ASSIGNMENT 1: PROFESSION PREDICTION

Motivation: The motivation of this assignment is to get you some practice with text categorization using Machine Learning algorithms.

Problem Statement: The goal of the assignment is to build a text categorization system for the profession of a person from a brief description of their life. The input of the code will be a description of the life of a person, and the output should be their profession.

Training Data: We are sharing a training dataset of profiles named *profiles.csv*. There are 2 columns in the csv file - *Profile* and *Profession*. There are 127855 rows in this csv file except the header. There are a total of 28 professions, that can be collected from the csv.

The Task: You need to write a classifier that, given a profile, predicts its class. In this Part, you must develop a <u>non-neural classifier</u>. Ideas include Naïve Bayes, logistic regression, nearest neighbor, rule/lexicon-based systems, SVM and so on.

As a baseline algorithm, you could use all words as features and learn a classifier (naïve Bayes or logistic regression). Before you begin, try to carefully think about the problem a little. To improve your system performance over the baseline, a few ideas to try are listed below.

1. Note that we are not releasing separate train and dev sets but only a single set. You may like to perform k-fold cross-validation (recommended but not mandatory). Read about k-fold validation yourself and ask queries over discussion forums if you have doubts. You may alternatively split the data into your own train and dev splits and train accordingly.

2. Since the data has class imbalance, you could try sampling with different frequencies for different classes.

3. Try regularizing in different ways, for example try L1/L2 regularization in Logistic regression, or change the regularization parameter in SVM..

4. Try to work with the features. You could lemmatize. You could get rid of stop words and highly infrequent words. You could use tfidf-based weighting.

5. You could work with bigrams or trigrams (in addition to unigrams).

6. You could define new features, like you could pos-tag each word and use the tagged word as a feature instead of the original word. You can use the presence of capitalization or all caps as features.

7. You are free to use any off-the-shelf tools such as NER etc. But before applying, think whether that tool could be helpful for the given problem or not.

8. However, do not use trained word embeddings, or use neural models for the task. By the same token, please don't use pre-trained neural models for a task (e.g., NER).

Methodology and Experiments:

As mentioned, it's up to you how to split the data into train and dev sets with a split ratio of your choice. You may like to perform k-fold cross-validation with k of your choice. Once you find the best hyperparameters, set them in your final training script to be submitted. Report the validation accuracy (or k-fold validation accuracy) achieved by you. We will be running your training code to replicate your reported results, so make sure you make your code deterministic (or not). We hold the test data with us which we are not releasing.

Most importantly, as you work on improving your baseline system, document its performance. Perform (statistical) error analysis on a subset of data and think about why the model is making these mistakes and what additional knowledge could help the classifier the most. That will guide you in picking the next feature (or model component) to add.

Note that this is a moderately large dataset, which will help you in honing your experimentation skills. Do smart grid searches, since plain grid search might be too expensive.

Evaluation Metric:

We will evaluate your output predictions using micro-F1 and macro-F1 scores. Read about these metrics. You can use a suitable aggregation (e.g. a weighted sum or average) of micro-F1 and macro-F1 for validation purposes. We will measure both of these on test data and give equal weightage to both while grading.

Test Format:

Your final program will take as input a set of profiles in the same csv format as training, with just 1 column, namely the *profile*. Your test script will output a file containing predicted professions ('professor', 'nurse', 'dentist') one in each new line – matching one prediction per profile. A sample submission has been given for example.

Submission Format:

The deadline for submission is **15th September 2022, 11:55 PM.**
Submit your code in a .zip file named in the format **<EntryNo>.zip.** Make sure that when we run "unzip yourfile.zip", a new directory is created with your entry number (in all caps). In that directory, the following files should be present. There can be more files for helper functions/utilities, but we are only concerned about the following:

- *requirements.txt*
- *train.sh*
- *test.sh*
- *writeup.txt*

Do not submit the dataset or trained model files in your submission. You will be penalized if your submission does not conform to the above requirements.

Your code will be run as :

- "pip install -r *requirements.txt*" (This should install necessary packages to setup your runtime environment.)
- "*./train.sh* <input path to *train.csv*> *trained_model* " (This should create a model file named *trained_model* in present directory)
- "*./test.sh trained_model* <input path to *test.csv*> *output.csv"* (This should create output predictions file named *output.csv* in present directory)

All the above commands will be run in an automated scripting fashion. Any error(s) will lead to a loss of 20% of the total credit. So make sure you conform to these requirements.

The *output.csv* should only contain a sequence of professions**,** one per line, with a total number of lines matching the number of profiles in *test.csv*.

The writeup.txt should have a first line that mentions names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone else say None.
After this first line you are welcome to write something about your code, though this is not necessary.

Runtime instructions:

Your training code must run on a single HPC CPU within 2 hours. Since this is a non-neural assignment, you are not supposed to use GPU but only a single CPU on HPC and make sure your training is complete within an hour. If your training time exceeds this limit, you will face a significant penalty.

Evaluation Criteria

(1) This part is worth 100 points.

(2) Bonus given to outstanding performers.

What is allowed? What is not?

1. The assignment is to be done individually.
2. **You must use only Python for this assignment.**
3. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class outside your team.** Please read academic integrity guidelines on the course home page and follow them carefully.
4. You are **not** allowed to use additional datasets for training your model.
5. Feel free to search the Web for papers or other websites describing how to build a text categorization system. However, you should not use (or read) other people's NLP classifier codes.
6. You are allowed to use any pre-existing ML softwares for your code (preprocessing, evaluation, k-fold etc.). Popular examples include Python Scikit (http://scikit-learn.org/stable/). You may use pre-trained non-neural models or off-the-shelf non-neural tools for tasks such as POS tagging, NER etc. However, if you include the other code, use a different directory and don't mix your code with pre-existing code. And you must not use existing text-categorization softwares.
7. We will run plagiarism detection software. Any team found guilty will be awarded a suitable penalty as per IIT rules.
8. Your code will be automatically evaluated. You get a significant penalty if it does not conform to given guidelines.