions**
  - http://www.masswerk.at/elizabot/eliza.html
- Chatterbots or chatbots
- Eliza effect
- Turing test
- Regular expressions

Please tell me what's been bothering you.

```
  Welcome to
                    EEEEEE  LL      IIII  ZZZZZZZ   AAAAA
                    EE      LL       II       ZZ   AA   AA
                    EEEEE   LL       II     ZZZ    AAAAAAA
                    EE      LL       II    ZZ      AA   AA
                    EEEEEE  LLLLLL  IIII  ZZZZZZZ  AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Please tell me what's been bothering you.
YOU:   My boyfriend made me come here.
ELIZA: Your boyfriend made you come here ?
YOU:   I am really unhappy.
ELIZA: Do you think coming here will help you not to be unhappy ?
YOU:   I hope it will.
ELIZA: Do you say you hope it will for some special reason ?
YOU:
```

Regular Expressions

- In computer science, RE (or regex, regexp) is a language used for **specifying text search string**.
- A *regular expression* is a ***formula*** **in a** ***special language*** that is used for specifying a simple class of *string*.
- Formally, a regular expression is an algebraic notation for characterizing a set of strings.
- RE search requires
  - a *pattern* that we want to search for, and
  - a *corpus* of texts to search through.
- http://www.regexr.com

Regular Expressions

- A RE search function will search through the corpus returning all texts that contain the pattern.
  - In a Web search engine, they might be the entire documents or Web pages.
  - In a word-processor, they might be individual words, or lines of a document. (We take this paradigm.)
    - E.g., the UNIX `grep` command

Regular Expressions
Basic Regular Expression Patterns

- The use of the brackets `[]` to specify a disjunction of characters.

RE	Match	Example Patterns
`/[wW]oodchuck/`	Woodchuck or woodchuck	"Woodchuck"
`/[abc]/`	'a', 'b', *or* 'c'	"In uomini, in soldati"
`/[1234567890]/`	any digit	"plenty of 7 to 5"

- The use of the brackets `[]` plus the dash − to specify a range.

RE	Match	Example Patterns Matched
`/[A-Z]/`	an uppercase letter	"we should call it 'Drenched Blossoms'"
`/[a-z]/`	a lowercase letter	"my beans were impatient to be hoed!"
`/[0-9]/`	a single digit	"Chapter 1: Down the Rabbit Hole"

Regular Expressions
Basic Regular Expression Patterns

- Uses of the caret ^ for negation or just to mean ^

RE	Match (single characters)	Example Patterns Matched
[^A-Z]	not an uppercase letter	"Oyfn pripetchik"
[^Ss]	neither 'S' nor 's'	"I have no exquisite reason for't"
[^\.]	not a period	"our resident Djinn"
[e^]	either 'e' or '^'	"look up ^ now"
a^b	the pattern 'a^b'	"look up a^ b now"

- The question-mark ? marks optionality of the previous expression.

RE	Match	Example Patterns Matched
woodchucks?	woodchuck or woodchucks	"woodchuck"
colou?r	color or colour	"colour"

- The use of period . to specify any character

RE	Match	Example Patterns
/beg.n/	any character between *beg* and *n*	begin, beg'n, begun

Regular Expressions: Anchors ^ $

Pattern	Matches
^[A-Z] Starting with uppercase letter	Palo Alto
^[^A-Za-z]	1 "Hello"
\.$	The end.
.$	The end? The end!

Regular Expressions
Disjunction, Grouping, and Precedence

- Disjunction

```
/cat|dog/
```

- Precedence

```
/gupp(y|ies)/
```

- Operator precedence hierarchy

```
()
+ ? { }
the   ^my   end$
|
```

Regular Expressions
A Simple Example

- To find the English article ***the***

```
/the/

/[tT]he/

/\b[tT]he\b/

/[^a-zA-Z][tT]he[^a-zA-Z]/

/(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/
```

Errors

- The process we just went through was based on fixing two kinds of errors
  - Matching strings that we should not have matched (there, then, other)
    - False positives (Type I)
  - Not matching things that we should have matched (The)
    - False negatives (Type II)

Errors cont.

- In NLP we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy or precision (minimizing false positives)
  - Increasing coverage or recall (minimizing false negatives).

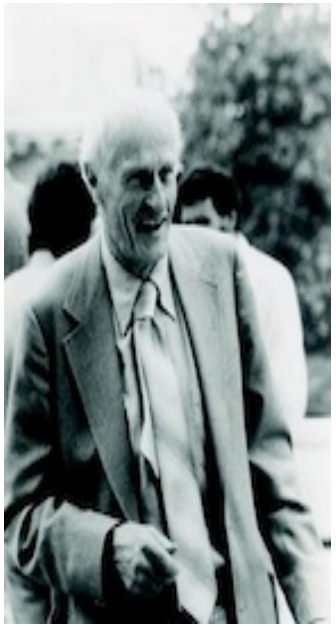
Regular Expressions
Advanced Operators

Aliases for common sets of characters

RE	Expansion	Match	Example Patterns
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9␣]	any alphanumeric or space	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

Regular Expressions: ? * + .

Pattern	Matches	
`colou?r`	Optional previous char	color colour
`oo*h!`	0 or more of previous char	oh! ooh! oooh! ooooh!
`o+h!`	1 or more of previous char	oh! ooh! oooh! ooooh!
`baa+`		baa baaa baaaa baaaaa
`beg.n`	any char	begin begun begun beg3n



Stephen C Kleene

Kleene *, Kleene +

Regular Expressions
Advanced Operators

Regular expression operators for counting

RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	*n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous char or expression
{n,}	at least *n* occurrences of the previous char or expression

Regular Expressions
Advanced Operators

Some characters that need to be backslashed

RE	Match	Example Patterns Matched
*	an asterisk "*"	"K*A*P*L*A*N"
\.	a period "."	"Dr. Livingston, I presume"
\?	a question mark	"Would you light my candle?"
\n	a newline	
\t	a tab	

```
s/regexp1/regexp2/
```

- E.g. *the 35 boxes* → *the <35> boxes*

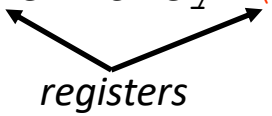
```
s/([0-9]+)/<\1>/
```

- The following pattern matches "*The bigger they were, the bigger they will be*", not *"The bigger they were, the faster they will be"*

```
/the (.*)er they were, the\1er they will be/
```

- The following pattern matches "*The bigger they were, the bigger they were*", not *"The bigger they were, the bigger they will be"*

```
/the (.*)er they (.*), the\1er they \2/
```



Regular Expressions
Regular Expressions Substitution, Memory, and ELIZA

- Eliza worked by having a cascade of regular expression substitutions that each match some part of the input lines and changed them
  - *my → YOUR, I'm → YOU ARE …*

```
s/.* YOU ARE (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* YOU ARE (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

User$_1$: Men are all alike.
ELIZA$_1$: IN WHAT WAY
User$_2$: They're always bugging us about something or other.
ELIZA$_2$: CAN YOU THINK OF A SPECIFIC EXAMPLE
User$_3$: Well, my boyfriend made me come here.
ELIZA$_3$: YOUR BOYBRIEND MADE YOU COME HERE
User$_4$: He says I'm depressed much of the time.
ELIZA$_4$: I AM SORRY TO HEAR YOU ARE DEPRESSED

Discussion

Outline

1. Regular Expressions
2. ***Unix tools***
3. *Tokenization*
4. *Text processing with* Scikit-Learn

Unix Tools

- **grep**: search for a pattern (regex)
- **sort**
- **uniq –c** (count duplicates)
- **tr** (translate characters)
- **wc** (word – or line – count)
- **sed** (edit string -- replacement)
- **cat** (send file(s) in stream)
- **man** (manual)
- **cut** (columns in tab-separated files)
- **paste** (paste columns)
- **head** (top of the file)
- **tail** (bottom of the file)
- **rev** (reverse lines)
- **comm** (compare)
- **shuf** (shuffle lines of text)
- Input/output redirection:
  - `>, <, |`

- *Piping commands together can be simple yet powerful in Unix*
- *It gives flexibility.*
- *Traditional Unix philosophy: small tools that can be composed*

Simple Tokenization in UNIX

- (Inspired by Ken Church's UNIX for Poets.)
- Given a text file, output the word tokens and their frequencies

```
tr -sc 'A-Za-z' '\n' < shakes.txt
      | sort
      | uniq –c
```

Change all non-alpha to newlines

Sort in alphabetical order

Merge and count each type

```
1945 A                25 Aaron
  72 AARON             6 Abate
  19 ABBESS            1 Abates
   5 ABBOT             5 Abbess
 ... ...               6 Abbey
                       3 Abbot
                      .... …
```

Simple Tokenization in UNIX

- (Inspired by Ken Church's UNIX for Poets.)
- Given a text file, output the word tokens and their frequencies

```
cat shakes.txt
      | tr -sc 'A-Za-z' '\n'
      | sort
      | uniq –c
```

Change all non-alpha to newlines

Sort in alphabetical order

Merge and count each type

```
1945 A                25 Aaron
  72 AARON             6 Abate
  19 ABBESS            1 Abates
   5 ABBOT             5 Abbess
 ... ...               6 Abbey
                       3 Abbot
                     .... …
```

Counting

- Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr –sc 'A-Za-z' '\n' | sort | uniq –c
```

- Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr –sc 'A-Za-z' '\n' | sort | uniq –c | sort –n –r
```

```
23243 the
22225 i
18618 and
16339 to
15687 of
12780 a
12163 you
10839 my
10005 in
8954  d
```

What happened here?

Shakespeare - Sonnet 50

How heavy do I journey on the way,
When what I seek, my weary travel's end,
Doth teach that ease and that repose to say,
'Thus far the miles are measured from thy friend!'
The beast that bears me, tired with my woe,
Plods dully on, to bear that weight in me,
As if by some instinct the wretch did know
His rider **lov'd** not speed being made from thee.

....

Not to mention contractions such as had/would:
he'd said, he'd say…

Some of the output

- `tr -sc 'A-Za-z' '\n' < nyt_200811.txt | sort | uniq -c | head –n 5`

```
 25476 a
  1271 A
     3 AA
     3 AAA
     1 Aalborg
```

- `tr -sc 'A-Za-z' '\n' < nyt_200811.txt | sort | uniq -c | head`
- Gives you the first 10 lines
- tail does the same with the end of the input
- (You can omit the “-n” but it’s discouraged.)

Sorting and reversing lines of text

- `sort`
- `sort –f` Ignore case
- `sort –n` Numeric order
- `sort –r` Reverse sort
- `sort –nr` Reverse numeric sort

- `echo "Hello" | rev`

grep

- Grep finds patterns specified as regular expressions
  - **g**lobally search for **r**egular **e**xpression and **p**rint

- Finding words ending in –ing:
- `grep 'ing$' nyt.words |sort | uniq –c`

grep

- grep is a filter – you keep only some lines of the input
- `grep gh` keep lines containing "gh"
- `grep '^con'` keep lines beginning with "con"
- `grep 'ing$'` keep lines ending with "ing"
- `grep –v gh` keep lines NOT containing "gh"

- `grep -P` Perl regular expressions (extended syntax)
- `grep -P '^[A-Z]+$' nyt.words |sort|uniq -c` ALL UPPERCASE

Perl Regex

```
cat text|head -4
```

```
To Sherlock Holmes she is always the woman. I have seldom heard him mention her under any other name. In his eyes she eclipses and predominates the whole of her sex. It was not that he
elt any emotion akin to love for Irene Adler. All emotions, and that one particularly, were abhorrent to his cold, precise but admirably balanced mind. He was, I take it, the most perf
t reasoning and observing machine that the world has seen, but as a lover he would have placed himself in a false position. He never spoke of the softer passions, save with a gibe and
sneer. They were admirable things for the observer--excellent for drawing the veil from men's motives and actions. But for the trained reasoner to admit such intrusions into his own de
cate and finely adjusted temperament was to introduce a distracting factor which might throw a doubt upon all his mental results. Grit in a sensitive instrument, or a crack in one of h
 own high-power lenses, would not be more disturbing than a strong emotion in a nature such as his. And yet there was but one woman to him, and that woman was the late Irene Adler, of
bious and questionable memory.COM-IT118-32148:Week2NLP haithem.afli$
```

```
cat text|perl -pe 'tr/[a-z]/[A-Z]/;' |head -4
```

```
COM-IT118-32148:Week2NLP haithem.afli$ cat text |perl -pe 'tr/[a-z]/[A-Z]/;' |head -4
TO SHERLOCK HOLMES SHE IS ALWAYS THE WOMAN. I HAVE SELDOM HEARD HIM MENTION HER UNDER ANY OTHER NAME. IN HIS EYES SHE ECLIPSES AND PREDOMINATES THE WHOLE OF HER SEX. IT WAS NOT THAT HE
ELT ANY EMOTION AKIN TO LOVE FOR IRENE ADLER. ALL EMOTIONS, AND THAT ONE PARTICULARLY, WERE ABHORRENT TO HIS COLD, PRECISE BUT ADMIRABLY BALANCED MIND. HE WAS, I TAKE IT, THE MOST PERFE
T REASONING AND OBSERVING MACHINE THAT THE WORLD HAS SEEN, BUT AS A LOVER HE WOULD HAVE PLACED HIMSELF IN A FALSE POSITION. HE NEVER SPOKE OF THE SOFTER PASSIONS, SAVE WITH A GIBE AND A
SNEER. THEY WERE ADMIRABLE THINGS FOR THE OBSERVER--EXCELLENT FOR DRAWING THE VEIL FROM MEN'S MOTIVES AND ACTIONS. BUT FOR THE TRAINED REASONER TO ADMIT SUCH INTRUSIONS INTO HIS OWN DEL
CATE AND FINELY ADJUSTED TEMPERAMENT WAS TO INTRODUCE A DISTRACTING FACTOR WHICH MIGHT THROW A DOUBT UPON ALL HIS MENTAL RESULTS. GRIT IN A SENSITIVE INSTRUMENT, OR A CRACK IN ONE OF HIS
 OWN HIGH-POWER LENSES, WOULD NOT BE MORE DISTURBING THAN A STRONG EMOTION IN A NATURE SUCH AS HIS. AND YET THERE WAS BUT ONE WOMAN TO HIM, AND THAT WOMAN WAS THE LATE IRENE ADLER, OF DU
BIOUS AND QUESTIONABCOM-IT118-32148:COM-IT118-32148:COM-IT11COM-IT118-32148:Week2NLPCOM-IT118-32148:COM-IT118-32148:Week2NLPCOM-IT11COM-IT11COM-IT118-32148:Week2NLP haithemCOM-IT118-3214
8:COM-IT11COM-IT118-32148:COM-IT11COM-IT11COM-IT11COM-IT11COM-IT11COM-IT11COM-IT11COM-IT11COM-IT11COM-IT118-32148:Week2NLP haithemCOM-IT11COM-IT11COM-IT11COM-IT11COM-IT11COM-IT11COM-IT1
COM-IT118-32148:Week2NLP haithem.afli$
```

Extended Counting Exercises

How common are different sequences of vowels (e.g., ieu)?

Extended Counting Exercises

How common are different sequences of vowels (e.g., ieu)?

```
cat text|perl -pe 'tr/[A-Z]/[a-z]/; s/[^aieou]/\n/g;'|sort |uniq -c |sort –nr
```

Discussion

Recap

1) Natural language understanding, is it still challenging?

2) Describe NLP technologies with three words!

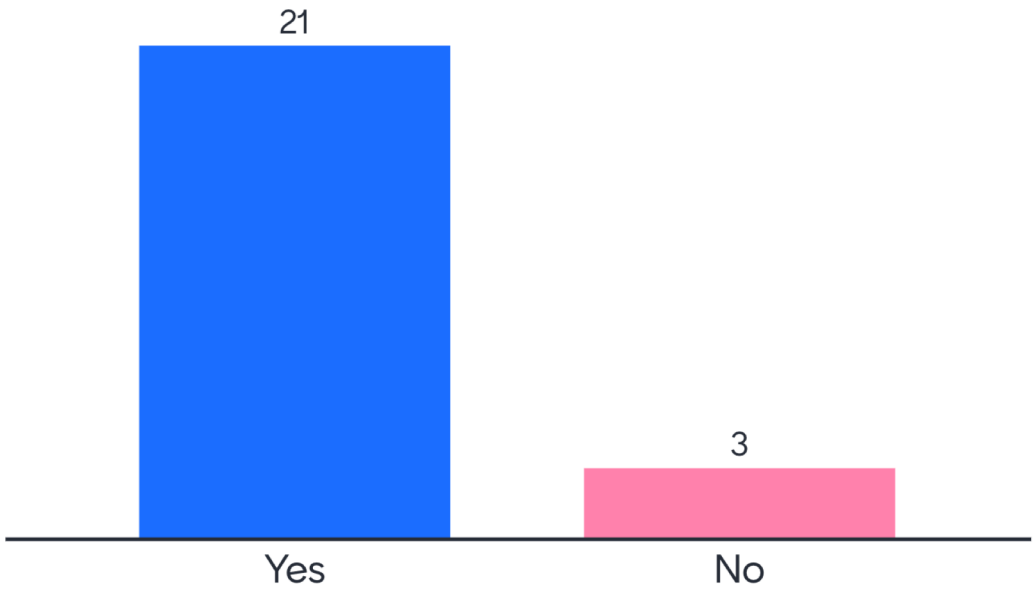
Go to https://www.menti.com/wxhysyji96 the digit code **46 08 69 4**

Or use the following QR Code

Go to **www.menti.com** and use the code **46 08 69 4**

Mentimeter

NLU is it still challenging?

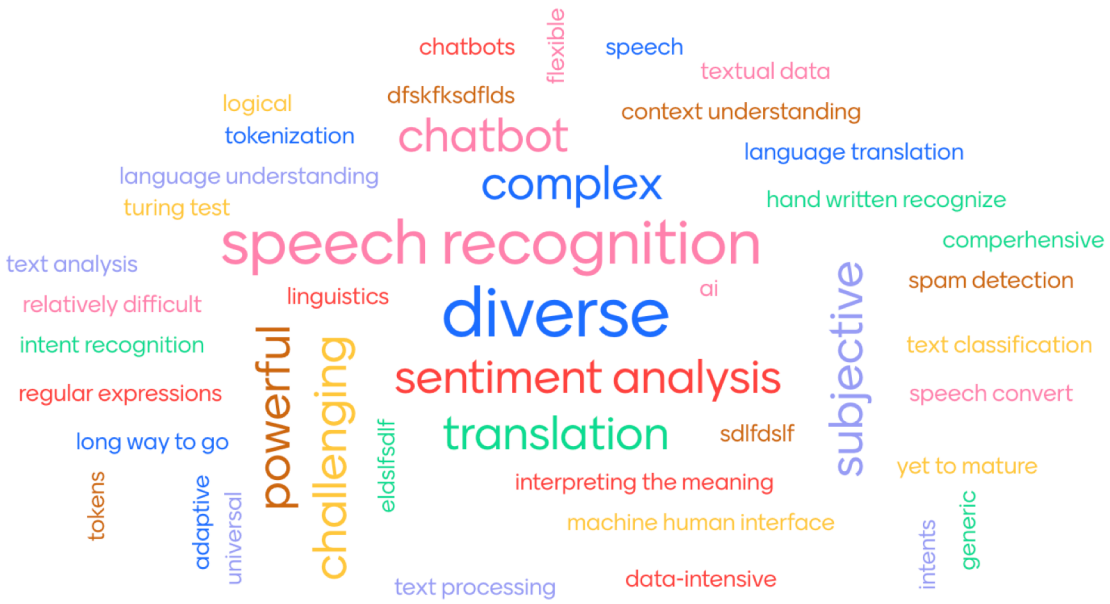


24

Go to **www.menti.com** and use the code **46 08 69 4**

Describe NLP technologies with three words!

Mentimeter



Discussion

Outline

1. Regular Expressions
2. *Unix tools*
3. ***Tokenization***
4. *Text processing with* Scikit-Learn

Issues in Tokenization

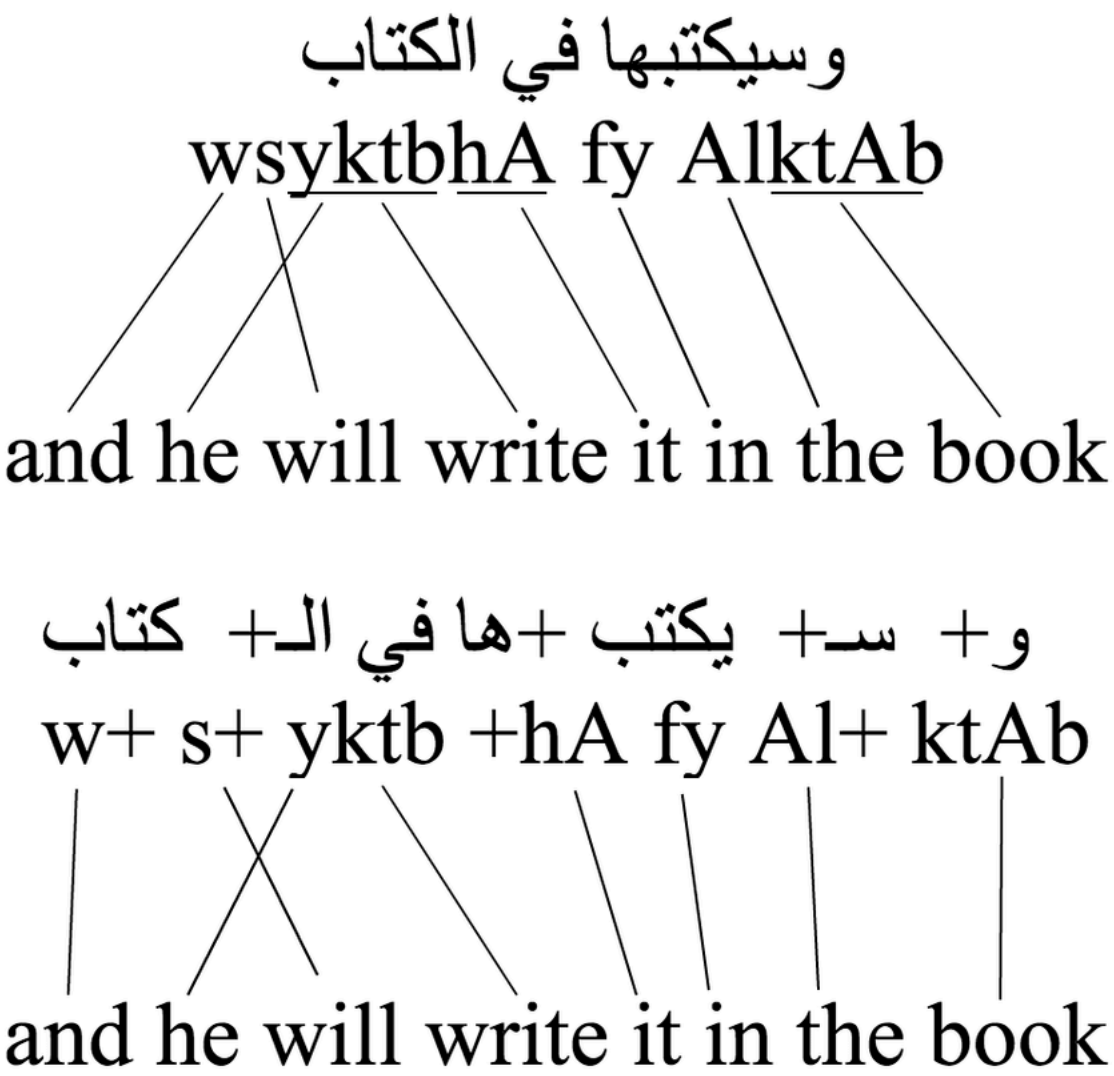
- `Finland's capital` → `Finland Finlands Finland's ?`
- `what're, I'm, isn't` → `What are, I am, is not`
- `Hewlett-Packard` → `Hewlett Packard ?`
- `state-of-the-art` → `state of the art ?`
- `Lowercase` → `lower-case lowercase lower case ?`
- `San Francisco` → one token or two?
- m.p.h., PhD. → ??

Tokenization: language issues

- French
  - ***L'ensemble*** → one token or two?
    - ***L*** ? ***L'*** ? ***Le*** ?
    - Want ***l'ensemble*** to match with ***un ensemble***

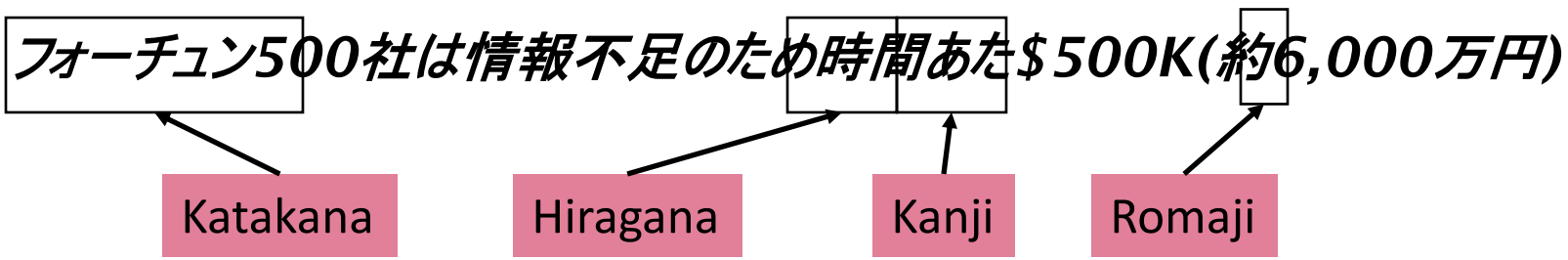
- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
  - 'life insurance company employee'
  - German information retrieval needs **compound splitter**

Tokenization: language issues



Tokenization: language issues

- Chinese and Japanese no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
  - Sharapova now lives in US southeastern Florida
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



End-user can express query entirely in hiragana!

Word Tokenization in Chinese

- Also called **Word Segmentation**
- Chinese words are composed of characters
- Characters are generally 1 syllable and 1 morpheme.
- Average word is 2.4 characters long.
- Standard baseline segmentation algorithm:

➢ Maximum Matching (also called Greedy)

Maximum Matching Word Segmentation Algorithm

Given a wordlist of Chinese, and a string.

1. Start a pointer at the beginning of the string
2. Find the longest word in dictionary that matches the string starting at pointer
3. Move the pointer over the word in string
4. Go to 2

Max-match segmentation illustration

- Thecatinthehat
- Thetabledownthere

the cat in the hat

the table down there

theta bled own there

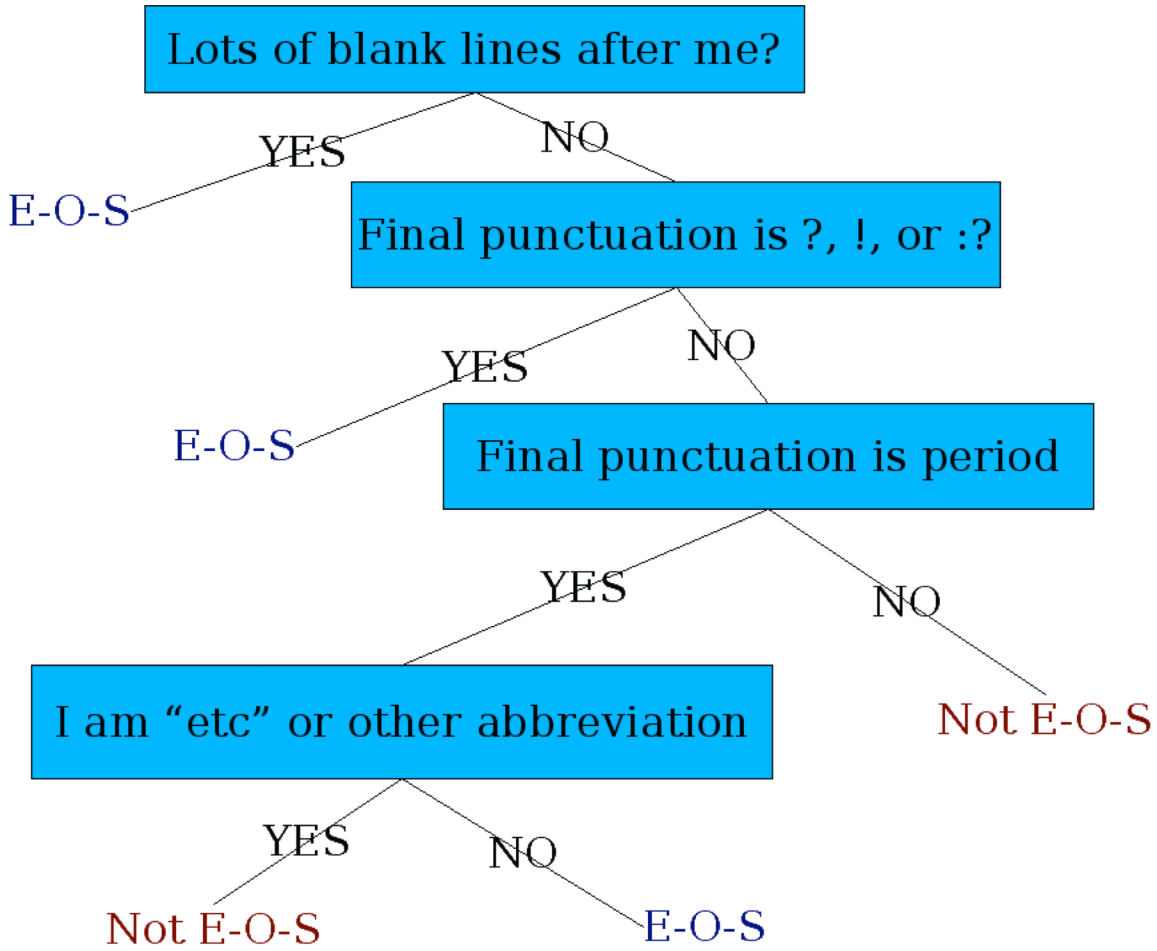
- Doesn’t generally work in English!

- But works astonishingly well in Chinese
- 莎拉波娃现在居住在美国东南部的佛罗里达。
- 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
- Modern probabilistic segmentation algorithms even better

Sentence Segmentation

- !, ? are relatively unambiguous
- Period "." is quite ambiguous
- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3
- Build a binary classifier
- Looks at a "."
- Decides EndOfSentence/NotEndOfSentence
- Classifiers: hand-written rules, regular expressions, or machine-learning

Determining if a word is end-of-sentence: a Decision Tree



More sophisticated decision tree features

- Case of word with ".": Upper, Lower, Cap, Number
- Case of word after ".": Upper, Lower, Cap, Number

- Numeric features
- Length of word with "."
- Probability(word with "." occurs at end-of-s)
- Probability(word after "." occurs at beginning-of-s)

Implementing Decision Trees

- A decision tree is just an if-then-else statement
- The interesting research is choosing the features
- Setting up the structure is often too hard to do by hand
- Hand-building only possible for very simple features, domains
- For numeric features, it's too hard to pick each threshold
- Instead, structure usually learned by machine learning from a training corpus

Decision Trees and other classifiers

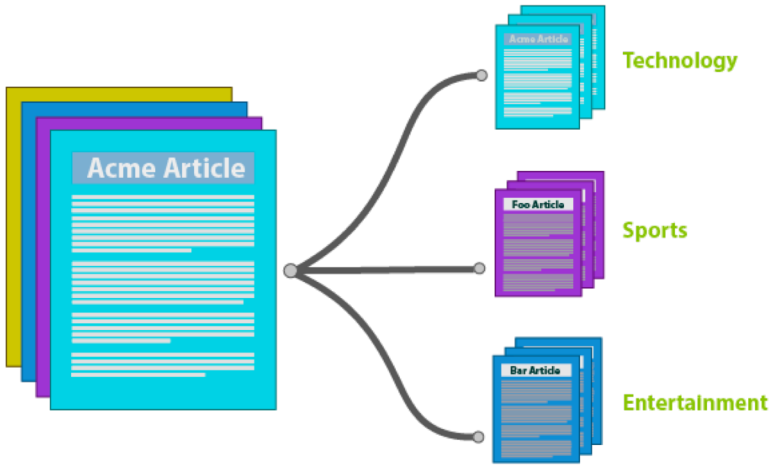
- We can think of the questions in a decision tree
- As features that could be exploited by any kind of classifier
- Logistic regression
- SVM
- Neural Nets
- etc.

Discussion

Outline

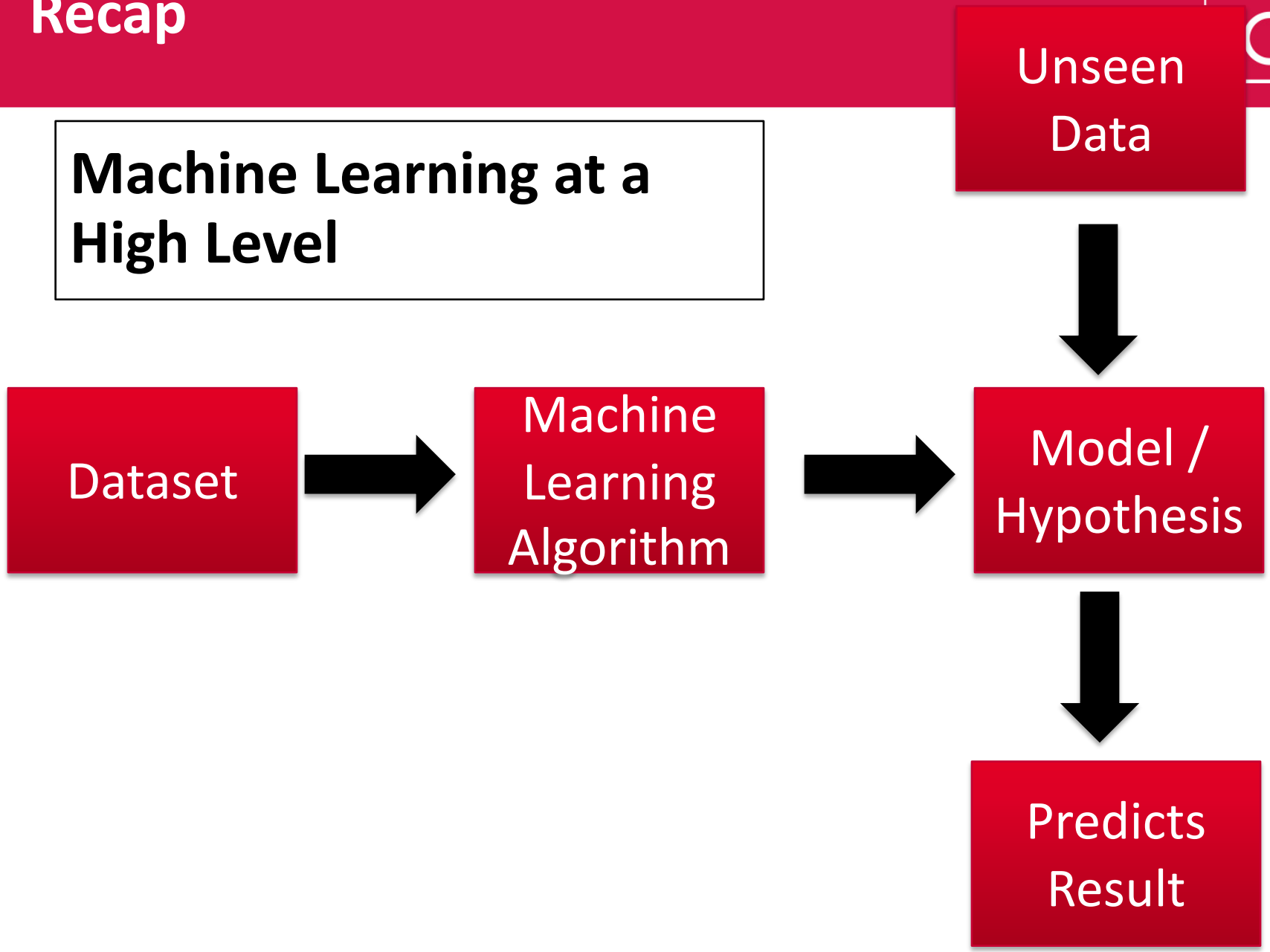
1. Regular Expressions
2. *Unix tools*
3. *Tokenization*
4. ***Text processing with* Scikit-Learn**

Text Processing

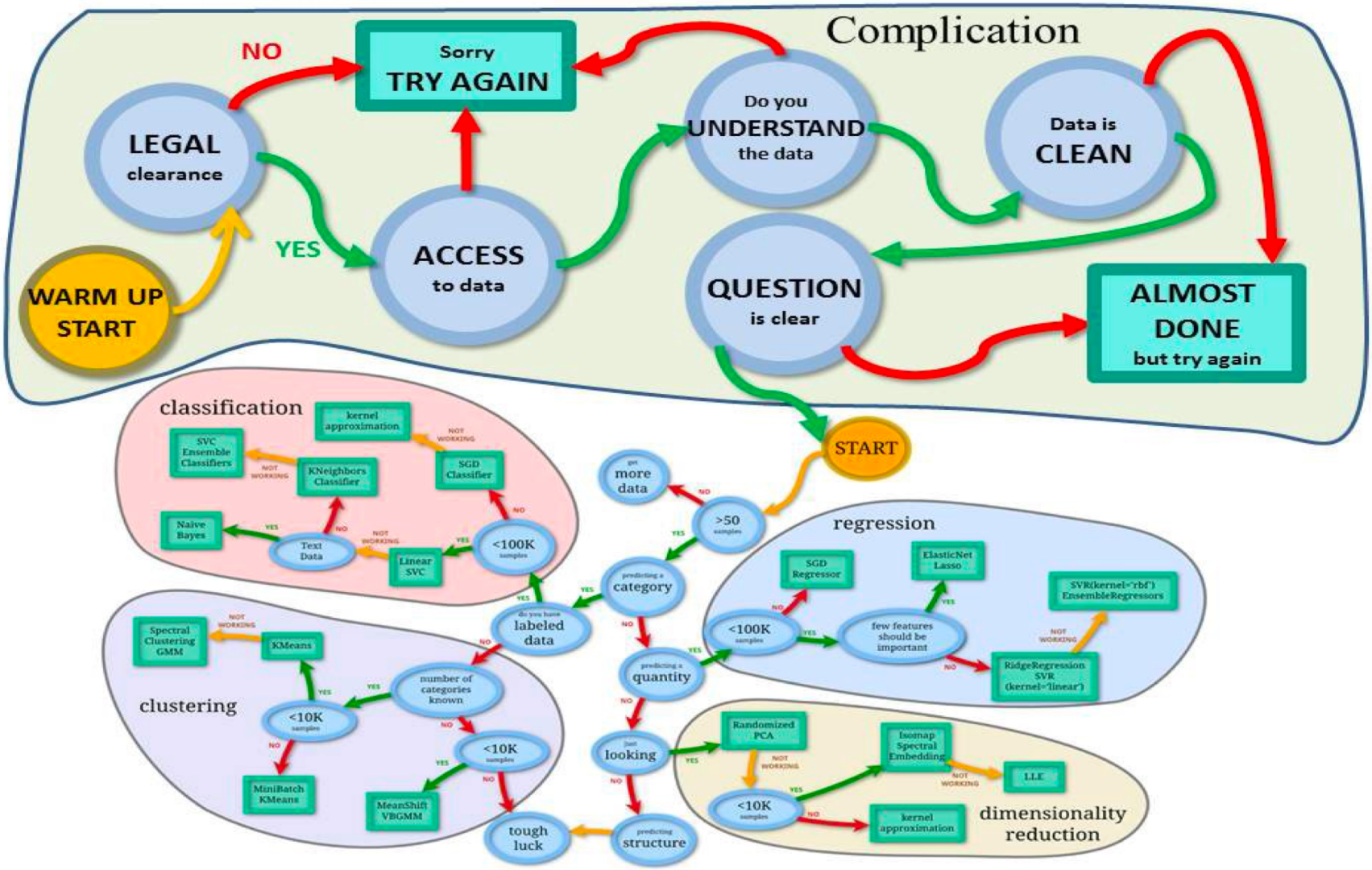


Recap

Machine Learning at a High Level



Recap


Complication
NO
Sorry
TRY AGAIN
LEGAL
clearance
YES
ACCESS
to data
Do you
UNDERSTAND
the data
Data is
CLEAN
WARM UP
START
QUESTION
is clear
ALMOST
DONE
but try again
START
classification
kernel approximation
SVC Ensemble Classifiers
KNeighbors Classifier
SGD Classifier
Naive Bayes
Text Data
Linear SVC
<100K
more data
>50
category
labeled data
regression
SGD Regressor
ElasticNet Lasso
<100K
few features should be important
SVR(kernel='rbf') EnsembleRegressors
RidgeRegression SVR (kernel='linear')
quantity
clustering
Spectral Clustering GMM
KMeans
number of categories known
<10K
<10K
MiniBatch KMeans
MeanShift VBGMM
looking
Randomized PCA
Isomap Spectral Embedding
LLE
<10K
kernel approximation
dimensionality reduction
tough luck
structure

Assessing Accuracy

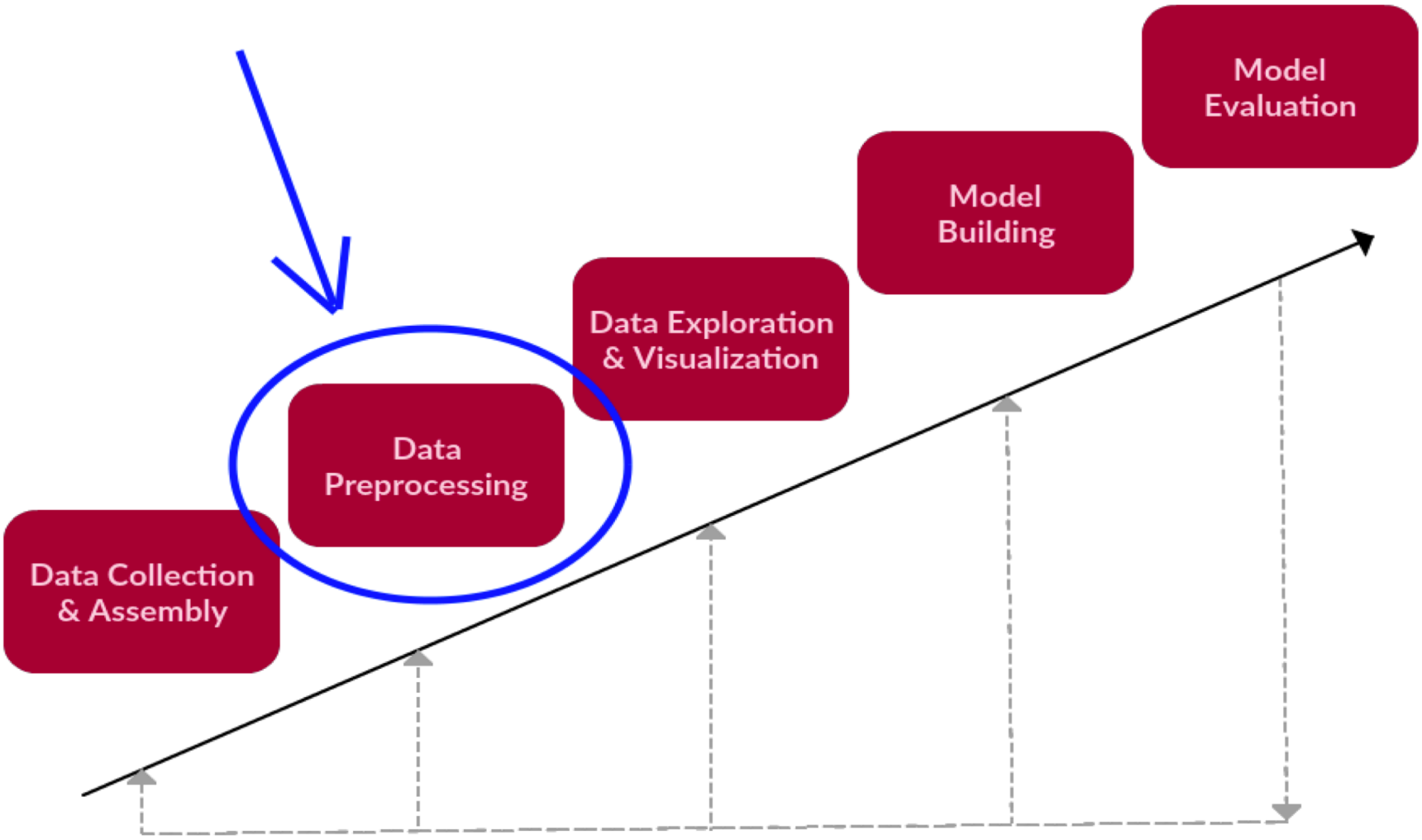
- Assuming I have separate training and test data I might have the following arrays
  - features_train, labels_train
  - features_test, labels_test

```
from sklearn import metrics
from sklearn import tree

# creates a new decision tree object classifier
clf = tree.DecisionTreeClassifier()

# trains the classifier pass it the training data and classes
clf = clf.fit(features_train, labels_train)

# predict the class for an unseen example
results= clf.predict(features_test)

print (   metrics.accuracy_score(results, labels_test)   )
```

Notice in this example when we call **predict** we are passing it as 2D NumPy array.

The **accuracy_score** function (available in the metrics module) will count the number of classes we correctly predicated and express that as a **percentage** of the total number of test data instances.

Text Processing



A few important notes about Scikit Learn

- The following are some important requirements that you should keep in mind when working with Scikit learn.

1. Features and classes/target values are **separate** objects (data structures)
2. Features and classes should be **numerical**
3. Features and classes should be **NumPy** arrays
4. Features and classes should have a **specific shape**
   - Features should be 2D (Columns correspond to numbers of features and rows are number of data instances)
   - Class array should be one dimensional with same number of instances as there are data instances in the features array

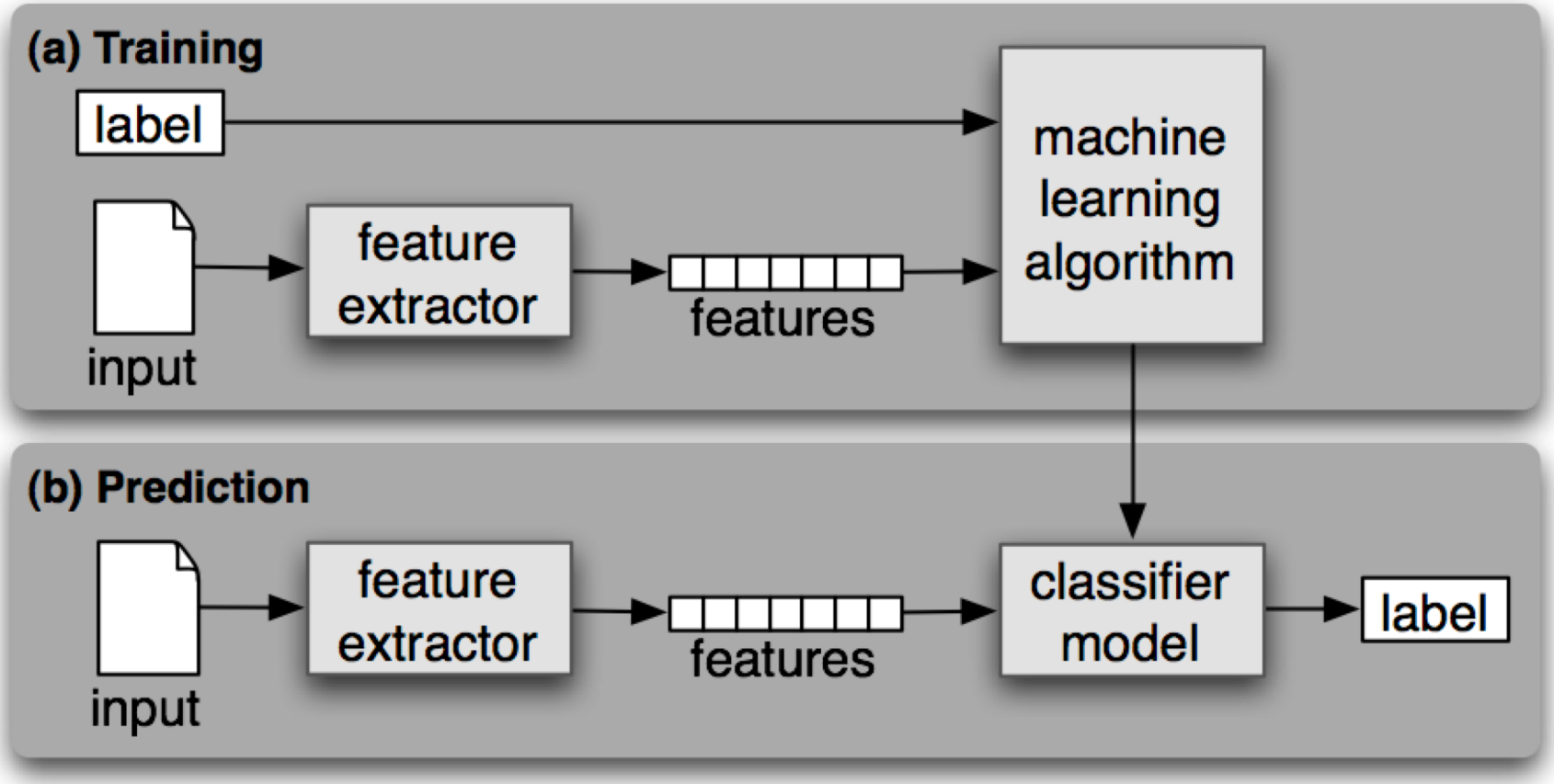
Text Classification with Python and Scikit-Learn

1. Importing Libraries
2. Importing The dataset
3. **Text Preprocessing**
4. Converting Text to Numbers
5. Training and Test Sets
6. Training Text Classification Model and Predicting Classes/Sentiment
7. Evaluating The Model
8. Saving and Loading the Model

Preparing text data for ML

- Text data requires special preparation before you can start using it for predictive modeling.
- The text must be cleaned and tokenized.
- Then the words need to be encoded as **integers or floating point values** for use as input to a machine learning algorithm, called **feature extraction** (or **vectorization**).

Feature extraction



Processing text

- We cannot work with text directly when using machine learning algorithms.

- Instead, we need to convert the text to numbers.
- We call **vectorization** the general process of turning a collection of text documents into numerical feature vectors.

- This specific strategy (tokenization, counting and normalization) is called the **Bag of Words** or “Bag of n-grams” representation.
- Documents are described by word occurrences while completely ignoring the relative position information of the words in the document.

We define a word as a vector

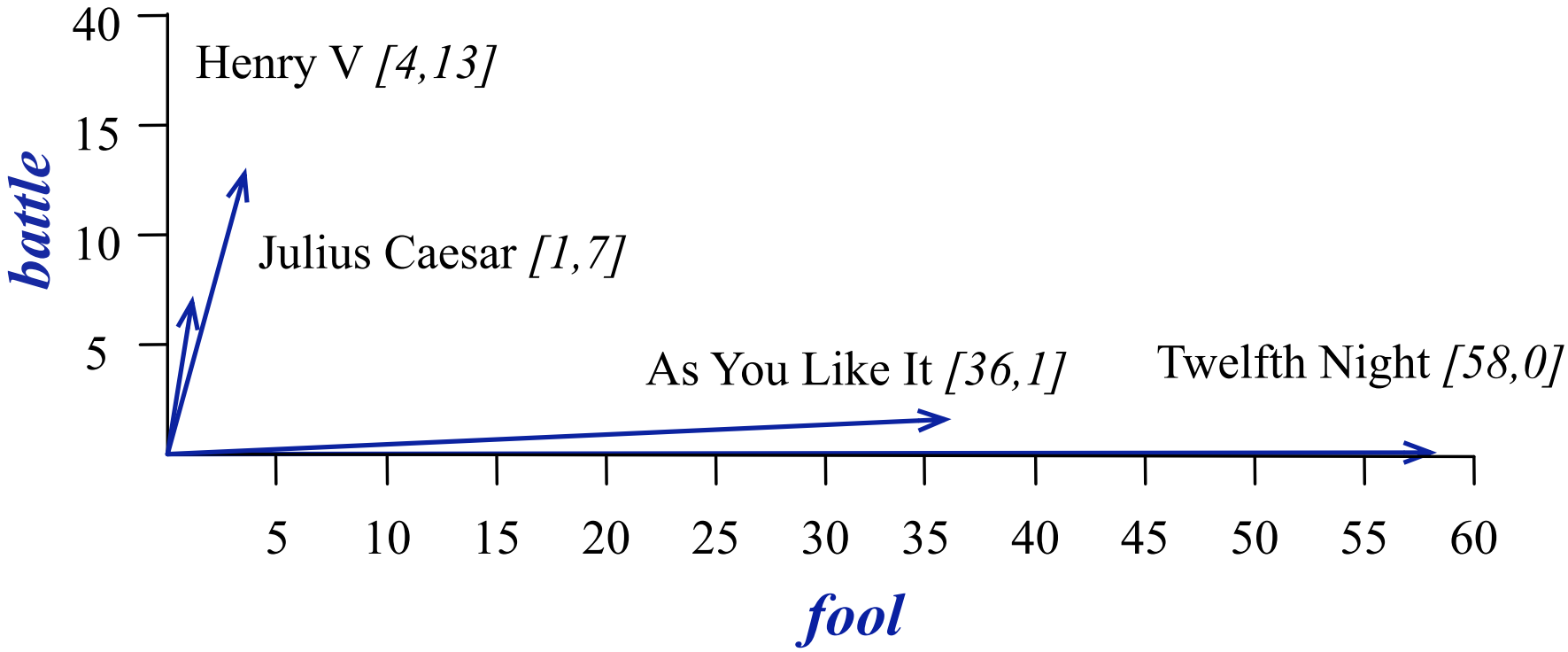
- Called an "embedding" because it's embedded into a space
- The standard way to represent meaning in Natural Language Processing (NLP)
- Fine-grained model of meaning for similarity
  - NLP tasks like sentiment analysis
    - With words, requires **same** word to be in training and test
    - With embeddings: ok if **similar** words occurred!!!
  - Question answering, conversational agents, etc

Term-document matrix

Each document is represented by a vector of words

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Visualizing document vectors



Vectors are the basis of information retrieval

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Vectors are similar for the two comedies
Different than the history

Comedies have more fools and wit and fewer battles.

Words can be vectors too

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

battle is "the kind of word that occurs in Julius Caesar and Henry V"

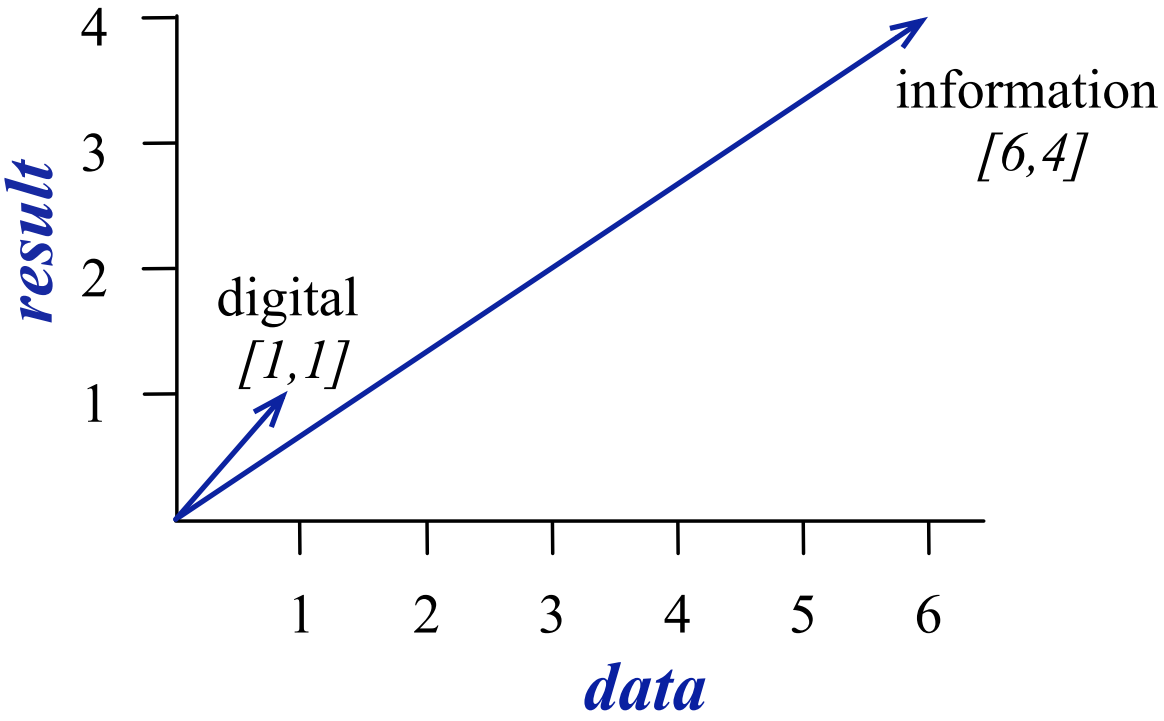
fool is "the kind of word that occurs in comedies, especially Twelfth Night"

More common: word-word matrix (or "term-context matrix")

- Two **words** are similar in meaning if their context vectors are similar

sugar, a sliced lemon, a tablespoonful of	**apricot**	jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	


4
3
2
1
result
digital
[1,1]
information
[6,4]
1
2
3
4
5
6
data

Reminders from linear algebra

$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

$$\text{vector length} \quad |\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

Cosine for computing similarity

$$\text{cosine}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

v_i is the count for word v in context i

w_i is the count for word w in context i.

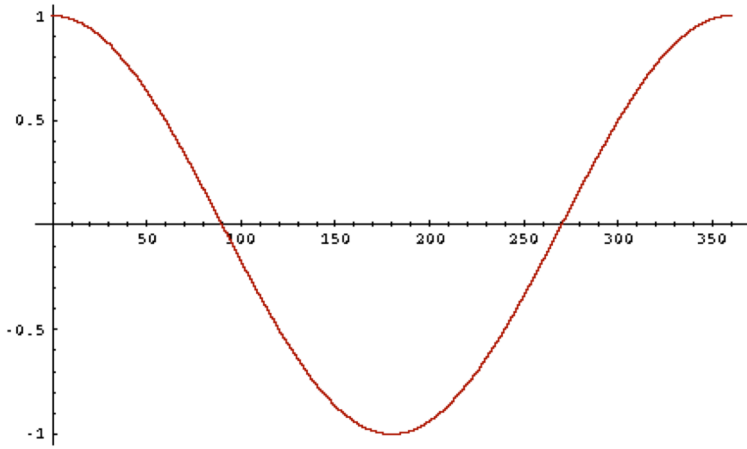
Cos(v,w) is the cosine similarity of v and w

$$\vec{a}\cdot\vec{b} = |\vec{a}||\vec{b}|\cos\theta$$

$$\frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|} = \cos\theta$$

Cosine as a similarity metric

- -1: vectors point in opposite directions
- +1: vectors point in same directions
- 0: vectors are orthogonal



- Frequency is non-negative, so cosine range 0-1

$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\bullet\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\bullet\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

	large	data	computer
apricot	1	0	0
digital	0	1	2
information	1	6	1

Which pair of words is more similar?

cosine(apricot,information) =

$$\frac{1+0+0}{\sqrt{1+0+0}\ \sqrt{1+36+1}} = \frac{1}{\sqrt{38}} = .16$$

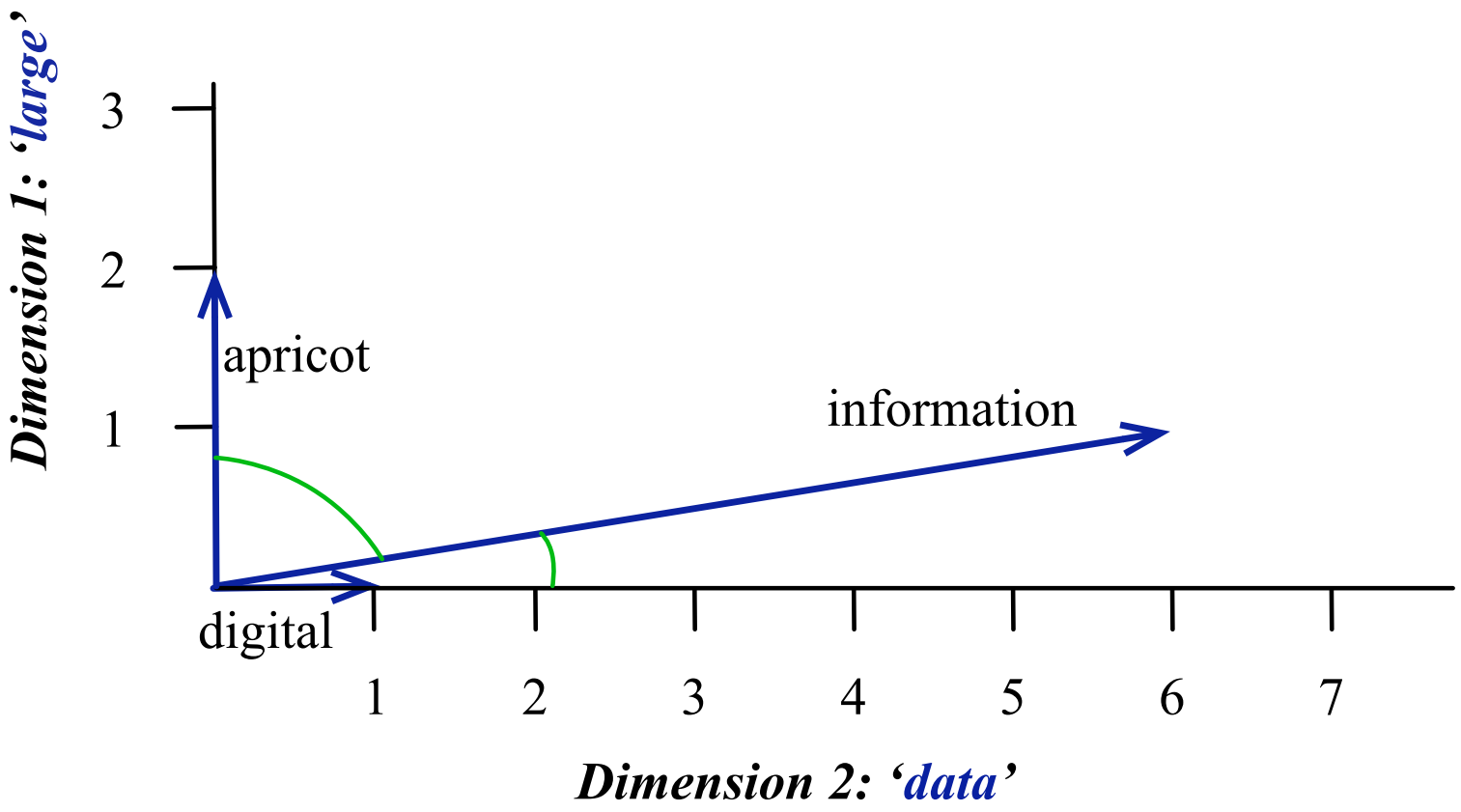
cosine(digital,information) =

$$\frac{0+6+2}{\sqrt{0+1+4}\ \sqrt{1+36+1}} = \frac{8}{\sqrt{38}\sqrt{5}} = .58$$

cosine(apricot,digital) =

$$\frac{0+0+0}{\sqrt{1+0+0}\ \sqrt{0+1+4}} = 0$$

Visualizing cosines (well, angles)



Bags of words

- Assign a fixed integer id to each **word occurring** in any document of the training set (for instance by building a dictionary from words to integer indices).
- For each document **#i**, count the number of occurrences of each word w and store it in **X[i, j]** as the value of feature **#j** where j is the index of word w in the dictionary.
- The bags of words representation implies that **n_features** is the number of distinct words in the corpus

Text Classification with Python and Scikit-Learn

1. Importing Libraries
2. Importing The dataset
3. Text Preprocessing
4. **Converting Text to Numbers**
5. Training and Test Sets
6. Training Text Classification Model and Predicting Classes/Sentiment
7. Evaluating The Model
8. Saving and Loading the Model

CountVectorizer

Home Installation Documentation Examples

Google Custom Search

Fork me on G

sklearn.feature_extraction.text.CountVectorizer

class `sklearn.feature_extraction.text.` **`CountVectorizer`** (*input='content', encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, stop_words=None, token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), analyzer='word', max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.int64'>*) [source]

Convert a collection of text documents to a matrix of token counts

This implementation produces a sparse representation of the counts using scipy.sparse.csr_matrix.

If you do not provide an a-priori dictionary and you do not use an analyzer that does some kind of feature selection then the number of features will be equal to the vocabulary size found by analyzing the data.

Read more in the User Guide.

Parameters: **input : *string {'filename', 'file', 'content'}***

If 'filename', the sequence passed as an argument to fit is expected to be a list of filenames that need reading to fetch the raw content to analyze.

If 'file', the sequence items must have a 'read' method (file-like object) that is called to fetch the bytes in memory.

Otherwise the input is expected to be the sequence strings or bytes items are expected to be analyzed directly.

encoding : *string, 'utf-8' by default.*

Word Counts with CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer

# list of text documents
text = ["The quick brown fox jumped over the lazy dog."]

# create the transform
vectorizer = CountVectorizer()
# tokenize and build vocab
vectorizer.fit(text)

# summarize
print(vectorizer.vocabulary_)
# encode document
vector = vectorizer.transform(text)
# summarize encoded vector

print(vector.shape)
print(type(vector))
print(vector.toarray())
```

```
{'dog': 1, 'fox': 2, 'over': 5, 'brown': 0, 'quick': 6, 'the': 7, 'lazy': 4, 'jumped': 3}

(1, 8)

<class 'scipy.sparse.csr.csr_matrix'>

[[1 1 1 1 1 1 1 2]]
```

example from machine learning mastery

Using the vectorizer

We can use the vectorizer to encode a document with one word in the vocab and one word that is not.

```
# encode another document
text2 = ["the puppy"]


vector = vectorizer.transform(text2)
print(vector.toarray())
```

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

But raw occurrence is a bad representation

- Occurrence is clearly useful; if *sugar* appears a lot near *apricot*, that's useful information.

But overly frequent words like *the*, *it,* or *they* are not very informative about the context

Need a function that resolves this paradox!

TF-IDF

- To avoid these potential discrepancies it suffices to divide the **number of occurrences of each word** in a document by **the total number of words in the document**: these new features are called tf for Term Frequencies.

- Another refinement on top of tf is to **downscale weights** for words that **occur in many documents** in the corpus and are therefore less informative than those that occur only in a smaller portion of the corpus.

TF-IDF

- This downscaling is called tf–idf for “Term Frequency times Inverse Document Frequency”.

- **Term Frequency**: This summarizes how often a given word appears within a document.
- **Inverse Document Frequency**: This downscales words that appear a lot across documents.

tf-idf: combine two factors

- **tf: term frequency**. frequency count (usually log-transformed):

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- **Idf: inverse document frequency: tf-**

$$\mathrm{idf}_i = \log\left(\frac{N}{\mathrm{df}_i}\right)$$

Total # of docs in collection

# of docs that have word i

Words like "the" or "good" have very low idf

tf-idf value for word t in document d:

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

Tf-idf is a sparse representation

- tf-idf vectors are
  - **long** (length |V|= 20,000 to 50,000)
  - **sparse** (most elements are zero)

TF-IDF with Scikit-Learn

- The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents.

- Alternately, if you already have a learned CountVectorizer, you can use it with a TfidfTransformer to just calculate the inverse document frequencies and start encoding documents.

Word Counts with CountVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
# list of text documents
text = ["The quick brown fox jumped over the lazy dog.",
                    "The dog.",
                    "The fox"]

# create the transform
vectorizer = TfidfVectorizer()
# tokenize and build vocab
vectorizer.fit(text)

# summarize
print(vectorizer.vocabulary_)
print(vectorizer.idf_)
# encode document
vector = vectorizer.transform([text[0]])

# summarize encoded vector
print(vector.shape)
print(vector.toarray())
```

```
{'fox': 2, 'lazy': 4, 'dog': 1,
'quick': 6, 'the': 7, 'over': 5,
'brown': 0, 'jumped': 3}

[ 1.69314718 1.28768207
1.28768207 1.69314718
1.69314718 1.69314718
1.69314718 1. ]

(1, 8)

[[ 0.36388646 0.27674503
0.27674503 0.36388646
0.36388646 0.36388646
0.36388646 0.42983441]]
```

example from machine learning mastery

Word Counts with CountVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
# list of text documents
text = ["The quick brown fox jumped over the lazy dog.",
                    "The dog.",
                    "The fox"]

# create the transform
vectorizer = TfidfVectorizer()
# tokenize and build vocab
vectorizer.fit(text)

# summarize
print(vectorizer.vocabulary_)
print(vectorizer.idf_)
# encode document
vector = vectorizer.transform([text[0]])

# summarize encoded vector
print(vector.shape)
print(vector.toarray())
```

A vocabulary of 8 words is learned from the documents and each word is assigned a unique integer index in the output vector.

```
{'fox': 2, 'lazy': 4, 'dog': 1,
'quick': 6, 'the': 7, 'over': 5,
'brown': 0, 'jumped': 3}

[ 1.69314718 1.28768207
1.28768207 1.69314718
1.69314718 1.69314718
1.69314718 1. ]

(1, 8)

[[ 0.36388646 0.27674503
0.27674503 0.36388646
0.36388646 0.36388646
0.36388646 0.42983441]]
```

Word Counts with CountVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
# list of text documents
text = ["The quick brown fox jumped over the lazy dog.",
                    "The dog.",
                    "The fox"]

# create the transform
vectorizer = TfidfVectorizer()
# tokenize and build vocab
vectorizer.fit(text)

# summarize
print(vectorizer.vocabulary_)
print(vectorizer.idf_)
# encode document
vector = vectorizer.transform([text[0]])

# summarize encoded vector
print(vector.shape)
print(vector.toarray())
```

The inverse document frequencies are calculated for each word in the vocabulary, assigning the lowest score of 1.0 to the most frequently observed word: "*the*" at index 7.

```
{'fox': 2, 'lazy': 4, 'dog': 1,
'quick': 6, 'the': 7, 'over': 5,
'brown': 0, 'jumped': 3}

[ 1.69314718 1.28768207
1.28768207 1.69314718
1.69314718 1.69314718
1.69314718 1.        ]

(1, 8)

[[ 0.36388646 0.27674503
0.27674503 0.36388646
0.36388646 0.36388646
0.36388646 0.42983441]]
```

Word Counts with CountVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
# list of text documents
text = ["The quick brown fox jumped over the lazy dog.",
                    "The dog.",
                    "The fox"]

# create the transform
vectorizer = TfidfVectorizer()
# tokenize and build vocab
vectorizer.fit(text)

# summarize
print(vectorizer.vocabulary_)
print(vectorizer.idf_)
# encode document
vector = vectorizer.transform([text[0]])

# summarize encoded vector
print(vector.shape)
print(vector.toarray())
```

Finally, the first document is encoded as an 8-element sparse array and we can review the final scorings of each word with different values for "*the*", "*fox*", and "*dog*" from the other words in the vocabulary.

```
{'fox': 2, 'lazy': 4, 'dog': 1,
'quick': 6, 'the': 7, 'over': 5,
'brown': 0, 'jumped': 3}

[ 1.69314718 1.28768207
1.28768207 1.69314718
1.69314718 1.69314718
1.69314718 1. ]

(1, 8)

[[ 0.36388646 0.27674503
0.27674503 0.36388646
0.36388646 0.36388646
0.36388646 0.42983441]]
```

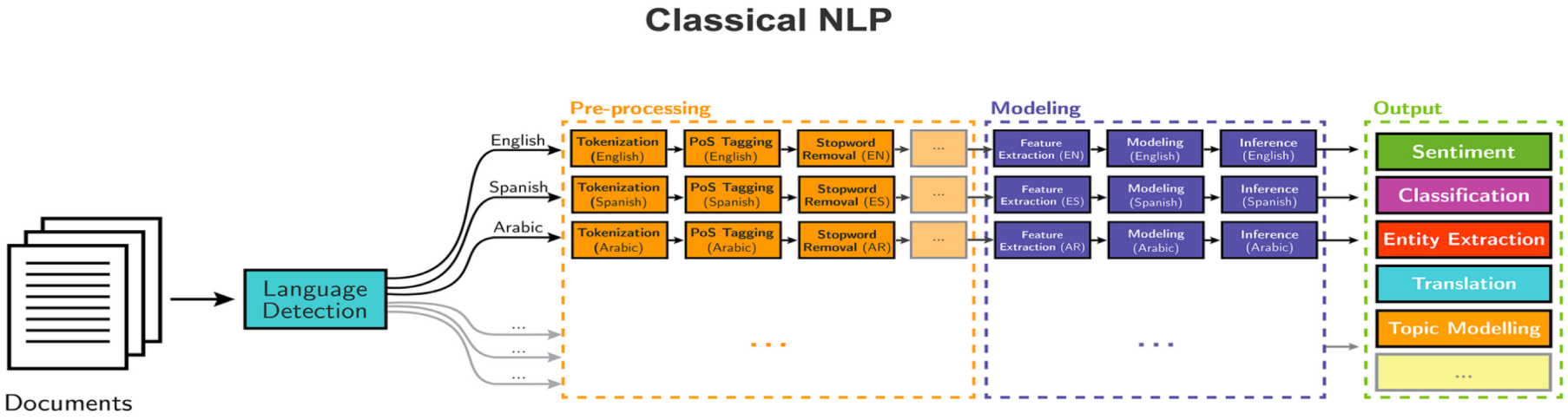
Word Counts with TfidfVectorizer

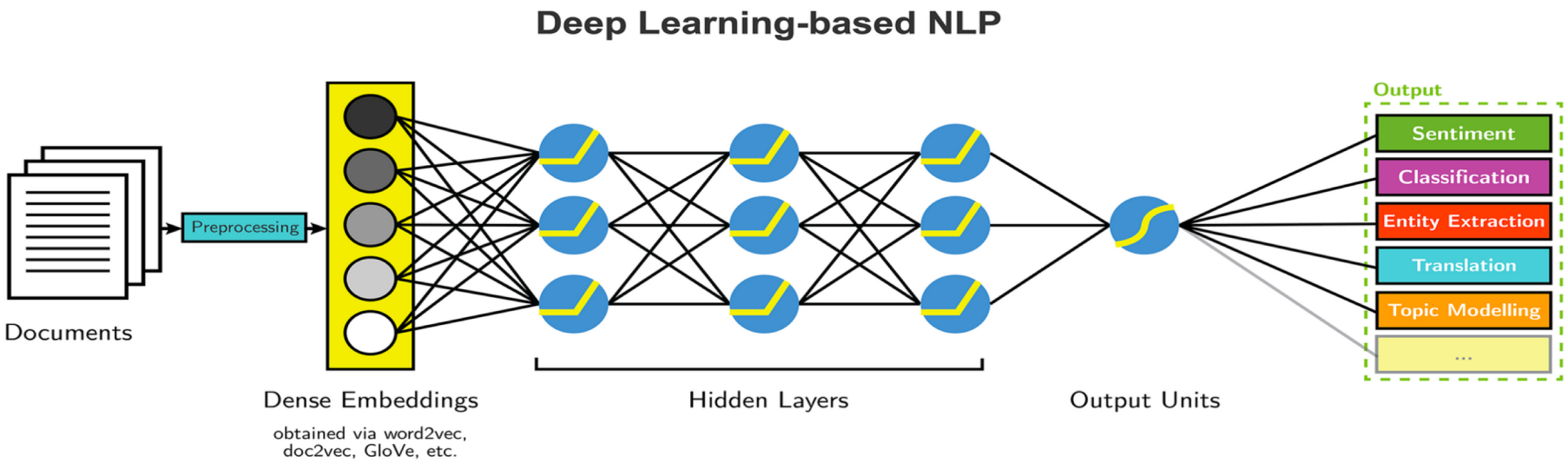
```
from sklearn.feature_extraction.text import TfidfVectorizer
# list of text documents
text = ["The quick brown fox jumped over the lazy dog.",
                    "The dog.",
                    "The fox"]

# create the transform
vectorizer = TfidfVectorizer()
# tokenize and build vocab
vectorizer.fit(text)

# summarize
print(vectorizer.vocabulary_)
print(vectorizer.idf_)
# encode document
vector = vectorizer.transform([text[0]])

# summarize encoded vector
print(vector.shape)
print(vector.toarray())
```

The scores are normalized to values between 0 and 1 and the encoded document vectors can then be used directly with most machine learning algorithms.

```
{'fox': 2, 'lazy': 4, 'dog': 1,
'quick': 6, 'the': 7, 'over': 5,
'brown': 0, 'jumped': 3}

[ 1.69314718 1.28768207
1.28768207 1.69314718
1.69314718 1.69314718
1.69314718 1. ]

(1, 8)

[[ 0.36388646 0.27674503
0.27674503 0.36388646
0.36388646 0.36388646
0.36388646 0.42983441]]
```

Natural Language Processing





Discussion

Some content was adapted from Speech and Language Processing - Jurafsky and Martin

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computer Science

http://www.cit.ie

Thank you

Haithem.afli@cit.ie

@AfliHaithem