UNIVERSITY OF BARCELONA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Assignment 1: Quora question pairs (Kaggle competition)

Student name: *Adrián Fernández Cid*

Course: *Natural Language Processing*
Due date: *April 12th, 2021*

1. Strategy

I took as baseline the model proposed in the assignment notebook, *i.e.* a logistic regression with a plain count-vectorised feature matrix as input. From there, I considered improving the pipeline at the preprocessing step by implementing a tf-idf vectorisation (which, as we have seen, adds an appropriate weight to the count of each feature/-word) and trying extra features, namely the Euclidean distance, the cosine similarity and the edit distance, all defined for the pair of questions of each sample.
At the level of the ML model, taking into account the limitations expected from a linear classifier such as a logistic regression (unable to draw correlations between features, and hence not even accounting for the parallelism between the features of each question in a sample), I decided to try an XGB classifier and, later on, a neural network as well.
With this in mind, the procedure followed in `train_models.ipynb` was:

1. Fit each of the two vectorisers considered on the train partition of the training/labelled dataset provided by Kaggle (to reproduce a real situation in which new data generally comes with new, unknown words).

2. Get the corresponding feature representations of train, validation and test partitions of the training dataset. Add, to each and one at a time, one of the three new features (Euclidean distance, cosine similarity and edit distance).

3. Train the logistic regression and the XGB classifier on every preprocessing combination and draw validation AUCs.

4. Pick the best pipeline for test. In addition, feed its preprocessing part to a neural network (added only at this step because they generally take longer to train) and compare its test AUC with the previous one.

2. Folder description

In the submitted `zip`, along with this report, you will find:

- `train_models.ipynb`. The main notebook, where all computations take place. As can be seen, it takes nearly 10 hours to run completely.

- `reproduce_results.ipynb`. Loads the models and feature matrices computed in the previous notebook to reproduce the validation and test AUCs. It also includes a comment on the mistakes of the two best models (the logistic regression and the neural network; see Results).
- `utils.py`. Contains the definitions of custom functions (along with their description) used in other notebooks, and the handmade tf-idf vectoriser class. All custom functions used in all the folder are defined here, with the exception of the cythonised edit distance and its implementation during vectorisation (as I could not find a way to import them from an external file). The most important are likely the mentioned class, `MyTfidfVectoriser`, and `get_features_from_df`, used to vectorise questions and optionally add one of the extra features.
- `results/`. Folder with intermediate results of the training, such as the instance of the custom tf-idf class (that takes around 45 min to fit) or the feature matrices.
- `models/`. Folder with the trained ML models.

To avoid submitting too heavy a folder, I have removed from the subfolders all quantities not necessary to `reproduce_results.ipynb`.

3. Issues while working on the challenge

Since we were told to concentrate on the preprocessing part, I first focused on building the custom tf-idf class. As a consequence of my previous lack of experience in building Python classes, this took most of my efforts, and the class is still remarkably inefficient w.r.t. its `sklearn` counterpart: although I managed to reproduce the latter's results (as shown in `train_models.ipynb`), my `fit` and `transform` methods take respectively about 45 min and 5h, in contrast with `sklearn`'s few seconds and 1 min, respectively. I handled this inconvenience by using the `sklearn` equivalent for validation and my own vectoriser for test, thereby producing genuine results while avoiding inefficient preprocessing. In any case, working on the class did allow me to fully grasp all the details of the tf-idf representation, and even get additional insight on the vectorisation process by enabling me to print out-of-vocabulary (OoV) words that came in each new dataset (validation and test).
Probably I could have optimised the custom tf-idf with cython, but my lack of expertise on the library and the fact that I had already spent much time on the class prompted me to move on. Another issue I faced, also related to cython, was importing the edit distance (the one you kindly provided) from an external file: I tried for some time to no avail, so I had to include its explicit definition in `train_models.ipynb`.
Further considering the edit distance, I encountered a warning while implementing it that resulted in an error when fitting the logistic regression. As I was not able to identify and correct the error, the corresponding AUCs are missing from the validation results table.
Finally, when checking a sample of the mistakes produced by the two best models (the logistic regression and the neural net; see Results), I noticed that some reference labels looked wrong, especially for the neural network. This could simply mean a poor labelling, but I suspect it could also be due to my handling of the input feature matrices to make them compatible with `tensorflow`: indeed, one cannot directly feed

a sparse matrix to the neural network, but it has to be transformed into a SparseTensor `tensorflow` object and then its indices must be reordered "into the canonical, row-major ordering"[1]. I initially assumed `tensorflow` would account for such a reordering, but I am no longer certain that is the case.

4. Results

The table below shows the **validation results** for all combinations considered:

Val. AUCs (%)	logistic regression		XGB classifier	
Added feature \Vectoriser	count	tf-idf	count	tf-idf
None (raw)	**80.58**	80.34	81.97	82.01
Euclidean dist.	80.58	80.34	81.97	82.01
cosine sim.	85.67	**87.00**	85.52	86.40
edit dist.	*	*	82.01	82.01

where "*" denotes the missing values corresponding to the edit distance issue referred to above and the baseline and best results are highlighted in bold.
It is apparent in the above values that the Euclidean distance does not help: this was expected, since it is sensitive to length difference in the strings and such a feature is not relevant for the task at hand, of a semantic character. The same applies to the edit distance[2] (bearing similar effects on the AUC –*i.e.*, none–), with the extra drawback that the latter also takes order into account: given that the syntactic structure of a question may vary while preserving its meaning, order-dependence is not desirable for the present purpose. On the contrary, the cosine similarity does improve results, as it is neither order nor length-dependent.
If one looks at the performances of the two classifiers, the expected improvement from a nonlinear algorithm is indeed present, although as soon as one explicitly adds a meaningful interaction feature (*i.e.* the cosine similarity above) that enables the logistic regression to take correlation into account it seems to perform slightly better than the XGB classifier.
Regarding the vectorisation, the improvement of tf-idf over the simple count manifests clearly when adding the cosine similarity, but is milder, non-existent or even "negative" in other cases. This was unexpected: I was hoping for the more powerful tf-idf to clearly surpass the count method. A possible explanation is that the differences in performance of various representations are somewhat diluted by the OoV words in the validation and test partitions (928 and 978, respectively).
The issue of OoV words affects as well the Euclidean distance and the cosine similarity: as shown in `reproduce_results.ipynb` when checking mistakes, there are likely to be pairs of questions where one contains a relevant new word that would tell both apart, but is not taken into account. It is worth noting that this is not an issue for the edit distance, as it is computed at string level and does not rely on a previously fitted vocabulary. In the light of all this, it would be interesting to explore more complete

[1] See the `tensorflow` documentation at https://www.tensorflow.org/api_docs/python/tf/sparse/reorder.
[2] One could decrease the cost of addition in the edit distance to diminish this, but its order-dependence is still inconvenient here.

vocabularies (such as predefined word embeddings) and also ways of adapting the vocabulary on the fly.
Finally, the validated pipeline is that of the tf-idf vectorisation with cosine similarity and the logistic regression, with a validation AUC of 87.00%. Feeding the same preprocessing to a neural network, I obtained a validation AUC of 87.65%. The **test AUCs** for these two combinations are:

Test AUCs (%)	
logistic regression	neural network
87.58	**88.06**

So the final winner seems to be a neural network (a nonlinear model) trained on a tf-idf matrix with the cosine similarity, with the *caveat* mentioned in issues regarding a possible index-reordering issue.

5. Conclusion

In this assignment, I have extensively explored and coded preprocessing techniques typical of the field of natural language processing, such as vector representations and string distances. These have been implemented in a full ML pipeline to solve a real, interesting problem, yielding the valuable insights (at least for a beginner) reflected in this report.
Further development of the project could explore weighting the classes (to account for the unbalance in data), adding other distance indicators, new vector representations (such as a pre-defined word embedding, like `word2vec`), dynamic vocabulary adaptation and more involved training or ML models. Any of these should be accompanied by code optimisation at the preprocessing stage, as it now takes too long.
Overall, despite the struggle (or perhaps because of it) I think this has been an interesting task, and I look forward to any feedback on code optimisation, code cleaning or the ML pipeline in general.