# Natural Language Processing (NLP)

```
 # Feature extraction from text
# Method: bag of words

from sklearn.feature_extraction.text import CountVectorizer

corpus = [
'Once upon a time there was a brave knight called George.',
'George had lots of adventures as he travelled by horse across many lands.',
'One day he came to a small village and met a man who lived in a cave next to the village.',
]

vectorizer = CountVectorizer()
print( vectorizer.fit_transform(corpus).todense() )
print( vectorizer.vocabulary_ )
```

```
[[0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0]
 [1 1 0 1 0 1 0 0 0 0 1 1 1 1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0 1 1 1 0 0 2 0 0 2 0 1]]
{'once': 24, 'upon': 32, 'time': 29, 'there': 28, 'was': 34, 'brave': 4, 'knight': 15, 'called': 6,
'george': 10, 'had': 11, 'lots': 18, 'of': 23, 'adventures': 1, 'as': 3, 'he': 12, 'travelled': 31,
'by': 5, 'horse': 13, 'across': 0, 'many': 20, 'lands': 16, 'one': 25, 'day': 9, 'came': 7, 'to': 30,
'small': 26, 'village': 33, 'and': 2, 'met': 21, 'man': 19, 'who': 35, 'lived': 17, 'in': 14, 'cave':
8, 'next': 22, 'the': 27}
```

```
# Store url
url = 'https://en.wikipedia.org/wiki/Natural_language_processing'

# Import `requests`
import requests

# Make the request and check object type
r = requests.get(url)
type(r)

requests.models.Response

# Extract HTML from Response object and print
html = r.text
print(html)
```

# SPELL Checking

# SPELL Checking

- A Spell checker is one of the basic tools required for language processing.
- Spell checking involves
  - identifying **words** and **non words**
  - also suggesting the possible alternatives for its correction.
- used in
  - Word processing
  - Character or text recognition
  - Speech recognition and generation.

# SPELL Checking

- Most available spell checkers focus on processing **isolated** words and do not take into account the context.
  - "Henry **sat** on the box"
  - "Henry **at** on the box"

# Spelling Errors

**Three** cause of error are:

- **Insertion:** Insertion of extra letter while typing. E.g. maximum typed as *max**ii**mum.*
- **Deletion:** A case of a letter missing or not typed in a word. E.g. *netwrk* instead of netw**o**rk.
- **Substitution:** Typing of a letter in place of the correct one. E.g. *inte**ll**ugence.*

Spelling errors may be classified into following types:

- **Typographic errors:**
  - Cause due to mistakes committed while **typing.**
  - E.g. netwrk in place of network.
- **Orthographic errors:**
  - Result due to a lack of **comprehension** of the concerned language on part of user.
  - E.g. arithmatic, wellcome, accomodation.
- **Phonetic errors:**
  - result due to poor cognition on part of **listener.**
  - E.g. the word **rough** could be spelt as **ruff.**
  - Listen as lisen, piece as peace or peas, reed as read.

# Spell checking techniques

Spell checking techniques can be broadly classified into **three** categories-

**(a) Non-Word** error detection:

**(b) Isolated-word** error correction:

**(c) Context dependent** error detection and correction:

# Spell checking techniques

(a) **Non-Word error detection:** This process involves the detection of misspelled words or non-words.

- E.g. the word sopar is a non-word ; its correct form is super or sober.
- The most commonly used techniques to detect such errors are the
  - **N-gram analysis**
  - **Dictionary look-up.**

# Spell checking techniques

**N-Gram Analysis:**

- Make use of probabilities of occurrence of N-grams in a large corpus of text to decide on the error in the word.
- N-gram to be sequence of letters rather than words.
- Try to predict next letter rather than next words.
- Used in text (handwritten or printed) recognition.

# Spell checking techniques

**Dictionary look-up**

involves the use of an efficient dictionary lookup coupled with pattern-matching algorithm (such as hashing technique, Finite state automata), dictionary portioning schemes and morphological processing methods.

# Spell checking techniques

(b) **Isolated-word error correction:**

- Focus on the correction of an isolated non-words by finding its nearest and meaningful word and make an attempt to rectify the error.
- It thus transform the word “soper” into super.
- Isolated word correction may be looked upon as a combination of three sub-problems

  *Error detection,*

  *candidate (correct word) generation,*

  *ranking of correct candidates.*

Minimum Edit distance technique:

- Wanger[1974] define the minimum edit distance between the misspelled word and the possible correct candidate
- minimum number of edit operations needed to transform the misspelled word to the correct candidate.
- Edit operation-insertion, deletion, and substitution of a single character.
- The minimum number of such operations required to affect the transformation is commonly known as ***Levenshtein distance.***

## (c) Context dependent error detection and correction:

- In addition to detect errors, try to find whether the corrected word fits in to context of the sentence.
- More complex to implement.
- “**Peace** comes from within”, “**Piece** comes from within” ; **first** word in both sentence is a correct word.
- This involves correction of real-word errors or those that result in another valid error.

# Soundex

- Class of heuristics to expand a query into phonetic equivalents
  - Language specific – mainly for names
  - E.g., ***chebyshev*** $\rightarrow$ ***tchebycheff***
- Invented for the U.S. census … in 1918
- It is used to find similar words for replacement in error correction (torn, turn have same code T650)

# Soundex – typical algorithm

- Turn every token to be indexed into a 4-character reduced form
- Do the same with query terms
- Build and search an index on the reduced forms
  - (when the query calls for a soundex match)

# Soundex – typical algorithm

1. Retain the first letter of the word.
2. Change all occurrences of the following letters to '0' (zero):
   'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
3. Change letters to digits as follows:
   - B, F, P, V $\rightarrow$ 1
   - C, G, J, K, Q, S, X, Z $\rightarrow$ 2
   - D,T $\rightarrow$ 3
   - L $\rightarrow$ 4
   - M, N $\rightarrow$ 5
   - R $\rightarrow$ 6

# Soundex continued

4. Remove all pairs of consecutive digits.
5. Remove all zeros from the resulting string.
6. Pad the resulting string with trailing zeros and return the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

E.g., ***Herman*** becomes H655.

Will ***hermann*** generate the same code?

| Word | Soundex code |
| --- | --- |
| Grate, Great | |
| Network, network | |
| Henry, Henary | |
| Torn | |
| Worn | |
| Horn | |

- Soundex code for some words:

| Word | Soundex code |
|---|---|
| Grate, Great | G630 |
| Network, network | N362 |
| Henry, Henary | H560 |
| Torn | T650 |
| Worn | W650 |
| Horn | H650 |

- Used to measure similarity of two words.