Assignment 1: Text Preprocessing and Classification(Due Feb 16, 2020, 23:59pm)

The goals of this assignment are to:

1. Get introduced to doing data preprocessing for NLP, which is an essential part of practicing NLP.
2. Implement NLP evaluation methods (precision, recall) by yourself in order to understand them better.
3. Get a feel for using prepackaged NLP libraries (`spacy`, in this case).
4. Learn to do experiments in NLP by starting from simpler baselines and applying more complex models as necessary, experimenting with a range of features and models using `sklearn` (a popular Python machine learning library).

The deliverables are:

1. **Your code**: implementing functions in the skeleton files `regex.py` `wiki_experiments.py`, `our_metrics.py` and `classify_frames.py`.
2. A **file called `predictions.tsv`**: In part 2, you will design and train a machine learning classification model. You will need to submit model's output for the test data, in file called `predictions.tsv`.

The materials provided in this zip file are the following. Note that the part after the `#` is an explanation of what each file is.

```
|-- classify_frames.py      # Skeleleton code for part 2
|-- our_metrics.py          # Skeleleton code for part 2
|-- raw_data
|  |-- GunViolence          # Data for part 2
|  |  |-- dev.tsv
|  |  |-- test.tsv
|  |  |-- train.tsv
|  |-- Wikipedia.xml        # Data for part 1
|-- regex.py
|-- wiki_experiments.py     # Skeleton code for part 1
|-- WikiExtractor.py        # Helper script for part 1
```

Notes regarding development environment setup

1. We will use Python3 in this assignment.
2. The following libraries will need to be installed in your Python environment: `spacy`, `pandas`, `sklearn`.
3. Additionally, you will need to run the command below to complete setting up `spacy`.

   python -m spacy download en

Part 1: Data preprocessing and understanding

In this part of the assignment, we will deal with processing and understanding Wikipedia textual data. Wikipedia is a very popular source of data for NLP research due to its size, relative quality, and multilinguality.

Provided for you is an `xml` file named `Wikipedia.xml`. It's contents are the current (English) versions of what were voted to be the most important Wikipedia articles, in the category of society and social sciences.

The file format is very specific to Wikipedia. And therefore, we have provided a script that can be used to extract "normal" text from such a Wikipedia "dump". The script itself can output files in two formats: `json` and again, `xml`. The skeleton code provided assumes that we will use the script to output `json` files.

However, in the sub-section below regarding regular expressions, *you will not use `WikiExtractor.py`*. You will experiment with extracting useful information using regular expressions yourself.

Regular expressions (15 points)

In this section, we'll experiment with regular expressions to extract pieces of texts that we are particularly interested in.

The python file `regex.py` contains code that will take your regular expressions, search for all non-overlapping occurrences of the expression in the `xml` file, print out examples, and finally print out the total counts.

1. As mentioned above, Wikipedia uses its own internal format. The format to link to a particular section of another Wikipedia page is as follows. Let's say we'd like to link to the "Criticisms" section of the "Economics" Wikipedia page, and we'd like our link to display the text "criticisms of Economics." This would be Wikipedia's format:

   ```
   [[Economics#Criticisms|criticisms of Economics]]
   ```

   Note that the part including and after the pipe character(`|`) is optional.

   Write a regular expression to find all links like the above(ie, links to sections on other Wikipedia pages). Note that we are not interested in links to other Wikipedia pages that don't specify a section.

2. Similarly, here is an example citation in the Wikipedia format.

   ```
   {{cite journal|last1=Benschop|first1=H. P.|
   last2=van der Schans|first2=G. P. |last3=Noort |
   first3=D. |last4=Fidder|first4=A.|last5=Mars-Groenendijk|
   first5=R. H.|last6=de Jong| first6=L. P. A.|
   title=Verification of Exposure to Sulfur Mustard in
   Two Casualties of the Iran-Iraq Conflict|
   journal=Journal of Analytical Toxicology|volume=21|
   issue=4|year=1997|pages=249-251|issn=0146-4760|
   doi=10.1093/jat/21.4.249|pmid=9248939}}
   ```

   Note the `doi=` at the last line. DOIs are universal addresses that are given to journal articles, government documents, ...etc.

   Write a regular expression to find all citations with DOI numbers.

3. Finally, write a regular expression to find all subheadings. In the wikipedia format, a subheading is written as below.

   `== Criticisms ==`

   Or

   `=== Criticism from the classical school of Economics ===`

   Where there are at least two `=` signs, and the number of `=` indicates the "subheading level". We intend to find *all* levels of subheadings.

Using `Spacy`, an NLP library (20 points)

In this subsection, we'll get used to using `spacy` an NLP library to do data preprocessing. `Spacy` is a popular NLP package that simplifies doing common NLP tasks such as tokenization, parsing, tagging, and even named entity recognition.

We will use `spacy` to count the number of tokens and lemmas in the dump, as well as to extract named entities from the Wikipedia articles.

Your first task is to use `WikiExtractor.py` to extract "normal" text from the Wikipedia dump. Below is the command that we will use to do that.

```
python WikiExtractor.py --json raw_data/Society_and_social_sciences.xml
```

This will create a directory structure `text/AA/`, and within that, files named `wiki_00`, `wiki_01`, ... etc. Note that each line within each file contains a JSON document corresponding to one article.

Note that the skeleton code is designed to work with the output of the script as described above, and takes care of reading the files one at a time(take a look at `yield_all_articles()` within `wiki_experiments.py`.)

Complete the code in the file `wiki_experiments.py` under the function `count_things()`, and print:

1. The total number of tokens that are not spaces or punctuations.
2. The total number of lemmas.
3. A count of the named entities to be found in the articles categorized by entity type.

You'll have to read of `spacy` documentation for this part. The following pages should be particularly helpful:

1. https://spacy.io/usage/processing-pipelines
2. https://spacy.io/usage/linguistic-features#named-entities

Part 2: Classification

In the `raw_data/` directory is a file named `GunViolenceData.tsv`. It contains a dataset annotated by experts in the field of communications. Each expert was shown news headlines from US media regarding US gun violence, and tasked to pick the top two "frames" that were present in the headline(along with some other questions, but we're not interested in those in this assignment).

In the field of Communications, "framing" is defined as "[selecting] some aspects of a perceived reality and make them more salient in a communicating text."[1]

The skeleton code provided within `classify_frames.py` takes care of reading the the `tsv` file and extracting the headlines and the first frame that the annotates picked as being present in the headline(look at `load_data_file(data_file)`).

Naturally, frame detection can be viewed as a text classification task, where the text are the news headlines and the classes/categories are the frames. We will take this approach to try and predict the news headlines.

[1] Robert M Entman. 1993. Framing: Toward clarification of a fractured paradigm. *Journal of communication*, 43(4):51-58.

Evaluation functions (15 points)

We will implement the precision and recall values we discussed in class. Here are the function signatures in the skeleton code `our_metrics.py` that you will complete.

- `precision(y_pred, y_true, average, labels)`
- `recall(y_pred, y_true, average, labels)`

where:

- `y_pred` is the list of predicted labels from a classifier.
- `y_true` is the list of gold labels(true labels).
- `average` indicates whether we are using the `macro` average, or the `micro` average(look at the slides again to remember what these mean).
- `labels` is a list of the possible values that `y_pred` or `y_true` can take.

You must write your own code to calculate these, do not use sklearn or other package's built-in functions for this. You will use these functions to evaluate your classifiers in this assignment.

Also, note the presence of a function called `make_onehot` to turn `y_true` and `y_pred` into a "one hot" representation. In multiclass classification, "one hot" encoding comes up a lot, and you should read up about it at this excellent blog post: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

Even though this function is provided **you do NOT have to use it for any purpose. You MAY use it if you find it helpful.**

Please make use of the `test()` function to ensure that your implementation works. You are encouraged to add test cases.

Using sklearn (15 points)

`sklearn` is a popular machine learning library. `sklearn` classifiers take in numpy arrays(which are arrays of numbers). That means, we need to process our text into an array of numbers(ie, a list of features). Remember from class that the counts of the ngrams present is one popular way to "vectorize" text(to turn text into an array of numerical features).

"Vectorizer" is the `sklearn` term for a Python class that preprocesses text or other data to yield feature vectors.

In this assignment, we will use `sklearn`'s prepackaged "vectorizer", specifically, the `CountVectorizer`): The next step is to choose a classifier to turn the features into probabilities. In this assignment, you are required to try out at least a Naive Bayes classifier, a Logistic Regression classifier, and for full points, another kind of classifier of your own choice.

Finally, we can piece together our "vectorizer" and our classifier in a "pipeline" that can process our data.

Take a look at `sklearn`'s "working with text" guide here:

https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html#building-a-pipeline

This guide should be extremely helpful for this section.

Complete the code in `build_naive_bayes()` and `build_logistic_regr()` to return `sklearn Pipeline` objects that contain a `CountVectorizer` and the respective classifier.

Training and evaluating (15 points)

Note that the dataset has been split into train, dev (meaning development), and test sets. If you look carefully, you will see that they have a 80-10-10 ratio in the number of examples they contain.

We will use the dev set to report metrics regarding our classifiers' performance. You will additionally submit your predictions on the test set(which do not contain any frame annotations), and the top performers will be rewarded points.

Complete the function `main()` such that, for each of the `pipeline`s you built above, you:

1. Train the model(the `Pipeline`) using the `.fit()` method on the training set.
2. Predict the frames for the dev set.
3. Print the macro and micro precision and recall for the above prediction.

Building Your Own Model (25 points)

You will build your own classifier for the frame detection task, and compare your results to that of your classmates. You can choose any other types of classifier.

For classifiers, beyond Naive Bayes and Logistic Regression, you might consider trying SVM, Random Forests, among others. When trying different classifiers, we recommend that you train on training data, and test on the development data, like the previous sections.

There's a function `build_own_pipeline()` that you should use to build your pipeline.

To receive full credit, you must try at least 1 type of classifier (not including Naive Bayes and Logistic Regression).

Complete the function `output_predictions()` such that you:

1. Train your best model on both the training and development data. Call this classifier to predict and output the labels in a text file named `predictions.tsv` (with one label per line); be sure NOT to shuffle the order of the test examples.
2. Call the `output_predictions()` function under where `main()` is called.

Bonus points: Leaderboard and hyper parameter searching (max 10 points)

1. +3 for top 10 on the leaderboard.
2. +5 Top 3 on the leaderboard.
3. +5 for anyone who employs hyperparameter tunning when building their model.

You can look at the documentation on how to employ hyperparameter searching here: https://scikit-learn.org/stable/modules/grid_search.html

In order to receive full bonus credit, your model must be able to outperform all of the baselines.