# Homework 7: Unsupervised Learning: Kmeans. Lexical information

Benjamin Roth, Marina Sedinkina
Symbolische Programmiersprache

Due: Thursday December 21, 2017, 16:00

In this exercise you will:

- use NLTK to perform kmeans clustering
- use NLTK to analyze text and perform its morphological analysis

## Exercise 1: Kmeans [5 points]

On the course homepage, you can find the file `courses.txt`, containing several LMU courses of studies per line. Download it into the `data/` folder of your project. Use NLTK Kmeans clusterer to cluster LMU courses.

Take a look at `hw07_nltk_kmeans/nltk_kmean.py`. In this exercise you will have to implement some methods to perform the clustering.

This homework will be graded using unit tests by running: `python3 -m unittest -v hw07_nltk_kmeans/test_kmeans.py`

1. Implement the Reader class method `get_lines(self, labels)`. This method should return the list of courses from the file courses.txt with listed courses, i.e. one course per line.
2. Implement the Reader class method `normalized_word(self, word)`. This method should normalize the word by making it lower case and deleting punctuation marks from it. `Hint`: you can use set(string.punctuation)
3. Implement the Reader class method `get_vocabulary(self)`. This method should return vocabulary: the list of unique normalized words from file, sorted alphabetically. `Note:` words in vocabulary should be normalized, use `normalized_word(self, word)` to do this.

4. Implement the Reader class method `vectorspaced(self,course)`. This method should represent each course by one-hot vector: vector filled with 0s, except for a 1 at the position associated with the word in vocabulary. `Note`: the length of vector should be equal to the vocabulary size

5. Implement the KMeans class method `nltk_cluster(self,data)`. This method should use NLTK KMeansClusterer to cluster the data and return the list of clusters for given data.

6. Once you have implemented all missing functionality, you can have a look at `run_kmeans.py` to see how to use Kmeans in practice. Run the code with:

   ```
   python3 -m hw07_nltk_kmeans.run_kmeans
   ```

## Exercise 2: Lexical information and Morphological Analysis [10 points]

Take a look at `ex07/test_analyzer.py`. In this exercise you will have to implement some methods in class Analyzer, that can analyze any text from nltk.book.

This homework will be graded using unit tests by running:

```
python3 -m unittest -v hw07_nltk_kmeans/test_analyzer.py
```

Implement the following methods:

## Exercise 2.1: Lexical information

- numberOfTokens(self) – should return the number of tokens in the text
- vocabulary(self) – returns a list of the vocabulary of the text sorted alphabetically.
- vocabularySize(self) – returns the size of the vocabulary.
- lexicalRichness(self) – returns the lexical richness of the text.
- hapaxes(self) – returns all hapaxes of the text
- numberOfHapaxes(self) – returns the number of hapaxes in the text
- avWordLength(self) – returns the average word length of the text.

## Exercise 2.2: Morphological Analysis

An important problem in computational linguistics is morphological analysis. This consists of breaking down a word into its component pieces, for example *losses* might be broken down as *loss + es*. In English, morphology is relatively simple and is mostly comprised of prefixes and suffixes. To get an idea of what suffixes are common in English (and thus could be morphemes), we can look at the frequencies of the last $n$ characters of sufficiently long words.

Implement additional methods in class Analyzer, that can perform morphological analysis:

- topSuffixes (self) - returns top 10 most often seen suffixes of length 2. We define a n-character suffix as the last n characters of any word of length 5 or more, thus ignore any word shorter than five characters
- topPrefixes(self) - returns top 10 most often seen prefixes of length 2. We define a n-character prefix as the first n characters of any word of length 5 or more, thus ignore any word shorter than five characters
- tokensTypical(self) - returns first 5 tokens of the (alphabetically sorted) vocabulary that contain both often seen prefixes and often seen suffixes in the corpus