CSCE-638, Programming Assignment #2 Sentiment Analyzer
Due: Sunday, October 9, 2022 by 11:59pm

Your goal for this homework is to perform Sentiment Analysis: classifying an entire movie review as positive or negative.

Train a Naive Bayes classifier and a Perceptron classifier on the imdb1 data set provided with the starter package. This is the actual Internet Movie Database review data used in the original Pang and Lee paper:

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? Sentiment Classification using Machine Learning Techniques. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 79-86.

The python starter code comes already set up for 10-fold cross-validation training and testing on this data. Recall that cross-validation involves dividing the data into several sections (10 in this case), then training and testing the classifier repeatedly, with a different section as the held-out test set each time. Your final accuracy is the average of the 10 runs. When using a movie review for training, you use the fact that it is positive or negative (the hand-labeled "true class") to help compute the correct statistics. But when the same review is used for testing, you only use this label to compute your accuracy. The data comes with the crossvalidation sections; they are defined in the file:

```
data/poldata.README.2.0
```

Naive Bayes Algorithm:

(3 Points!) Task 1- Your first task is to implement the Naive Bayes classifier training and testing code and evaluate them using the cross-validation mechanism.

(1 Point!) Task 2- Evaluate your model again with the stop words removed. Does this approach affect average accuracy (for the current given data set)?

(2 Points!) Task 3- You will implement a binarized version of the multinomial Naive Bayes classifier. The binarized version uses word presence/absence as features (binary) in classifying a document instead of word counts (frequencies). Therefore, whether a word, e.g.,"great", appears one time or ten times in a document does not make a difference or affect the sentiment prediction result on the document. But it does make a difference whether a word was seen or never seen in a document. **Hint:** both the training and the testing code remain the same with the only exception that in both training and testing, instead of considering multiple occurrences of a word in EACH document, we clip all word counts in EACH document at 1.

Two option booleans FILTER_STOP_WORDS, BOOLEAN_NB have been defined. If FILTER_STOP_WORDS is true, stop word tokens will be removed from both training and test documents. If the BOOLEAN_NB flag is true, then your code (specifically the train and evaluate functions) should train and test on the binarized version of the Naive Bayes classifier (with boolean features - for presence/absence of each word as opposed to word counts).

Perceptron Algorithm:

(4 Points!)

Next, implement the Perceptron classification algorithm. Your perceptron code should implement parameter averaging. Instead of using parameters by the end of training to classify test examples, use the average of the parameters after updating with each data instance in each iteration. A nice trick for doing this efficiently is described in section 2.1.1 of Hal Daume's thesis (http://users.umiacs.umd.edu/~hal/docs/daume06thesis.pdf).

The only hyperparameter is the number of iterations. Run the classifier and report the cross validation results with various numbers of iterations (for exaple: 1, 10, 50, 100, 1000).

Extra Credit (5 Points at maximum!) (Hierarchical Attention Network):

Implement the Hierarchical Attention Network for document classification. The network is introduced in the paper:

Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 1480-1489.

(4 Points!) Task 1- You code should (1) implement the neural network model with existing software library for deep learning (PyTorch or Keras, Tensorflow); (2) train function; (3) classify function. The dataset is loaded as numpy array `x_data,y_data`. `x_data` stores the word embeddings as input. You will need to add new classes and new functions with the chosen deep learning library.

Questions to answer in your report:

1). Two hyperparameters are the number of iterations and batch size (default=1). Run the classifier with pre-trained glove word embeddings and report the cross validation results with various numbers of iterations (for example: 1, 5, 10, 20) and different batch sizes (for example, 1, 2, 4, 8).

2). With the best parameters you found, run the classifier but with pre-trained glove word embeddings replaced by one hot word vectors (word indices), report the performance your system obtain. Which set of word vectors enables higher performance?

3) Remove the attention layer at the word level, at the sentence level or at both levels, report the performance of the system variations with only one attention layer or with no attention layer, how does the performance change compared with the full system? explain why?

(1 Points!) Task 2- In this task, replace pre-trained glove word embeddings with contextualized deep word embeddings, ELMO or BERT. Run the classifer and report the performance. You'll need to write your own data loader with the chosen DL library.

ELMO was introduced in the paper:

Peters, Matthew and Neumann, Mark and Iyyer, Mohit and Gardner, Matt and Clark, Christopher and Lee, Kenton and Zettlemoyer, Luke. 2018. Deep Contextualized Word Representations. Proceedings of the 2018 Conference of the North American

Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 2227-2237.

BERT was introduced in the paper:

Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 4171-4186.

---

1. Python Starter Code

You are encouraged to use Python for the programming assignments in this class, but if you prefer, you can use Java too. However, simple starter code will be provided in some of the programming assignments (including this one) and only in Python.

In the same directory where the provided code and data are, you can run the code:

```
cd python
python NaiveBayes.py ../data/imdb1
OR
python NaiveBayes.py -f ../data/imdb1
OR
python NaiveBayes.py -b ../data/imdb1
OR
python Perceptron.py ../data/imdb1/ 1 (run for 1 iteration)
OR
(optional) python HierAttNet_Glove.py ../data/imdb1/ 1 2 glove_embedding.txt
(with glove embedding, run for 1 iteration with batch size 2)
OR
(optional) python HierAttNet.py ../data/imdb1/ 1 2
(with word indices, run for 1 iteration with batch size 2)
OR
(optional) python HierAttNet_Deep.py ../data/imdb1/ 1 2
(with deep contextualized word representation, run for 1 iteration with batch size 2)
```

The functions

```
def classify(self, words):
AND
def addExample(self, klass, words):  (in the code for NaiveBayes and Perceptron)
AND
def train(self, split, iterations):  (in the code for Perceptron and HierAttNet)
```

have been flagged for you with the universal "TODO" marker. You're free to add other elements further invoked from these three functions (you would need to add some data structures for your work - where to add them will be mentioned in the starter code).

OUTPUT FORMATTING

The results will look something like the figure in the next page. Even if you program using a language different from Python and do not use the starter code, your program output should follow the same structure.

GRADING CRITERIA

Your program will be graded all based on cross validation results on the provided training set, and mainly based on the correctness of your code (please refer back to the first section for how we will weight each task). We understand that you may produce slightly different results if you have implemented some details in a different way. Therefore, please describe enough details in your written report. In addition, we will test your code on a separate test set to make sure it runs properly. For instance, if we found any experimental result has been hard coded, you won't get any credit for that part.

ELECTRONIC SUBMISSION INSTRUCTIONS (a.k.a. "What to turn in and how to tun in")

You need to submit 2 things:

1. The source code files for your program. Be sure to include all files that we will need to compile and run your program!
2. A report file that includes the following information:
   - how to compile and run your code
   - results and analysis
   - any known bugs, problems, or limitations of your program

   REMINDER: your program **must** compile and run. We will not grade programs that cannot be run.

How to turn in:

1. please include everything in a single .zip package, and submit it via canvas.

[INFO] Fold 0 Accuracy: 0.500000

[INFO] Fold 1 Accuracy: 0.500000

[INFO] Fold 2 Accuracy: 0.500000

[INFO] Fold 3 Accuracy: 0.500000

[INFO] Fold 4 Accuracy: 0.500000

[INFO] Fold 5 Accuracy: 0.500000

[INFO] Fold 6 Accuracy: 0.500000

[INFO] Fold 7 Accuracy: 0.500000

[INFO] Fold 8 Accuracy: 0.500000

[INFO] Fold 9 Accuracy: 0.500000

[INFO] Accuracy: 0.500000

Figure 1: Sample Output for Sentiment Analyzer