CS-5340/6340, Programming Assignment #3
Due: Friday, October 29, 2021 by 11:59pm

This assignment will give you experience applying machine learning systems for natural language processing. You will be writing a program to produce feature vector representations to train a machine learning system for Named Entity classification. This task involves assigning a label to individual words within a sentence as detailed below.

---

1 Named Entity Classifier

Your task is to train a machine learning system for Named Entity (NE) classification. A *Named Entity* is a proper name corresponding to a semantic category. Your classifier should recognize 3 types of Named Entities: Persons (PER), Locations (LOC), and Organizations (ORG).

Your classifier will use a **BIO** labeling scheme. A **"B"** indicates the **B**eginning of an entity's name, and **"I"** indicates the **I**nside of an entity's name. The **"O"** label represents **O**ther (i.e., not part of an entity's name). Since your system will identify 3 types of entities, we will need a **"B"** and an **"I"** label for each type. So the ML system will use 7 labels: **B-PER**, **I-PER**, **B-LOC**, **I-LOC**, **B-ORG**, **I-ORG**, and **O**. For example, the sentence *"Ellen Riloff teaches in Utah"* would be labeled as: *Ellen/B-PER Riloff/I-PER teaches/O in/O Utah/B-LOC.*

You should write a program called `feature_extraction` that accepts 2 arguments: (1) a file of training sentences, (2) a file of test sentences.Your program should accept the input via command-line arguments as follows:

```
python3 feature_extraction.py <train_file> <test_file>
```

It is also perfectly fine to use Java. The arguments should similarly be accepted on the command-line in the same order as above.

1.1 Input Files

The training and test files will have the same format. Each file will contain sentences with a gold NE class label and a part-of-speech (POS) tag assigned to each word. Each word will be one instance for the ML classifier. The information for each word will appear on a separate line, formatted as: `label POS word`. One or more blank lines will separate sentences. For example, an input file might look like this:

B-LOC	NNP	Israel
O	NN	television
O	VBD	rejected
O	DT	the
O	NN	skit
O	IN	by
O	DT	the
O	NN	comedian
B-PER	NNP	Mr.
I-PER	NNP	Tuvia
I-PER	NNP	Tzafir

NOTE: Except where indicated otherwise, all string matching should be **case-sensitive** for this assignment (e.g., "Apple" and "apple" should be treated as different strings).

1.2 Feature Types

Your program should be able to produce 11 types of features for a word w:

WORD: the word w itself.
POS: the POS tag p of w.
ABBR: a binary feature indicating whether w is an abbreviation. An abbreviation must: (1) end with a period, (2) consist entirely of alphabetic characters [a-z][A-Z] and one or more periods (including the ending one), and (3) have length ≤ 4.
CAP: a binary feature indicating whether the first letter of w is capitalized.
WORD+1: the word w_{+1} that immediately follows w in the sentence.
WORD-1: the word w_{-1} that immediately precedes w in the sentence.
POS+1: the POS tag p_{+1} of the word that immediately follows w in the sentence.
POS-1: the POS tag p_{-1} of the word that immediately precedes w in the sentence.
LOC: a binary feature indicating whether w matches any of the countries or capital cities listed in the provided file `locations.csv`. Please do *case-insensitive* matching against this file (e.g., "Israel" should match "ISRAEL").
PREF: a binary feature indicating whether the word w_{-1} that immediately precedes w matches any of the prefix terms listed in the provided file `prefixes.txt`.
SUFF: a binary feature indicating whether the word w_{+1} that immediately follows w matches any of the suffix terms listed in the provided file `suffixes.txt`.

There are a few cases where you will also need to define pseudo-words or pseudo-POS tags as feature values. Please use the following conventions:

Pseudo-word	**Description**
PHI	pseudo-word for the beginning-of-sentence position
PHIPOS	pseudo-POS for the beginning-of-sentence position
OMEGA	pseudo-word for the end-of-sentence position
OMEGAPOS	pseudo-POS for the end-of-sentence position

The PHI, PHIPOS, OMEGA, and OMEGAPOS pseudo-words are only necessary for the contextual features (WORD-1, POS-1, WORD+1, POS+1, respectively). They are needed when the current word is the first word of a sentence or when the current word is the last word of a sentence. **Important:** Do not cross sentence boundaries when creating these contextual features! The PHI and OMEGA pseudo-words should never be the current word, they are only placeholders for the beginning and end of sentence positions.

CS-6340 Students

In addition to the features above, your program will need to produce 3 types of global features for a word w:

GLOBCAP: a binary feature indicating whether there is a capitalized instance of w anywhere in the document (including w itself). This feature will be True if you find a word w_o in the document that satisfies ALL the following conditions (otherwise, it will be False):

1. w_o consists only of alphabetic characters.
2. lower-cased w_o is not a preposition in the given `prepositions.txt` file.
3. w_o is not the starting word of a sentence
4. w_o is the same string as w when ignoring case, and the first character of w_o is capitalized.

GLOBPREF: a binary feature indicating whether there is an instance of w in the document (including w itself) that has a prefix in `prefixes.text`. This feature will be True if you find a word w_o in the document that satisfies ALL the following conditions (otherwise, it will be False):

1. w_o consists only of alphabetic characters.
2. w_o is not the first word of a sentence.
3. w_o is equal to w, i.e. strings w and w_o match.
4. the preceding word w_{o-1} matches a prefix term in `prefixes.txt`.

GLOBSUFF: a binary feature indicating whether there is an instance of w in the document (including w itself) that has a suffix in `suffixes.text`. This feature will be True if you find a word w_o in the document that satisfies ALL the following conditions (otherwise, it will be False):

1. w_o consists only of alphabetic characters.
2. w_o is not the last word of a sentence.
3. w_o is equal to w, i.e. strings w and w_o match.
4. the following word w_{o+1} matches a suffix term in `suffixes.txt`.

1.3 Output Files

Your `feature_extraction` program should generate two `csv` (comma-separated values) files. These files should contain the feature vector representation for every word in the train or test files, along with the corresponding gold NE label.

You should print these files to the working directory and name them: <`inputfile`>`_ft.csv`, where <inputfile> is the input file's name prior to the ".txt" extension. For example, given `mytrain.txt` and `mytest.txt` as input, you should print files in the working directory named `mytrain_ft.csv` and `mytest_ft.csv`.

In each file, the first line should be a header with the names of the features (defined in Section 1.2) as the names of the columns in the csv file. Importantly, the header should start with the column name **LABEL**. The remaining columns should represent the features, in alphabetical order. Each subsequent row after the header should correspond to the feature vector of a word in the input document.

Note that the **CS-5340 Students** and **CS-6340 Students** should produce files that have 12 and 15 columns, respectively.

As an example, consider a data set containing only the sample input shown earlier in Section 1.1. The generated csv file should look like this:

```
LABEL,ABBR,CAP,LOC,POS,POS+1,POS-1,PREF,SUFF,WORD,WORD+1,WORD-1
B-LOC,0,1,1,NNP,NN,PHIPOS,0,0,Israel,television,PHI
O,0,0,0,NN,VBD,NNP,0,0,television,rejected,Israel
O,0,0,0,VBD,DT,NN,0,0,rejected,the,television
O,0,0,0,DT,NN,VBD,0,0,the,skit,rejected
O,0,0,0,NN,IN,DT,0,0,skit,by,the
O,0,0,0,IN,DT,NN,0,0,by,the,skit
O,0,0,0,DT,NN,IN,0,0,the,comedian,by
O,0,0,0,NN,NNP,DT,0,0,comedian,Mr.,the
B-PER,1,1,0,NNP,NNP,NN,0,0,Mr.,Tuvia,comedian
I-PER,0,1,0,NNP,NNP,NNP,1,0,Tuvia,Tzafir,Mr.
I-PER,0,1,0,NNP,OMEGAPOS,NNP,0,0,Tzafir,OMEGA,Tuvia
```

IMPORTANT: The generated csv files should preserve the same word order as the input files. For instance, if the first two sentences in an input file, $S1$ and $S2$, have 10 and 5 words respectively, then the first 10 lines (following the header) should correspond to the words in $S1$ in the original order, and the next 5 lines should correspond to the words in $S2$ in the original order.

1.4 Debugging

We have provided sample .csv files on Canvas to help you debug your programs. We have also provided a script called `csv_compare.py` that accepts two csv files and prints the differences between them.

For instance, you could run the command:

```
python3 compare.py small_train_ft_CS5340.csv MyOutput.csv
```

and the script will print whether the files have the same number of lines, the same column names, and the line and column number of entries that differ.

---

2 Applying Machine Learning

We are providing a python script called `ml.py` that will train and test a logistic regression classifier using your generated .csv files.

This program will recognize all possible feature types corresponding to the names of the features shown in Section 1.2. There must always be one feature, WORD, which should appear first. That can be followed by any combination of additional features, in any order!

To use the script over your featurized data invoke the command:

```
python3 ml.py train_ft.csv test_ft.csv WORD [...  <feature_n >]
```

This program will generate a table showing the performance of the classifier under different metrics for each of the NE classes. (It will also print the number of training epochs used by the algorithm until convergence, but you can ignore that for the purposes of this assignment).

2.1 Experiments

You should create logistic regression classifiers using your csv files and experiment with different feature combinations using the ml.py program. All students should perform 3 experiments, and CS-6340 students should also perform a fourth experiment:

Experiment 1: Produce results for a classifier that uses only the features WORD, POS, CAP, and ABBR.

Experiment 2: Produce results for a classifier that uses the four features above as well as the contextual features WORD-1, WORD+1, POS-1, and POS+1.

Experiment 3: Produce results for a classifier that uses all 8 features above as well as the external list features LOC, PREF, and SUFF.

Experiment 4 (CS-6340 students only) Produce results for a classifier that uses all 11 features above as well as the global features GLOBCAP, GLOBPREF, and GLOBSUFF.

SUBMISSION INSTRUCTIONS

On CANVAS, please submit an archived (.tar) and/or gzipped (.gz) file containing:

1. The source code files for your `feature_extraction` program. Be sure to include all files that we will need to compile and run your program!
2. A README file that includes the following information:
   - how to compile and run your code
   - which CADE machine you tested your program on (this info may be useful to us if we have trouble running your program)
   - any known bugs, problems, or limitations of your program
3. On the **CADE** machines, run your feature extraction program on the `train.txt` and `test.txt` files. For example:

   ```
   python3 feature_extraction.py train.txt test.txt
   ```

   CS-5340 students should have 11 features (12 columns) in their .csv files, and CS-6340 students should have 14 features (15 columns) in their .csv files. Submit the two generated csv files: `train_ft.csv` and `test_ft.csv`.
4. Use the `ml.py` script to perform the experiments described in Section 2.1 using your `train_ft.csv` and `test_ft.csv` files. You will need to run this program several times using the different feature combinations described in Section 2.1.

   You can save the output by adding the symbol > and the name of your desired output file at the end of the command line. For instance, you can save the results of the first experiment like this:

   ```
   python3 ml.py train_ft.csv test_ft.csv WORD POS CAP ABBR > experiment1.txt
   ```

   Please submit the following files:

   (a) `experiment1.txt`
   (b) `experiment2.txt`
   (c) `experiment3.txt`
   (d) (CS-6340 students only) `experiment4.txt`

Important: All of the python packages needed to run the machine learning script `ml.py` are already installed on the CADE machines. Do not download/install packages or modify the script in any way. And a reminder that your program must be written in Python or Java and must compile and run on the linux-based CADE machines! We will not grade programs that cannot be run on the linux-based CADE machines.

Your program will be graded based on new data files! **So please test your program thoroughly to evaluate the generality and correctness of your code!** Even if your program works perfectly on the examples that we give you, that does not guarantee that it will work perfectly on new files.