CS-5340/6340, Programming Assignment #1
Due: Wednesday, September 14, 2022 by 11:59pm

This assignment will give you experience creating machine learning systems for natural language processing. You will be writing a program to produce feature vectors to train a machine learning model for sentiment classification.

---

1 Sentiment Classification Task

Your task is to create a machine learning model for *sentiment classification*. Your classifier should accept sentences from movie reviews as input and label each sentence as either **Positive** or **Negative**. We will provide a software tool to produce the machine learning (ML) model. You will need to write a program that creates feature vector representations for sentences, which will ultimately be the input to the ML tool.

You should write a program called `sentiment` that accepts three arguments as input: (1) a file of training sentences, (2) a file of test sentences, (3) a sentiment lexicon. Your program should accept the input as command-line arguments in the following order:

```
<train_file> <test_file> <lexicon_file>
```

For example, if you are using python, then we should be able to run your program like this:

```
python3 sentiment.py train.txt test.txt lexicon.txt
```

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

1.1 Training and Test Files

The training and test files will have the same format. Each training or test instance will be a sentence paired with a gold class label. The integer 1 represents a **Positive** class label, and the integer 0 represents a **Negative** class label. Each instance will begin with either a 1 or a 0 indicating its class label, immediately followed by the sentence, with each word on a separate line. All words will be in lower case. One or more blank lines will separate different instances.

Below is an example of a small training (or test) file containing two instances:

```
1
love
this
movie
!

0
that
was
awful.
```

Important Use the words exactly as they are provided. Some words may be isolated punctuation marks, while others may be strings that include punctuation marks (e.g., "awful."). Do not do any additional tokenization. For example, "awful." should be treated as a single word.

1.2 Sentiment Lexicon File

Each line of the lexicon file will consist of one word followed by its sentiment class (POS or NEG). There may be any amount of whitespace between the word and its class label. For example, a tiny lexicon file could look like this:

```
happy POS
excited POS
sad NEG
angry NEG
love POS
```

1.3 Feature Types

Your program should extract 11 features from the training and test sentences. The features that you need to extract from each sentence *S* are defined below.

W1: the first word in *S*
W2: the second word in *S*
W3: the third word in *S*
W4: the fourth word in *S*
W5: the fifth word in *S*
EXCL: the number of exclamation points in *S*
POS: the number of words in *S* that appear in the sentiment lexicon with class POS
NEG: the number of words in *S* that appear in the sentiment lexicon with class NEG
DIFF: POS minus **NEG**
NGTPOS: the number of *Negation* words in *S* that occur "near" a Positive word (i.e., a word that appears in the sentiment lexicon with class POS).
NGTNEG: the number of *Negation* words in *S* that occur "near" Negative word (i.e., a word that appears in the sentiment lexicon with class NEG).

The set of *Negation* words should be defined as: { *no, not, never, isn't, wasn't, won't* }

A few clarifications and details:

- The W1-W5 features will have string values. All other features will have integer values.
- For the W1-W5 features, if a sentence has fewer than the necessary words, use the dummy symbol "UNK" as the feature value. For example, if a sentence only contains 4 words, then the value of W5 should be "UNK".
- For the POS, NEG, NGTPOS, and NGTNEG features, if the same word appears more than once in a sentence, count all instances. For example, if a sentence contains multiple instances of "good", every instance counts as a lexicon match.
- For the NGTPOS and NGTNEG features, define "near" as meaning that a Negation word occurs up to 2 words before or after a lexicon term. For example, if "good" is a Positive word in the sentiment lexicon, then "not good" and "not very good" would count for the NGTPOS feature, but "not at all good" would not count.

1.4 Output Files

Your `sentiment` program should generate two `.csv` (comma-separated values) files. These files should contain the feature vector representations for the sentences in the training and test files, along with the corresponding gold sentiment label for each sentence.

You should print these files to the working directory and name them: `<inputfile>_features.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_features.csv` and `mytest_features.csv`.

DO NOT hard-code specific filenames in your source code. Your program should allow the input files to have any name, although you can assume that both filenames will end with a ".txt" extension.

In each file, the first line should be a header with the names of the features (defined in Section 1.3) used 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 sentence in the input file.

COMMAS: If a comma appears as a value itself or inside a word, the word/phrase needs to be put inside quotes (e.g., ",") in order for the .csv file to be processed properly by the ML tool. (Some libraries may do this for you automatically, but if not then you should print quotes yourself around any words that include a comma.)

IMPORTANT: The generated .csv files must preserve the same sentence order as the input files. For instance, the first 3 sentences of an input file should be the first 3 sentences of the corresponding .csv file (after the header), in exactly the same order.

1.4.1 Example

As an example, consider the following sentence (written horizontally for readability):

1 *a great movie , great acting but the plot was bad , not good at all.*

Assume that *good* and *great* are Positive words in the sentiment lexicon, and *bad* is a Negative word in the sentiment lexicon. Below shows what the corresponding .csv file should look like for this sentence: the first row is the header and the second row shows the feature vector for the sentence.

LABEL,DIFF,EXCL,NEG,NGTNEG,NGTPOS,POS,W1,W2,W3,W4,W5
1,2,0,1,1,1,3,*a*,*great*,*movie*, *","*,*great*

1.5 Creating the Machine Learning Model

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 of the feature names listed in Section 1.3 so you can instruct the tool to use any subset of those features simply by listing them on the command line (in any order). You must list at least one feature though.

To apply the script to your .csv files, invoke the command:

```
python3 ml.py <train_csv_file> > <test_csv_file> [<feature_1 ...  <feature_n >]
```

For example, you could run it like this:

```
python3 ml.py train_features.csv test_features.csv W1 W2 W3 W4 W5 POS NEG
```

The ml.py tool will also recognize a special argument named "ALL", which you can use instead of listing a set of features. Given the ALL argument, it will use <u>all</u> of the features in the provided .csv files. For example, you could run it like this:

```
python3 ml.py train_features.csv test_features.csv ALL
```

This ML tool will (1) train a logistic regression classifier for sentiment analysis using the given training file, (2) apply the classifier to the examples in the test file, and (3) print a table showing the performance of the classifier on the test examples. (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).

1.6 Experiments

Using the `ml.py` tool, you should perform 4 experiments:

Experiment 1: Use only features W1, W2, W3, W4, W5
This experiment will create and evaluate a classifier that only uses the first 5 words of each sentence.

Experiment 2: Use only features POS, NEG
This experiment will create and evaluate a classifier that only uses words that are present in the sentiment lexicon.

Experiment 3: Use only features W1, W2, W3, W4, W5, POS, NEG

Experiment 4: Use ALL of the features (i.e., W1, W2, W3, W4, W5, POS, NEG, DIFF, EXCL, NGTPOS, and NGTNEG).

Part II: For CS-6340 students only (30 additional points)

CS-6340 students should write an additional program that will create a ML classifier for sentiment classification using *bag-of-words (BOW)* features. This program should be called `sentimentBOW` and accept three arguments as input: (1) a file of training sentences, (2) a file of test sentences, and (3) an integer representing a frequency threshold. This program should accept the input as command-line arguments in the following order:

```
<train_file> <test_file> <threshold>
```

For example, if you are using python, then we should be able to run your program like this:

```
python3 sentimentBOW.py train.txt test.txt 5
```

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

The training and test files will have the same format described in Section 1.1.

1.7 Constructing BOW Feature Set

For the BOW model, you will need to *dynamically* generate the feature set from the training texts. (In contrast to the pre-defined feature set in the earlier part of the assignment.) To create the feature set, you should:

- Collect all of the words in the training file.
- Select the words that have a frequency $\geq$ threshold θ, which will be the threshold value specified on the command line.
- Sort the words by their frequency in the training file (highest frequency listed first). When there are ties, sort the words alphabetically (lowest first, so A before Z).
- Assign a Feature ID (identifier) to each of the selected words based on their position in the sorted list. A word's Feature ID should be defined as "F#" where the number # is the word's position in the sorted list, beginning with 0 for the top position.

As an example, consider the following sentence (written horizontally for readability):

1 *This movie was so so good , really good - such a good movie !*

If this sentence was the entire training file, then the threshold θ=2 should produce a *BOW Feature Set* consisting of 3 features with these IDs:

F0 good
F1 movie
F2 so

IMPORTANT: The *BOW Feature Set* should be created from the training file ONLY. These features will then be used to create feature vectors for both the training and test sentences. It is critical that both the training and test sentences use the same feature set.

1.8 Creating Feature Vectors

Once you've extracted a *BOW Feature Set*, you will need to generate a feature vector for each sentence, which will be the sentence's representation in the .csv files (described in the next section).

For each sentence, you will create a vector that includes all of the BOW features. The value for each feature will be the frequency of that word in the sentence.

Using the example above, you would create a feature vector with 3 features: F0, F1, and F2.

For the example sentence shown in the previous section, the values would be F0=3, F1=2, F2=2 (because "good" occurs 3 times in the sentence, while "movie" and "so" each occur 2 times).

All words in a sentence that are not in the feature set should be ignored!!

1.9 Output Files

Your `sentimentBOW` program should generate two `.csv` (comma-separated values) files. These files should contain the feature vector representations for the sentences in the training and test files, along with the corresponding gold sentiment label for each sentence following the same formatting described earlier.

You should print these files to the working directory and name them: `<inputfile>_BOW#.csv`, where <inputfile> is the input file's name prior to the ".txt" extension and the number (#) is the threshold specified on the command line. For example, given `mytrain.txt` and `mytest.txt` as input with a threshold value 5, you should print files in the working directory named `mytrain_BOW5.csv` and `mytest_BOW5.csv`.

DO NOT hard-code specific filenames in your source code. Your program should allow the input files to have any name, although you can assume that both filenames will end with a ".txt" extension.

In each file, the first line should be a header with the names of the features (defined in Section 1.7) used 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 sentence in the input file.

To illustrate, using the same example from the previous section, the .csv file would look like this:

```
LABEL,F0,F1,F2
1,3,2,2
```

SUBMISSION INSTRUCTIONS

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

1. The source code files for your `sentiment` program, and also for your `sentimentBOW` program if you are in CS-6340. Be sure to include <u>all</u> 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 `sentiment` program using the `trainB.txt`, `testB.txt`, and `lexicon.txt` files provided on Canvas. For example:

   ```
   python3 sentiment.py trainB.txt testB.txt lexicon.txt
   ```

   Submit the two generated csv files: `trainB_features.csv` and `testB_features.csv`.

4. Use the `ml.py` script to perform the experiments described in Section 1.6 using your `trainB_features.csv` and `testB_features.csv` files.

   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 trainB_features.csv testB_features.csv W1 W2 W3 W4 W5 > experiment1.txt
   ```

   Please submit the following files:

   (a) `experiment1.txt`
   (b) `experiment2.txt`
   (c) `experiment3.txt`
   (d) `experiment4.txt`

5. **CS-6340 students only:** On the **CADE** machines, run your `sentimentBOW` program using the `trainB.txt` and `testB.txt` files provided on Canvas. Run your program twice using the threshold values 5 and 10. For example:

   ```
   python3 sentimentBOW.py trainB.txt testB.txt 5
   python3 sentimentBOW.py trainB.txt testB.txt 10
   ```

   Submit the 4 generated csv files: `trainB_BOW5.csv`, `testB_BOW5.csv`, `trainB_BOW10.csv`, and `testB_BOW10.csv`

6. **CS-6340 students only:** Use the `ml.py` script to create ML models for both sets of .csv files: (1) use the `trainB_BOW5.csv` and `testB_BOW5.csv` files, and (2) use the `trainB_BOW10.csv` and `testB_BOW10.csv` files.

   Please submit the output files as `experiment_BOW5.txt` and `experiment_BOW10.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.