# Coursework 1: Sentiment analysis from tweets

Due: Monday 20th November, 2023 (11.59pm GMT)

## Instructions

This assignment is marked out of 100 and counts for 40% towards your final grade for the module.

**Note:** This is an individual coursework. You must attempt the questions yourself and provide strong evidence for understanding your method in both your notebooks and your report.

**Objective:** In this coursework, you will implement a Support Vector Machine classifier (or SVM) that classifies tweets according to their sentiment, i.e. positive or negative. You will derive features from the text of the tweets to build and train the classifier on part of the dataset. You will then test the accuracy of your classifier on an unseen portion of the dataset. Much of the background for this part is in Lecture 2 and Lab 2 on Text Classification, though you should use all of your knowledge across the module.

**How to submit:** Follow the below instructions, and submit well documented code as **one or more IPython files** (.ipynb) building on the template file "NLP_Assignment_1.ipynb" as your starting point (Python 3.7+). You must **also submit a 2-page report** where you describe how you achieved each question briefly, with the relevant observations asked for below. You have the data in the file 'sentiment-dataset.tsv'. Ensure your code runs from top to bottom without errors before submission. If you do use more than one IPython file, it must be clear which file corresponds to which questions.

The template file contains some functions to load in the dataset, but there are some missing parts that you are going to fill in as per the questions below.

## Question 1 (10 marks)

**Simple data input and pre-processing.** Start by implementing the `parse_data_line()` and the `pre_process()` functions. Given a line of a tab-separated text file, `parse_data_line()` should return a tuple containing the label and text. The simple `pre_process()` function should turn a text (a string) into a list of tokens (words).

## Question 2 (20 marks)

**Simple feature extraction.** The next step is to implement the `to_feature_vector()` function. Given a preprocessed text (that is, a list of tokens), it will return a Python dictionary that has as its keys the tokens/words, and for the values a weight for each of those tokens in the preprocessed texts. The weight could simply be 1 for each word, or the number of occurrences of a token in the preprocessed text (i.e. a bag-of-words representation), or it could give more weight to specific words. While building up this feature vector, you may want to incrementally build up the global `global_feature_dict`, which should be a list or dictionary that keeps track of all the tokens/feature names which appear in the whole dataset. While a global feature dictionary is not strictly required for this coursework, it will help you understand which features (and how many!) you are using to train your classifier and can help understand possible performance issues you encounter on the way.

**Hint:** you can start by using binary feature values; 1 if the feature is present, by default the sklearn learn vectorization function will give it 0 if it's not.

## Question 3 (20 marks)

**Cross-validation on training data.** Using the `load_data()` function already present in the template file, you are now ready to process the texts for sentiment analysis. In order to train a good classifier, finish the implementation of the `cross_validate()` function to do a 10-fold cross validation on the training data (leave the test data split of 20% alone for now). Make use of the given functions `train_classifier()` and `predict_labels()` to do the cross-validation. Make sure that your program stores the precision, recall, f1 score, and accuracy of your classifier in a variable `cv_results`, which should contain average scores for all folds and be returned by this function.

**Hint:** the package `sklearn.metrics` contains many utilities for evaluation metrics - you could try precision, recall, fscore, support to start with.

## Question 4 (20 marks)

**Error analysis.** Look at the performance of the classes using a confusion matrix through the method provided `confusion_matrix_heatmap()` to see what the balance of false positives and false negatives is for the positive and negative labels. Carry out an error analysis on a simple train-test split of the training data (e.g. the first fold from your cross-validation function). For this you should print out (or better, print to file) all the false positives and false negatives for the positive label to try to understand why the classifier is not getting these correct and write in your report some observations and examples of where it is getting confused.

**Hint:** see Lab 2.

## Question 5 (30 marks)

**Optimising pre-processing and feature extraction.** Now that you have the numbers for accuracy of your classifier and have done some initial error analysis, think of ways to improve this performance score. Some ideas as to how to do this:

- Improve the preprocessing. Which tokens might you want to throw out or preserve?
- What about punctuation? Do not forget normalisation, lemmatising, stop word removal - what aspects of this might be useful?
- Think about the features: what could you use other than unigram tokens? It may be useful to look beyond single words to combinations of words or characters. Also the feature weighting scheme: what could you do other than using binary values?
- You could add extra stylistic features like the number of words per sentence.
- You could consider playing with the parameters of the SVM (cost parameter? per-class weighting?)
- You could do some feature selection, limiting the numbers of features through different controls on e.g. the vocabulary.
- You could use external resources like the opinion lexicon available at `https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon`.

Report what methods you tried and what the effect was on the classifier performance in your report and evidence the exploration in your notebook.