Report project5 CS412 (introduction to Machine Learning) by Piervincenzo Ventrella.

For the project I choose Task #3 which goal is **to develop a model able to predict** where in a question-anwsering plattoform **some questions posted by the user should be removed or not** from the platform by using the Quora Insincere Question Dataset.
The project follows this structure:

Dataset Inspection and preprocessing.

The Dataset includes more than 1.3 million of instances whose structure is **q_id| q_text|q_target.**
To develop the model, I choose to use a BagOfWord model and in doing so I tried different kind of preprocessing such as:

* Basic **BagOfWord** in which I just built a vocabulary taking the words from the raw text and putting the lower-case version in the vocabulary. Finally, I created and saved a new sparse Dataset in which each question is represented with the list of binary features indicating the presence or not of the corresponding words in the vocabulary (this can be found in the Preprocessing0.py file).
* BagOfWord model in which in building the vocabulary I performed **tokenization** (with regular expressions), **stemming** (with Porter's stemmer) and **punctuation removal**. This decision was made after noticing that the first vocabulary was full of elements that are not actually words and this increases by a lot the size. In doing so the size of the vocabulary decreases from 21359 to 10469 distinct words and this was not just useful for computational purposes, but it led also to better performances in all the models that I used for comparison in the next step (this can be found in the Preprocessing1.py file).
* I tried also to apply **StopWord Removal** and a BagOfWord model with **tf-idf** representation but these 2 preprocessing have not shown improvements in the performances of the models. (this can be found respectively in the Preprocessing2.py and Preprocessing3.py files).

An **important note** is: the BagOfWord models require a lot of computation and because of the fact the purpose of the project is not to obtain the highest possible performances but apply a valid process in developing a final Classifier I decided to consider just a relatively small subset of the original dataset. This allowed me to experiment different preprocessing and different models in a reasonable amount of time (a single preprocessing still requires several hours on a normal PC). Of course, in other circumstances considering only a small subset would be wrong.

First Baseline and Model's Comparison.

During this step I compared several classifiers from the **scikit-learn library** including (**Naïve Bayes, Logistic Regression, Ridge, Lasso, KNN, Decision tree, Random forest**) for all the different preprocessing. The first thing I noticed was that even with the simplest preprocessing the classifiers had a high accuracy on the test set (around 94%). For that reason, in comparing the models I focused also in the **Precision, Recall and F1Score** metrics that resulted low. This is explained from the fact that the dataset includes **only 6% of positive instances** and a classifier predicting everything as negative would have an accuracy of 0.94. Of course, this will not be satisfying because the purpose of the task is to correctly identify the positive instance.
For thit reason, I decided to perform **Oversampling** of the positive instances:
I performed **hold-out** to split the dataset in train and test, then I performed Oversampling on the train set and leave the original distribution of the data in the test set. Oversampling allowed to improve remarkably the values of Precision, Recall (and so F1) in all the models, the Accuracy slightly decreased (in some of them less than 1%).
I choose to perform Oversampling and not Subsampling to not reduce again the size of the dataset.
During the model's comparison I used also a **validation test** for hyperparameter tuning.

Final Comparison and Development.

In this step I choose to use the dataset from preprocessing1.py that seemed to be the most promising one for almost all the models including the 2 most promising ones (Logistic Regression and Ridge with alpha = 20). I tried also different **ensembles** (i.e. between the best ridge and logistic regression classifiers or including also the best Lasso classifiers but without having improvements in the performances so I decided to keep a simpler classifier).
Finally, I choose to implement the **Ridge classifier with alpha = 20** that showed the best performances (**Accuracy** varied between **0.93-0.94** and **F1** between **0.53-0.55**). The **variance** in performances of both classifiers was not significative.
Compared with the original **baselines** the **Accuracy remained the same** but the **F1Score** improved by (**+0.5** w.r.t. **MajorityVoting** classifier, **+0.25** from **NaiveBayes** baseline)
Of course, the results can be improved (consider for example that I used only a small percentage of the original dataset).
You can find a full **comparison table** in comparison.csw
A **limitation** of the BagOfWord models and the applied classifiers is that it does not take into account the semantic meaning of the words, but it just discriminates the examples according to the presence or not of the words in the vocabulary. It can happen that 2 questions with the same meaning, but different contained words are classified differently where the true labels were identical.
Moreover, what happen with words that in the real world are indicative for a question to be insincere and the pre-built vocabulary does not include these words? The classifier will ignore them and loose important information.
Finally, if the classifier finds a word to be indicative of insincere questions, we can expect that synonyms or strongly correlated words would be indicative as well, but this cannot be captured by the classifier.
As further work would be nice to experiment how using word embeddings improves (or not) the performances, possibly being able to capture the semantical meaning of the words and their similarity.