Project Text Sentiment Classification

The task of this competition is to predict if a tweet message used to contain a positive :) or negative :(smiley, by considering only the remaining text.

As a baseline, we here provide sample code using word embeddings to build a text classifier system.

Submission system environment setup:

1. The dataset is available from the Kaggle page, as linked in the PDF project description

   Download the provided datasets `twitter-datasets.zip`.

2. To submit your solution to the online evaluation system, we require you to prepare a ".csv" file of the same structure as sampleSubmission.csv (the order of the predictions does not matter, but make sure the tweet ids and predictions match). Your submission is evaluated according to the classification error (number of misclassified tweets) of your predictions.

Working with Twitter data: We provide a large set of training tweets, one tweet per line. All tweets in the file train pos.txt (and the train pos full.txt counterpart) used to have positive smileys, those of train neg.txt used to have a negative smiley. Additionally, the file test data.txt contains 10'000 tweets without any labels, each line numbered by the tweet-id.

Your task is to predict the labels of these tweets, and upload the predictions to kaggle. Your submission file for the 10'000 tweets must be of the form `<tweet-id>`, `<prediction>`, see `sampleSubmission.csv`.

Note that all tweets have already been pre-processed so that all words (tokens) are separated by a single whitespace. Also, the smileys (labels) have been removed.

Classification using Word-Vectors

For building a good text classifier, it is crucial to find a good feature representation of the input text. Here we will start by using the word vectors (word embeddings) of each word in the given tweet. For simplicity of a first baseline, we will construct the feature representation of the entire text by simply averaging the word vectors.

Below is a solution pipeline with an evaluation step:

Generating Word Embeddings:

Load the training tweets given in `pos_train.txt`, `neg_train.txt` (or a suitable subset depending on RAM requirements), and construct a a vocabulary list of words appearing at least 5 times. This is done running the following commands. Note that the provided `cooc.py` script can take a few minutes to run, and displays the number of tweets processed.

```
bash build_vocab.sh cut_vocab.sh python3 pickle_vocab.py python3 cooc.py
```

Now given the co-occurrence matrix and the vocabulary, it is not hard to train GloVe word embeddings, that is to compute an embedding vector for wach word in the vocabulary. We suggest to implement SGD updates to train the matrix factorization, as in

```
glove_solution.py
```

Once you tested your system on the small set of 10% of all tweets, we suggest you run on the full datasets `pos_train_full.txt`, `neg_train_full.txt`

Building a Text Classifier:

1. Construct Features for the Training Texts: Load the training tweets and the built GloVe word embeddings. Using the word embeddings, construct a feature representation of each training tweet (by averaging the word vectors over all words of the tweet).
2. Train a Linear Classifier: Train a linear classifier (e.g. logistic regression or SVM) on your constructed features, using the scikit learn library, or your own code from the earlier labs. Recall that the labels indicate if a tweet used to contain a :) or :(smiley.
3. Prediction: Predict labels for all tweets in the test set.
4. Submission / Evaluation: Submit your predictions to kaggle, and verify the obtained misclassification error score. (You can also use a local separate validation set to get faster feedback on the accuracy of your system). Try to tune your system for best evaluation score.

Extensions:

Naturally, there are many ways to improve your solution, both in terms of accuracy and computation speed. More advanced techniques can be found in the recent literature.