# Lecture 1: Introduction to Text Mining

Aurelia Power

Hons. Degree in Computing

H4016 Text Analysis - 2024

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

# Module Aim and Overview

AIM: to give learners an understanding of how natural language processing (NLP) techniques and machine learning (ML) techniques can be applied to text analytics.

Overview:

- **Text Mining Methodology – CRISP**
- **Cleaning and pre-processing**
- **Text Mining Algorithms:**
  - **Clustering Algorithms**
  - **Classification Algorithms**
- **Big Data Analytics**

# Assessment and Enrollment

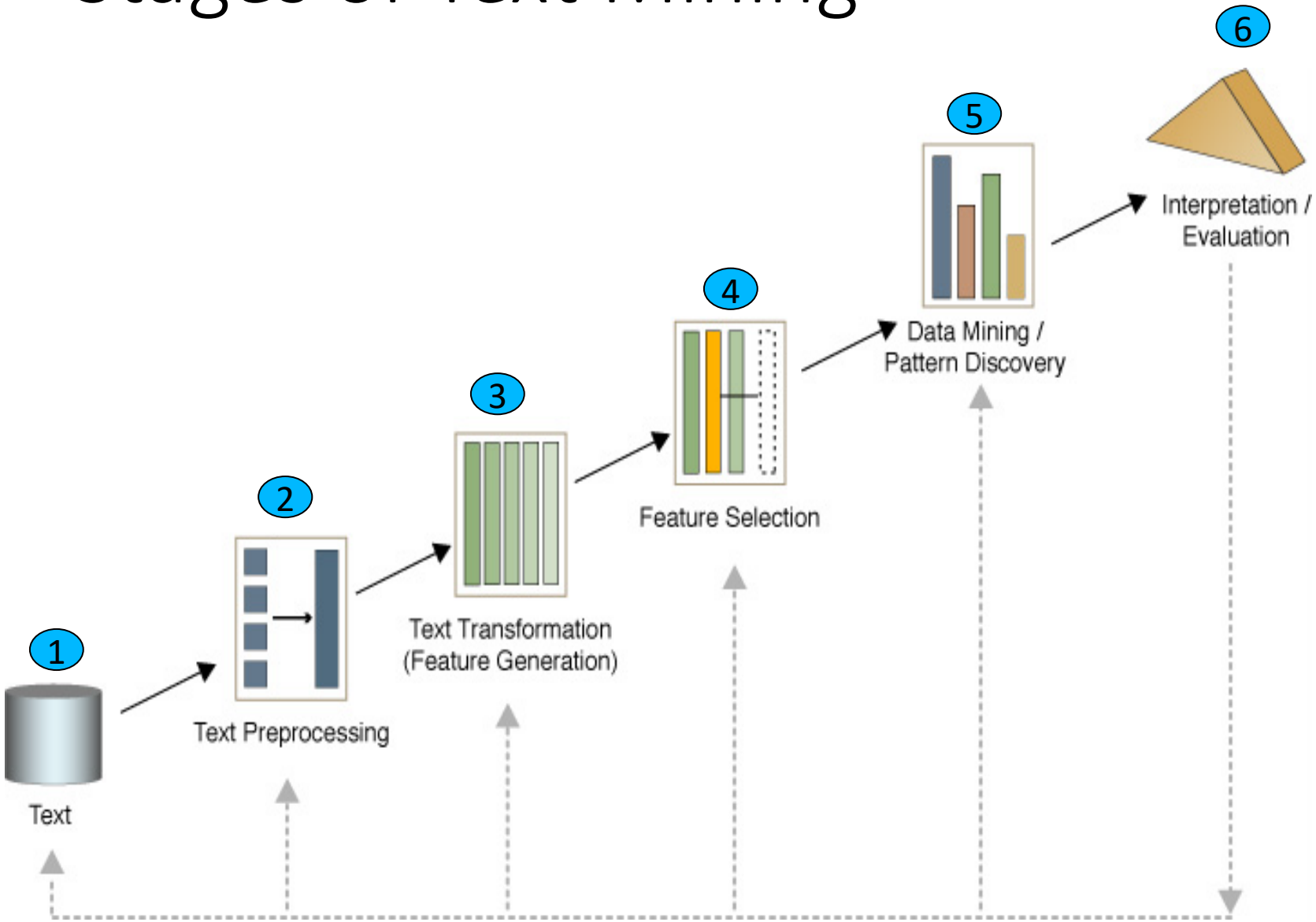
- Assessment:
  - **CA**: 40%
    - Moodle Test – 10%
    - Project - 30%: it includes peer review of interim report.
  - **Exam:** 60%
    - **Note**: All content covered in the lectures and labs, written or presented verbally, is examinable.
- Enrolment Key: **textMining2**

# What is Text Analytics?

- “The non-trivial extraction of implicit, previously unknown, and potentially useful information from (large amounts of) textual data”.

- An exploration and analysis of textual (natural-language) data by automatic and semi automatic means to discover new knowledge.

# Stages of Text Mining



# Stages of Text Mining

1. Identify the business and mining objectives.
2. Collect and understand representative textual data.
3. Cleaning and preprocessing the textual data.
4. Feature Generation.
5. Feature Selection: select the most predictive features based on their relevance.
6. Data Mining/Pattern Discovery - applying ML algorithms:
    1. clustering – unsupervised learning
    2. classification – supervised learning
7. Evaluation and Interpretation of the Results.

# Textual Data…

Some ambiguous and (possibly) funny headlines…

- Children make delicious snacks
- Dead expected to rise
- Stolen painting found by tree
- Red tape holds up new bridges

# Textual Data…

- Working with text is challenging… because language, in general, is ambiguous, poorly structured and, most of the time, relies on extra contextual cues.
- Some  more ambiguities… (from Beatrice Santorini's http://www.ling.upenn.edu/~beatrice/humor/): If you have to write a letter of recommendation for a fired employee, here are a few suggested phrases:

**Lexical ambiguity**

| | |
|---|---|
| For a chronically absent employee… | A man like him is hard to find. |
| For a dishonest employee… | He's an unbelievable worker. |

**Structural ambiguity**

| | |
|---|---|
| For a chronically absent employee… | It seemed her career was just taking off. |
| For a dishonest employee… | Her true ability was deceiving. |

# Textual Data… Other Sources of Ambiguity

- Ellipsis and parallelism:
  Jo likes NLP and Jojo Guinness.
- Adj modifier:
  We saw the beautiful Dublin and Navan.
- Negation:
  Ms. Doyle likes tea with no sugar and milk.
- Reference issues:
  Jane got a book for Helen. She was very happy.
- Metaphor:
  She moved me to tears.
- Subjectivity:
  I believe that you will do well this semester.

# Textual Data… Other Challenges

- Neologisms and slang:
  - Emoticon, selfie, twitterholic, Twictionary
  - www.urbandictionary.com
  - www.internetslang.com
- Acronyms and Abbreviations
  - anon -> anonymous
  - BRB -> be right back
- Vagueness:
  - The blogger posted twice (e.g., gender vagueness)
- Typos and grammatical errors:
  - misspellings
  - S-V disagreement
  - Double negation, etc.
- Emoticons/Emojis:
  - ☺ ☹
- Substitution:
  - L8r -> later
- So high dimensionality…

# Textual Data… More Difficulties

- Complex sentences:

  "My very photogenic mother died in a freak accident (picnic, lightning) when I was three, and, save for a pocket of warmth in the darkest past, nothing of her subsists within the hollows and dells of memory, over which, if you can still stand my style (I am writing under observation), the sun of my infancy had set: surely, you all know those redolent remnants of day suspended, with the midges, about some hedge in bloom or suddenly entered and traversed by the rambler, at the bottom of a hill, in the summer dusk; a furry warmth, golden midges." (Vladimir Nabokov – Lolita)

- Pragmatic Elements: Presupposition, Implicature, Word Knowledge, Inference: what has been left out…
  - I went home and I unlocked the door with the spare key.
    - Implies I have a home and that the home is a house that has a door.
    - Presupposes the door was locked.
    - Assumes knowledge about what a key is… and so on.
- Humour, Sarcasm, Satire: "Drinking alcohol daily reduces the risk of growing old by 68% " (WWN, 2018)

- And many other difficulty sources…

# Text Mining Applications

- Spam Detection/Email Filtering
  - Using classification to sort email as spam or not-spam
  - *Dear Aurelia, Following a review of your application, your loan has …* - *not spam*
  - *Get up to $5000 loans in only few minutes..* - *spam*

- Social Media Analysis
  - Cyberbullying/Abusive Language Detection
  - *This is useless…* - not abusive
  - *You are useless…* - abusive

- Surveys/Questionnaires Analysis
  - Automation of answers to open ended questions.

- Sentiment Analysis: analysing text to understand customer reviews/recommendations:
  - *This is worth watching…* - positive
  - *Not a nice watch…* - negative

# Text Mining Applications

- Call Center Analysis:
  - Using classification to categorise customer issues.

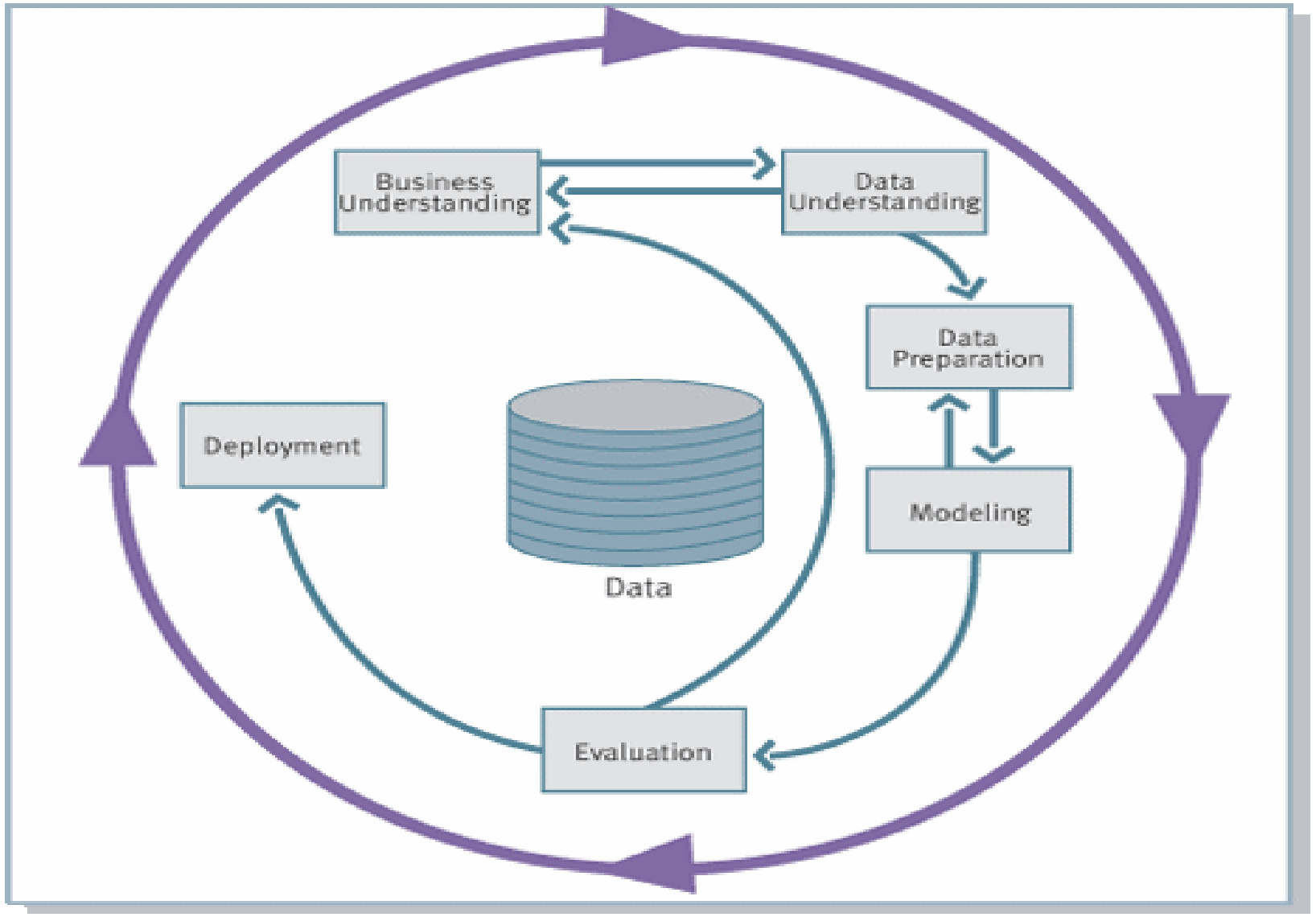
- Discovery Search:
  - Reviewing and analyzing scientific/scholarly articles.

- Competitive/Government Intelligence:
  - Automation of monitoring newsfeeds, new products launches, acquisitions, emails, etc.

- Analysis of Interviews and conversations:
  - Automation of reviewing patient reports

- And many others…

# CRISP-DM for Text Analytics



# 1. Business Understanding

- Similar to data mining process, this phase should include the following:
  - Business objectives
  - Mining objectives
  - Project analysis (cost benefit analysis; risk assessment; resources needed; assumptions)
  - Project plan


Example:

- **Business objective**: to improve SPAM detection by 10%
- **Mining objective**: to generate a predictive model that will classify an email as either SPAM or NOT SPAM with at least 90% accuracy.

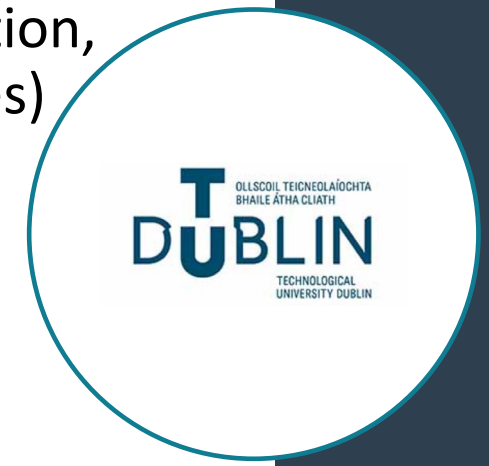
# 2. Data Understanding

1. Collect data: e.g., web scraping and related initial cleaning, such as eliminate html, javascript, etc.

2. Explore key concepts in the collection of documents:
   - Identify common/stop words to all topics.
   - Identify  predictive/unique words for each topic.
   - Identify phrases.
   - Identify synonyms and related terms.
   - Etc.

# 3. Data Preparation

1. Feature Generation:
   - Generate document or word vectors based on selected concepts.
2. Feature Reduction: based on the findings from the data exploration, we can reduce the number of features/attributes (words/phrases) before we generate the vectors:
   - Eliminate common words & stop words.
   - Replace synonyms and related terms.
   - Stemming or lemmatisation.
   - Etc.
3. Feature Selection:
   - Apply feature selection algorithms to retain only important/relevant features (words/phrases).



# 4. Mining

- Select the model: as with data mining, primarily focus on:
  - Classification
  - Clustering
- Generate test design:
  - Split Validation
  - Cross Validation
- Build the model and adjust parameters.
- Assess the model performance:
  - Evaluate model accuracy/precision/recall

# 5. Evaluation & Deployment

- Assessment of data mining results with respect to:
  - Business Objectives
  - Mining Objectives
  - Predictive features, etc.
- Project review
- Project deployment
  - Working with novel data: generate document vectors for new documents and run the model.

# Summary…

- Despite the complexity and ambiguity of the natural language, it is possible to extract structural (semantic, syntactic and morphological features) and contextual features (sentiment, etc.).

- The primary challenge is to determine if and where the signal is hiding within the noise.
  - This is done through the process of ***feature analysis***—determining which features, properties, or dimensions about our text best encode its meaning and its underlying structure.

# Core Packages Needed

- The Python Standard Library (3)
- NLTK
- NumPy
- Pandas
- Scikitlearn
- Matplotlib

- Common String Operations
- https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

# String Methods

- Apart from operations that apply to all sequences, strings allow additional ones.
- Most of these operations can be carried out either by calling it on the class str, or on actual string object.

str.upper('text'); 'text'.upper() ## both return a copy of the original string with all uppercase characters: TEXT

str.lower('Analysis of Text'); 'Analysis of Text'.lower() ## both return a copy of the original string with all lowercase characters: analysis of text

str.count('mining', 'in') ## returns how many times 'in' occurs in 'mining' (no overlapping): 2

'to be or not to be'.count('To') ## returns ???

'Jane'.startswith('j') ## returns False

str.endswith('sufixes', 'es') ## returns ???

# String Methods

str.isalpha('text'); ## returns true if all characters are alphabetic, so ??

'Analysis of Text 23'.isalnum() ## returns true if all characters are alphanumeric, so ??

str.isdigit('m234') ## returns true if all characters are digits, so ??

str.replace('sufixes', 'es', '') ## replaces all instances of 'es' in 'sufixes' with empty string, so?

'deer'.replace('ee', 'ea') ## ??

str.split('1,2,3', ',') ## Return a list of the words in the string, using *sep* as the delimiter string: ['1', '2', '3']

'1,2,,3,,4,5'.split(',') ##???

' xyz '.strip() ##Return a copy of the string with the leading and trailing characters removed: 'xyz'

str.strip(' xyz xyz') ##???

- REGULAR EXPRESSIONS
- https://docs.python.org/3/library/re.html

# Regular Expressions

- **Regex** = a sequence of characters (strings) that defines a specific search pattern that can be used to manipulate and search text-based data
- https://docs.python.org/3/library/re.html (Import **re** module/package)
- Functions typically used:

re.search(pattern, string) – scans the input *string* and returns only the first match of the *pattern;* it returns None if no match is found; it tries to locate a first match anywhere in the string.

re.match(pattern, string) – scans the input *string* only at the beginning for a match of the *pattern;* it returns None if no match is found; it does not look for a match anywhere in the string.

**re.findall(pattern, string)** – scans the input *string* and returns all non-overlapping matches of the *pattern;* it returns None if no match is found.

**re.sub(pattern, repl, string)** – scans the input *string* for any matches of the *pattern* and replaces them with the *repl* string; it returns the string with replacements; if no matches are found will return the unchanged string.

# Regular Expressions

- **re.split(pattern, string)** – scans the input *string* for matches of the *pattern* and it splits it there*;* it returns a list of all the substrings where splitting occurred.

- Basic Patterns:

**ing** matches the characters *ing* exactly in that order

**34** matches the characters *34* exactly in that order

**ss|tt|ed** matches either *ss* OR *tt* OR *ed* (disjunction)

- Character classes and ranges:

**[abc]** matches either **a, b,** or **c**

**[^abc]** matches any character EXCEPT **a, b,** or **c** (negation)

**[a-zA-D]** matches any character in the range **a** through to **z**, or **A** through to **D**, inclusive

# Regular Expressions

- Let’s apply the patterns:

text = “There are 343 attributes that we are extracting from the 434 downloaded assets in an interesting and assessable way.”

- Let’s find all matches:

```
re.findall(r'ing', text)      ## returns a list of 2 instances of ‘ing’
re.findall(r'34', text)       ## returns a list of 2 instances of ’34’
re.findall(r'ss|tt|ed', text)  ## returns a list of 1 instance of ‘tt’,  1
                                ## instance of ‘ed’, and 3 instances of ‘ss’
```

- Now let’s substitute characters from ranges:

```
re.sub(r'[io]', 'e', text) ## 'There are 343 attrebutes that we are extracteng
frem the 434 dewnleaded assets en an enteresteng and assessable way.''
re.sub(r'[w-z]', '?', text) ## 'There are 343 attributes that ?e are e?tracting
from the 434 do?nloaded assets in an interesting and assessable ?a?.'
```

# Regular Expressions

- Your turn… Using the same string try and work it out without the interpreter, then test your solution in the interpreter:

text = “There are 343 attributes that we are extracting from the 434 downloaded assets in an interesting and assessable way.”

re.findall(r'[a-z]re', text)

re.findall(r’[0-9]', text)

re.sub(r'[^aeiou]d', '*', text)

# Regular Expressions

- Pre-defined character classes:

| | |
|---|---|
| **.** | matches any character |
| **\d** | matches any digit (equivalent to [0-9]) |
| **\D** | matches any non-digit character (equivalent to [^0-9]) |
| **\s** | matches any whitespace character (equivalent to [\t\n\x0**B**\f\r]) |
| **\S** | matches any non-whitespace character (equivalent to [^\t\n\x0**B**\f\r] or [^\s]) |
| **\w** | matches any word character (equivalent to [a-zA-Z_0-9]) |
| **\W** | matches any non-word character (equivalent to [^a-zA-Z_0-9] or [^\w]) |

**NOTE:** Unless preceded by a backslash, the following characters are treated as meta characters : ( [ { \ ^ - $ | ] } ) ? * + .

# Regular Expressions

- Let's apply classes to the same input string:

text = "There are 343 attributes that we are extracting from the 434 downloaded assets in an interesting and assessable way."

re.findall(r'\wx\w', text) ## returns ['ext']

re.findall(r'\d', text) ## returns ['3', '4', '3', '4', '3', '4']

re.findall(r'[^aeiou\s]d', text) ## ???

re.sub(r'\d', '_', text) ## ???

# Regular Expressions

**Greedy Quantifiers (they match as much text as possible):**

**?** matches 0 or 1 repetitions

Example: *ab?* matches either a or ab, but not abb, because it looks for **a** followed by 0 or 1 **b**

***** matches 0 or more repetitions:

Example: *ab** matches a or ab, or abb, or abbbbbbbbb because it looks for **a** followed by 0 or more **b**'s

**+** matches 1 or more repetitions

Example: *ab+* matches ab, or abb, or abbbbbbbbb, but not a because it looks for **a** followed by at least one occurrence of **b**

**{n}** matches exactly n repetitions

Example: *ab{2}* matches abb, but not a, or ab or abbb, because it looks for **a** followed by exactly 2 **b**'s

# Regular Expressions

**Greedy Quantifiers (continued…):**

**{n,}** matches n or more repetitions

Example: *ab{2,}* matches abb or abbbbbbbbb, but not a, or ab because it looks for **a** followed by at least two **b**'s

**{n,m}** matches any n to m repetitions

Example: *ab{2, 4}* matches abb or abbb, or abbbb, but not a, or ab or abbbbb because it looks for **a** followed by at least 2 but no more than 4 **b**'s

- Let's apply quantifier patterns to the input string:

s = "abb ab a aa abbb ababab abbbabbabbab abbbbbbbbbb abbbb rabbit"

re.findall(r'ab?', s) ## ['ab', 'ab', 'a', 'a', 'a', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab']

re.findall(r'ab*', s)## ['abb', 'ab', 'a', 'a', 'a', 'abbb', 'ab', 'ab', 'ab', 'abbb', 'abb', 'abb', 'ab', 'abbbbbbbbbb', 'abbbb', 'abb']

# Regular Expressions

- Your turn: work the solution without the interpreter then  test it in your IDE:

```
s = “abb ab a aa abbb ababab abbbabbabbab abbbbbbbbbb abbbb rabbit”

re.findall(r'ab+', s)

re.findall(r'ab{2}', s)

re.findall(r'ab{2,}', s)

re.findall(r'ab{2,4}', s)
```

# Regular Expressions

Non-Greedy Quantifiers (they match as few characters as possible): they are formed by adding an extra question mark

**??** *ab??* matches only a because it looks for the shortest matches that meet the requirements of the pattern: *a followed by no b or one b* (so a is shorter than ab)

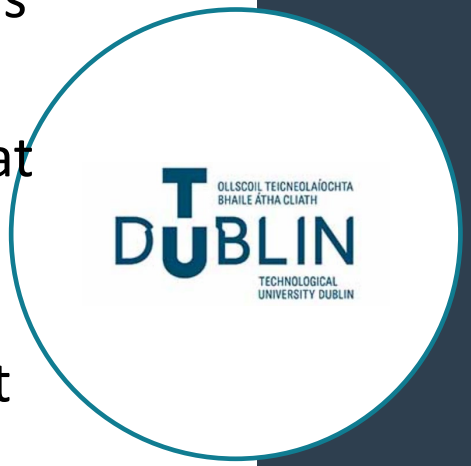
***?** *ab*?* matches only a because it looks for the shortest matches that meet the requirements of the pattern: *a followed by no b or one b, or more b's* (so a is shorter than ab or abb, etc.)

**+?** *ab+?* matches only ab because it looks for the shortest match that meets the requirements of the pattern *a followed by at least one b* (so ab is shorter than abb or abbb, etc.)

**{n}?** *ab{2}? m*atches only abb → no difference from ab{2}

**{n,}?** *ab{2}? m*atches only abb

**{n,m}?** matches any n repetitions: *ab{2, 4}* matches only abb



# Some Exercises on non-greedy quantifiers…

- Let's apply non-greedy quantifier patterns to the input string:

s = "abb ab a aa abbb ababab abbbabbabbab abbbbbbbbbb abbbb rabbit"

```
re.findall(r'ab??', s)
## ['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a']
re.findall(r'ab*?', s)
## ['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a']
re.findall(r'ab+?', s)
## ['ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab', 'ab']
re.findall(r'ab{2}?', s)
## ['abb', 'abb', 'abb', 'abb', 'abb', 'abb', 'abb', 'abb']
re.findall(r'ab{2,4}?', s)
## ['abb', 'abb', 'abb', 'abb', 'abb', 'abb', 'abb', 'abb']
```

# Regular Expressions

Boundary Matchers:

s = "ingestion is the process of taking food, drink, or another substance into the body by chewing, swallowing and/or absorbing“

**^** matches the beginning of string; for example: *^ing* matches *ing* at the beginning of string (do not confuse it with the negation inside a range: ‘[^ing]’)

```
re.findall(r'^ing\w*', s) ## matches ‘ingestion’
```

**$** matches the end of string: *$ing* matches *ing* at the end of string

```
re.findall(r' \w*ing$ ', s) ## matches ‘absorbing’
```

**\b** word boundary: matches the empty string right before or right after of a word

```
re.findall(r’\b\w*ing\b', s) ## matches ???????
```


Useful Flags:

- re.**A** or re.**ASCII:** Make \w, \W, \b, \B, \d, \D, \s and \S perform ASCII-only matching instead of full Unicode matching. This is only meaningful for Unicode patterns and it is ignored for byte patterns.
- re.**I or** re.**IGNORECASE:** Perform case-insensitive matching.

# Finding Patterns with NLTK

NLTK version of findall

- Allows us to look for particular language patterns: words beginning with ‘im’, as X as Y, same as X, etc.
- Uses the **<>** to mark word boundaries
- It ignores white spaces between the angle brackets
- It must be called on an instance of nltk.Text object
- The nltk.Text constructor takes a list of strings
- Example: search for modifiers used to modify ‘dinner’

```
text1 = nltk.Text(gutenberg.words('austen-emma.txt’))
text1.findall(r"<an*> <\w*> <dinner>")
##  a late dinner; a plentiful dinner; a late dinner
```

NOTE: you must import the nltk module and the gutenberg resource

EXERCISE: find all the words in emma that end with ‘est’; what do you observe?

# Scikit-learn … python library for learning

- An open-source library built on top of numpy, scipy and matplotlib (https://scikit-learn.org/stable/documentation.html )
- Support for pandas data frames.
- Some functionalities on the next few slides… (note: there are many others…)
- Cleaning and pre-processing functionalities (can be used as alternatives to the ones we have covered): https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing
  - Scaling of features
  - Normalization to have unit norm
  - Handling of missing values
  - Generating higher order features
  - Build custom transformations, etc.

# Scikit-learn … python library for learning

## Feature extraction from text specifically:

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.multiclass

The `sklearn.feature_extraction.text` submodule gathers utilities to build feature vectors from text documents.

| | |
|---|---|
| `feature_extraction.text.CountVectorizer` ([…]) | Convert a collection of text documents to a matrix of token counts |
| `feature_extraction.text.HashingVectorizer` ([…]) | Convert a collection of text documents to a matrix of token occurrences |
| `feature_extraction.text.TfidfTransformer` ([…]) | Transform a count matrix to a normalized tf or tf-idf representation |
| `feature_extraction.text.TfidfVectorizer` ([…]) | Convert a collection of raw documents to a matrix of TF-IDF features. |

## Model Selection and Validation:

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection

### Splitter Functions

| | |
|---|---|
| `model_selection.check_cv` ([cv, y, classifier]) | Input checker utility for building a cross-validator |
| `model_selection.train_test_split` (*arrays, …) | Split arrays or matrices into random train and test subsets |

### Model validation

| | |
|---|---|
| `model_selection.cross_validate` (estimator, X) | Evaluate metric(s) by cross-validation and also record fit/score times. |
| `model_selection.cross_val_predict` (estimator, X) | Generate cross-validated estimates for each input data point |
| `model_selection.cross_val_score` (estimator, X) | Evaluate a score by cross-validation |

# Scikit-learn … python library for learning

Clustering algorithms: https://scikit-learn.org/stable/modules/clustering.html#clustering

| Method name | Parameters | Scalability | Usecase | Geometry (metric used) |
|---|---|---|---|---|
| K-Means | number of clusters | Very large `n_samples`, medium `n_clusters` with MiniBatch code | General-purpose, even cluster size, flat geometry, not too many clusters | Distances between points |
| Agglomerative clustering | number of clusters, linkage type, distance | Large `n_samples` and `n_clusters` | Many clusters, possibly connectivity constraints, non Euclidean distances | Any pairwise distance |

Evaluation of clustering: https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics

| | |
|---|---|
| `metrics.davies_bouldin_score` (X, labels) | Computes the Davies-Bouldin score. |
| `metrics.completeness_score` (labels_true, …) | Completeness metric of a cluster labeling given a ground truth. |
| `metrics.homogeneity_completeness_v_measure` (…) | Compute the homogeneity and completeness and V-Measure scores at once. |
| `metrics.homogeneity_score` (labels_true, …) | Homogeneity metric of a cluster labeling given a ground truth. |

# Scikit-learn … python library for learning

Classification algorithms:

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.multiclass

- sklearn.tree.DecisionTreeClassifier
- sklearn.neighbors.KNeighborsClassifier
- sklearn.neural_network.MLPClassifier
- sklearn.neighbors.RadiusNeighborsClassifier
- sklearn.ensemble.RandomForestClassifier
- sklearn.svm.LinearSVC
- sklearn.svm.SVC
- sklearn.linear_model.Perceptron
- sklearn.ensemble.GradientBoostingClassifier
- sklearn.linear_model.LogisticRegression
- sklearn.naive_bayes
- Etc.

# Scikit-learn … python library for learning

Evaluation of classification:

https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics

| | |
|---|---|
| `metrics.accuracy_score` (y_true, y_pred[, …]) | Accuracy classification score. |
| `metrics.auc` (x, y[, reorder]) | Compute Area Under the Curve (AUC) using the trapezoidal rule |
| `metrics.average_precision_score` (y_true, y_score) | Compute average precision (AP) from prediction scores |
| `metrics.balanced_accuracy_score` (y_true, y_pred) | Compute the balanced accuracy |
| `metrics.brier_score_loss` (y_true, y_prob[, …]) | Compute the Brier score. |
| `metrics.classification_report` (y_true, y_pred) | Build a text report showing the main classification metrics |
| `metrics.cohen_kappa_score` (y1, y2[, labels, …]) | Cohen's kappa: a statistic that measures inter-annotator agreement. |
| `metrics.confusion_matrix` (y_true, y_pred[, …]) | Compute confusion matrix to evaluate the accuracy of a classification |
| `metrics.f1_score` (y_true, y_pred[, labels, …]) | Compute the F1 score, also known as balanced F-score or F-measure |
| `metrics.fbeta_score` (y_true, y_pred, beta[, …]) | Compute the F-beta score |
| `metrics.hamming_loss` (y_true, y_pred[, …]) | Compute the average Hamming loss. |
| `metrics.hinge_loss` (y_true, pred_decision[, …]) | Average hinge loss (non-regularized) |
| `metrics.jaccard_similarity_score` (y_true, y_pred) | Jaccard similarity coefficient score |
| `metrics.log_loss` (y_true, y_pred[, eps, …]) | Log loss, aka logistic loss or cross-entropy loss. |
| `metrics.matthews_corrcoef` (y_true, y_pred[, …]) | Compute the Matthews correlation coefficient (MCC) |
| `metrics.precision_recall_curve` (y_true, …) | Compute precision-recall pairs for different probability thresholds |
| `metrics.precision_recall_fscore_support` (…) | Compute precision, recall, F-measure and support for each class |
| `metrics.precision_score` (y_true, y_pred[, …]) | Compute the precision |
| `metrics.recall_score` (y_true, y_pred[, …]) | Compute the recall |
| `metrics.roc_auc_score` (y_true, y_score[, …]) | Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores. |
| `metrics.roc_curve` (y_true, y_score[, …]) | Compute Receiver operating characteristic (ROC) |
| `metrics.zero_one_loss` (y_true, y_pred[, …]) | Zero-one classification loss. |