# Assignment 1: Text Classification

CSE 256: Statistical NLP: Spring 2022

University of California, San Diego

Due: April 11, 2022

In this programming assignment, you will build a `supervised` text classifier.
The particular type of text classification to be performed for this homework is `sentiment classification`. The objective is to classify the sentiment of user reviews into `negative` and `positive` reviews, and to make the task more interesting (i.e. difficult), we have split the reviews into segments of around 140 characters. Notice that this might make it prohibitively difficult or impossible detect the sentiment of some segments – this can be ignored for our purposes.

## 1 Preliminaries

### 1.1 Data and Code

The data you will use is in the tarball `sentiment.tar.gz`, which contains the following files:

- `train.tsv:` Reviews and their associated labels to be used for training.
- `dev.tsv:` Reviews and their associated labels, to be used for development (do not use for training).
- `unlabeled.tsv:` Reviews without labels, which you can (optionally) use for section 2.2.

We have made some initial code available for you to develop your submission. The file `classify.py` contains a basic logistic regression classifier from `sklearn`.
The file `sentiment.py` contains methods for loading the dataset (without untarring the data file), creating basic features, and calling the methods to train and evaluate the classifier.

## 2 Improve the Basic Classifier (12.5 points)

As described above, we have included a basic logistic regression classifier. You can train and evaluate it as follows:

```
python sentiment.py
```

The output should be:

```
Accuracy on train is:  0.9821038847664775
Accuracy on dev is:  0.777292576419214
```

This classifier uses the default hyperparameters for regularization and optimization from *sklearn* and uses the basic *CountVectorizer* to create the features. Your task is to familiarize yourself with the data and the classification task by improving this supervised text classifier.

### 2.1 Guided Feature Engineering(6 points)

One easy way to change the current implementation is to swap out the basic CountVectorizer, with a vectorizer that utilizes Term Frequency Inverse Document Frequency (TFIDF) weighing on lexical features, as discussed in class. This is achieved by using:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

`TfidfVectorizer` takes a parameter, $n$, which is the maximum $n$-gram feature length. For example, if we allow $n = 2$, bigrams such as "excellent food" will be features, in addition to

single word features "excellent". Experiment with different values of $n$. Note that the logistic regression implementation takes a regularization parameter, C, [1], so you can vary this as you vary $n$.
**In the writeup:** describe your observations from this experiment, does performance improve, if so, why? You should use examples, figures, tables, and graphs to illustrate the results, for example plotting accuracy as the regularization strength is varied for different values of $n$.

### 2.2 Independent Feature Engineering(6.5 points)

There are other changes you can make, for example using different tokenization other than the simple white-space tokenization provided, adding/removing features, and so on. The primary guideline here is to utilize only your intuitions and the training data, and the performance on the development data, in order to make these modifications. Use of unlabeled data or any other external resource is not allowed.

You are not allowed to use a different classifier other than logistic regression (and as much as possible, use the scikit-learn implementation). The aim of this exercise is to train you to do feature engineering not from changing the underlying classifier.

**In the writeup:** describe the changes, your reasoning behind them, and the results. You should use figures, examples, tables, and graphs to illustrate your ideas and results.

## 3 Submission Instructions

Submit your work on Gradescope.

- **Code:** You will submit your code together with a neatly written README file with instructions on how to run your code. We assume that you always follow good practice of coding (commenting, structuring), and these factors are not central to your grade.
- **Report:** (use the filename A1.pdf): your writeup should be **three pages long, or less, in pdf** ( reasonable font sizes).

  Part of the training we aim to give you in this class includes practice with technical writing. Organize your report as neatly as possible, and articulate your thoughts as clearly as possible. We prefer quality over quantity. Do not flood the report with tangential information such as low-level documentation of your code that belongs in code comments or the README. Similarly, when discussing the experimental results, do not copy and paste the entire system output directly to the report. Instead, create tables and figures to organize the experimental results.

## 4 Report Writing Guideline

### 4.1 Guided Feature Engineering

Performance should be better than the baseline

### 4.2 Independent Feature Engineering

Describe your feature engineering solution, provide results, discuss why it worked/did not work; use tables/graphs/examples

[1] See API docs `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`