ECS763P/U Natural Language Processing Assignment 1 (worth 40% of your grade)

Fake News Detection

Background: Fake news is a pervasive problem in modern online life and having methods to automatically detect is becoming increasingly important. In this coursework, you will be building and evaluating a text classifier which detects fake news. You are provided with a dataset of over 10,000 statements on current affairs which have been annotated with labels for different degrees of fakeness, ranging from 'true' to 'pants on fire'. In this coursework we will simplify the task to a binary decision of **'real'** or **'fake'**.

Note this dataset is available publicly in various forums with existing models, but you will get marks for evidence that you have solved problems yourself rather than copied, and the quality of your investigation, given we are using traditional (not deep learning) models, and encourage close analysis of the data. **You must attempt the questions yourself and provide strong evidence for understanding your method in both your notebooks and your report.**

In this coursework, you will implement the discriminative classifier the Support Vector Machine classifier (or SVM) that classifies the statements as real or fake. You will use both the statement text and the additional features contained in the data set to build and train the classifier on part of the data set. You will then test the accuracy of your classifier on an unseen portion of the corpus. Much of the background for this part is in **Unit 2 and Unit 4 on Text Classification**, though you should use all of your knowledge across the module.

Instructions: Follow the below instructions, and submit WELL DOCUMENTED code as one or more IPython files (.ipynb) building on the template file NLP_Assignment_1.ipynb as your starting point (Python 3.7+). You must also submit a 2-page report where you describe how you achieved each question briefly, with the relevant observations asked for below. You have the data in the file fake_news.tsv. Ensure your code runs from top to bottom without errors before submission. If you do use more than one IPython file, it must be clear which file corresponds to which questions.

The template file contains some functions to load in the dataset, but there are some missing parts that you are going to fill in as per the questions below.

1. (10 points) Simple data input and pre-processing Start by implementing the *parse_data_line* and the *pre_process* functions. Given a line of a tab-separated text file, *parse_data_line* should return a tuple containing the *label* and *statement* column values, where you will have the label 'REAL' is for real news and 'FAKE' for fake news – to make this binary distinction from the multiple labels use the *convert_label* function provided and apply that to each label from the raw text. The simple *pre_process* function should turn a text (a string) into a list of tokens (words).

2. (20 points) Simple feature extraction The next step is to implement the *to_feature_vector* function. Given a preprocessed text (that is, a list of tokens), it will return a Python **dictionary** that has as its keys the tokens/words, and for the values a weight for each of those tokens in the preprocessed statements. The weight could be simply be 1 for each word, or the number of occurrences of a token in the preprocessed text (i.e. a bag-of-words representation), or it could give more weight to specific words. While building up this feature vector, you may want to incrementally build up the global *global_feature_dict*, which should be a list or dictionary that keeps track of all the tokens/feature names which appear in the whole dataset. While a global feature dictionary is not strictly required for

this coursework, it will help you understand which features (and how many!) you are using to train your classifier and can help understand possible performance issues you encounter on the way.

Hint: start by using binary feature values; 1 if the feature is present, by default the sklearn learn vectorization function will give it 0 if it's not.

3. (20 points) Cross-validation on training data Using the *load_data* function already present in the template file, you are now ready to process the fake news data from *fake_news.tsv*. In order to train a good classifier, finish the implementation of the *cross_validate* function to do a 10-fold cross validation on the *training data* (leave the *test data* split of 20% alone for now). Make use of the given functions *train_classifier* and *predict_labels* to do the cross-validation. Make sure that your program stores the precision, recall, f1 score, and accuracy of your classifier in a variable *cv_results*, which should be an average for all folds and be returned by this function.

Hint: the package sklearn.metrics contains many utilities for evaluation metrics - you could try precision, recall, fscore, support to start with.

4. (10 points) Error Analysis Look at the performance of the classes using a confusion matrix (through the method provided *confusion_matrix_heatmap* to see what the balance of false positives and false negatives is for the FAKE and REAL labels. Carry out an error analysis on a simple train-test split of the training data (e.g. the first fold from your cross-validation function). For this you should print out (or better, print to file) all the false positives and false negatives for the FAKE label to try to understand why the classifier is not getting these correct and write in your report some observations and examples of where it is getting confused. *Hint: see Lab for Unit 2.*

(Note for questions 5-6 you may want to start a new notebook, as you have to re-write funcitons you have already implemented):

5. (20 points) Optimising pre-processing and feature extraction. Now that you have the numbers for accuracy of your classifier and have done some initial error analysis, think of ways to improve this score. Some ideas as to how to do this:

- Improve the preprocessing. Which tokens might you want to throw out or preserve?
- What about punctuation? Do not forget normalisation, lemmatising, stop word removal - what aspects of this might be useful?
- Think about the features: what could you use other than unigram tokens? It may be useful to look beyond single words to combinations of words or characters. Also the feature weighting scheme: what could you do other than using binary values?
- You could add extra stylistic features like the number of words per sentence.
- You could consider playing with the parameters of the SVM (cost parameter? per-class weighting?)
- You could do some feature selection, limiting the numbers of features through different controls on e.g. the vocabulary.
- You could use external resources like publicly available lists of famous individuals or swear words to further add.

Report what methods you tried and what the effect was on the classifier performance in your report and evidence the exploration in your notebook.

6. (20 points) Using other metadata in the file Now look beyond textual features of the statement. The file provided contains a number of other features for each statement in the data file, which up till now you should not have used (*subject, speaker, speaker_job_title, state_info, party_affiliation,*

total_barely_true_counts, total_false_counts, total_half_true_counts, total_mostly_true_counts, total_pants_on_fire_counts). You have to modify the *load_data* function to get these into the system, in addition to the other functions you have already modified.

Experiment with using the values from these columns as additional features to optimize your classifier's performance and record the improvements in a results table in your report. Make it clear in your notebook how you have explored them and what improvements were caused by which features.

Note, even if you don't get as far as 5-6, make sure you run your best model on the test data (20%) as per the end of the template notebook, as this will have an effect on your model's performance mark.