# Lexical ressources: Word2Vec, Exercise 1

Original document by Sacha Beniamine, translated and adapted by Timothee Mickus

## Word2vec embeddings visualisation with PCA

This exercise is about how to load a set of word2vec embeddings with gensim, and how to visualise them using Principal Component Analysis (PCA).

We are going to use three python libraries: sklearn, gensim and matplotlib.

- gensim is an efficient python library for dealing with non-contextual word embeddings (GloVe, Word2vec, FastText. . . )
- scikit-learn, or sklearn for short is a python lirbary dedidcated to statistical machine learning. It contains a number of very useful components for classification, regression, clustering, dimensionality reduction. . .
- matplotlib is a well-known python library for data visualisation. You should definitely ake some time to discover those libraries if you don't know about them: you will definitely find them useful in the future.

In this zip, you should find a file called `embeddings.w2v` and a python script named `pca_visualization.py`. The script is mostly empty: you will have to fill it in.

### Exercise 1.1 Load the word2vec vectors

The first function is called `load_embeddings()`. It takes as an argument the name of a file, and is meant to return a gensim word2vec model. To do that, you can use the `load_word2vec_format()` from gensim's KeyedVector class.

### Exercise 1.2 Select k neighbors for a word to plot

The second function you have to implement is called `select_neighbors()`. It takes two obligatory parameters `word` and `model`, and a default parameter `k`. You are to return the k most similar words to the string `word`, according to the gensim word2vec model `model`. To do that, you can use the function from gensim's model `model.most_similar()`

### Exercise 1.3 Reduce the dimensionality using PCA

The next function is called `compute_PCA()`, and takes as a parameter a list of words and a gensim model. It is expected to return a list of triplets `[(word, x, y)...]`, such that the coordinates after PCA of the vector for `word` are `x` and `y`. There several steps to go:

1. transform the words into vectors. The `model` from gensim allows you to use a simple bracketted syntax to do that: `vector = model[word]`. Make sure you keep track of which word corresponds to which vector.
2. create a PCA process object with sklearn. You can create an object representing this process using sklearn's `PCA` class. Note that you can specify how many dimensions should the output be using the keyword `n_component`.
3. apply PCA to the vectors you've extracted. You can do this using the `PCA.fit_transform()` method. This method expects the vectors that you have to transform as arguments.
4. match the expect return value of the function: you have to create triplets by matching together wods and PCA-reduced coordinates.

## Exercise 1.4 Transform coordinates into a scatter plot

This exercise requires that you fill in the `plot_PCA()` function. This function expects a list `coordinates` of triples `[(word, x, y)...]` as argument. You will need to go over each element of the `coordinates` list, and for each of them

1. define a point at coordinate (x,y) using the `matplotlib.pyplot.scatter()` function
2. annotate to what word the point(x,y) corresponds to using the `matplotlib.pyplot.text()` function Once all of this is done, you can create the plot and show it on screen using the `matplotlib.pyplot.show()` function.

## Exercise 1.5 Put everything together

As a convention, the general execution flow of a script should be executed in a conditional block `if __name__ == "__main__":`, like what you have at the bottom of the python script. This is where you should bind everything together, calling each function in order, and passing the output of the one to the other as a coherent fashion.

- Test your functions with various setups: change the word you use to extract neighbors in the `select_neighbors()` function; multiply the number of neighbors, etc.
- If you're feeling adventurous, you can also try to a command line arguments using the `argparse` standard python module: a good choice would be to include an argument specifying the word, the number of neighbors and the word2vec model to use.