Tips for Coursework 1

General tips:

- Make sure all notebooks run without error from top to bottom before submission and they are well documented so it's clear what you're doing. You will get marks for clarity.
- Make sure you describe your method in your own words in the accompanying 2-page report, being clear and succinct. There is no need to include code snippets in the report as this should be in your code.

1. Reading in/pre-processing

- You only need to modify `parse_data_line()` and `pre_process()` – there is no need for additional calls to loading in the file, changing its format or anything else. If you're unsure what you've achieved in those functions, use print statements of the object you're returning before the return statements.

2. Feature extraction

- You only need to return a dictionary locally created in the function. This dictionary represents the features/words present in the tokens representing a statement, which are passed to it in the parameter tokens (which are the result of your preprocessing output from question 1).
- The global variable `global_feature_vector` is not needed to complete the question or the assignment but you can make use it within this function if you'd like to keep track of how many features you have extracted over the whole corpus.
- Make sure you know exactly what is coming out of this function – after calling the `split_data_and_preprocess()` function in the main part of the code you could see what the first instance looks like by adding a line `print(train_data[0])`. For this question, these dictionaries should be small, with their keys corresponding to the word types seen in each statement. If they're very large, be careful that you're not creating a massive global dictionary with all possible words in the data and appropriate 0 counts – you don't need to do this as the vector creation is done for you by the LinearSVC classifier's fit function when called in the function `trainClassifier()` provided.

3. Cross validation

- Don't use an off-the-shelf cross-validation implementation, you have to fill in the code within the for-loop as requested to show you understand it.
- You can look ahead to the bottom of the notebook for how training is done on all the training data and testing is done on the test data – you will need to call those functions within the for loop, but over different proportions of the training data.
- In Lecture 2/beginning of Lecture 3 cross-validation is explained in some detail (slide 69 in Unit 2), as it is in Jurasky and Martin – you need to create training and test data for each fold, and train the classifier on the larger training part, and test on the smaller test part.
- Return the average scores obtained from each fold, at least in terms of precision, recall, f-score and accuracy (as per the metrics at the bottom of the notebook for final testing).

4. Error Analysis

- Note the `confusion_matrix_heatmap()` code does not run in everyone's version of Python, so you may need to change the line `metrics.confusion_matrix(y_test, preds, labels)` to `metrics.confusion_matrix(y_test, preds, labels=labels)` for it to run.
- You need to print out/print to file all the False Positives and False Negatives for the FAKE label, ideally not just in terms of the features extracted, but also showing the original raw text too. Make it clear which kind of error each one is. See the solutions to Week 5's lab on QMPlus for how this is done for sequence labelling tasks (as opposed to text classification on whole documents as you're doing here).
- Printing the errors to files rather than within the notebook is preferred to avoid massive notebooks which can cause you memory issues.
- Credit will be given for trying to analyse what is going on in terms of the error types, and observations about the number of each kind of error based on the confusion matrix or classification reports. The idea is you want to create a classifier with better pre-processing and feature extraction which allows it to classify fake news with greater accuracy, so orientate your observations around that aim. If you can do some quantitative analysis of why the errors are being made or data analysis, include it.

5. Improving the pre-processing and feature extraction

- Start a new notebook (the easiest is probably a copy of what you had for Q1-4).
- As you change them, make sure you know exactly what is coming out of your functions – don't ever let this be a mystery. Use print statements on individual samples of data to check.
- Remember as you try different techniques, that you should only expect relatively small, marginal improvements to the f-score/accuracy measures, and you should understand roughly why the improvements are happening over the baseline or your best system so far. If something's too good to be true, it probably is! Make sure you are never looking ahead to the test data/test part of folds in any way if possible, as you may get unfair gains.
- You should only report results in terms of the cross-validation results (not final testing as per the bottom of the page). We can see if you've been looking ahead to the final test data alone for development and will lose marks for that.
- You don't need to save all versions of the functions as you develop but make a note of exactly what you tried and its associated results.
- Try to be systematic – try each technique individually, and if you can, combinations of them, noting the results for those techniques as you go in your notebook, and taking a sample of these, perhaps the most interesting ones which led to improvement, for your report.
- A results table in the report would be recommended to show how you get different results with different systems.
- You can use external libraries and even different classifiers, but the vast majority of marks are for good systematic preprocessing and feature engineering and understanding of what you've done.