ASSIGNMENT 1: EMAIL CLASSIFICATION

Motivation: The motivation of this assignment is to get you some practice with text categorization using Machine Learning algorithms. This assignment will proceed in two (maybe three) parts. Each part is worth equal points.

Problem Statement: The goal of the assignment is to build a text categorization system for academic emails from annotated data. The input of the code will be a set of annotated emails sent to the Dean (Faculty) office at IIT Delhi from over five years ago.

Training Data: We are sharing a training dataset of emails named *emails.csv*. There are 3 columns in the csv file - *Subject*, *Content* and *Class*. There are 1868 rows in this csv file except the header. There are a total of seven classes, namely - *Meeting and appointment*, *For circulation*, *Selection committee issues*, *Policy clarification/setting*, *Recruitment related*, *Assessment related* and *Other*. You may call these labels 1, 2, 3, 4, 5, 6 and 7 in the respective order they are mentioned in the last sentence. Note that the most frequent class is *Other* i.e. label 7. You may use the *Subject* field as well for training and inference.

The Task: You need to write a classifier that, given an email, predicts its class. In this Part, you must develop a non-neural classifier. Ideas include Naïve Bayes, logistic regression, nearest neighbor, rule/lexicon-based systems 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, downweight each *Other* example or upweight data points from the important classes.

3. Try changing the classifiers. Try SVMs or random forests.

4. If you use logistic regression try playing with regularizers – try L1 instead of L2. You could also implement a different feature selection procedure.

5. 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.

6. Try to curate datasets using bootstrapping, so that words in a dataset reflect the typical themes in a given class.

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

8. 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.

9. 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.

10. 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). Please use non-neural systems only. In the next part of the assignment we will allow you to use neural models.

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.

Given that total amount of training data is small (which will often be the case in new applications), pay careful attention to overfitting. It will be easy to overfit and you will have to be creative to achieve good performance on the unseen test set.

Evaluation Metric:

We will evaluate your output predictions using micro-accuracy and macro-accuracy scores (over all seven classes). Read about these metrics. You can use a suitable aggregation (e.g. a weighted sum or average) of micro-accuracy and macro-accuracy 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 emails in the same csv format as training, but without the 3rd column named *Class*. So, you have to submit a test script which runs on a csv file having 2 columns only - *Subject & Content.* Your test script will output a file containing predicted labels (1,2,3,4,5,6 or 7) one in each new line – matching one prediction per email.

Submission Format:

The deadline for submission is **28th November 2020, 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.py*
- *test.py*
- *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.)
- "python *train.py* <input path to *train.csv*> *email_model* " (This should create a model file named *email_model* in present directory)
- "python *test.py email_model* <input path to *test.csv*> *outputfile.txt"* (This should create output predictions file named *outputfile.txt* 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 *outputfile.txt* should only contain a sequence of numbers (ranging from 1 to 7)**,** one per line, with a total number of lines matching the number of emails 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 one hour. 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. 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.
5. 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 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.
6. We will run plagiarism detection software. Any team found guilty will be awarded a suitable penalty as per IIT rules.
7. Your code will be automatically evaluated. You get a significant penalty if it does not conform to given guidelines.