Homework 3: Sentiment Analysis [starter code]

Note: depending on your machine, this assignment may take a long time to run. The code in this assignment takes longer to run than previous ones, expect upwards of 10 minutes. Start working on it early.

Your goal for this homework is to perform **Sentiment Analysis**: classifying movie reviews as positive or negative. Recall from lecture that sentiment analysis can be used to extract people's opinions about all sorts of things (congressional debates, presidential speeches, reviews, blogs) and at many levels of granularity (the sentence, the paragraph, the entire document). Our goal in this task is to look at an entire movie review and classify it as positive or negative.

Model - **Logistic Regression**

You will be using **Logistic Regression** to classify reviews as either positive or negative. You will first implement a unigram model that uses individual word counts as features and predicts what class a new review is likely to belong to. It does so using a weight vector obtained after training the model. Finally, you will implement your own features and/or heuristics to improve performance (more on this below).

Assignment

Train a **Logistic Regression** classifier on the **imdb1** data set provided with the starter code. This is the actual Internet Movie Database review data used in the original Pang and Lee paper. The starter code comes already set up for training and testing on this data. 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 class. But when the same review is used for testing, you only use this label to compute your accuracy.

Logistic Regression:

Task 1a - Your first task is to train a logistic regression model using unigram (bag-of-words) features. Add your code in the **train()** function. We have provided a **gradientDescent()** function that you should call in train(). Use the default parameters for gradient descent ie. gradientDescent(alpha=0.001, numiters=1000). You will use sklearn's CountVectorizer() to create your unigram feature vector. We have provided a self.vect that is initialized to the CountVectorizer you should use (more information about CountVectorizer() is given below).

Task 1b - Implement the sigmoid() function. Note that the input to your sigmoid function can either be scalar or NumPy array so implement your function using NumPy's exponential function (Links to an external site.).

Task 1c - Implement the predict() function that determines what class an input belongs to based on its score.

Task 1d - Implement the classify() function that classifies a review as either positive or negative.

Task 2 - When INCLUDE_LEXICON is set to True, add two more features to our Logistic Regression model: the total number of positive words in a review, and the total number of negative words in a review. We have already preprocessed the NRC emotion lexicon for you, and you can access the set of lexicon words in self.posWords and self.negWords. We have also provided the function addFeatures() that takes in the original feature matrix X and a list of lists as the second argument; each list element is a feature vector for one input. This allows the logistic regression model to include your own features along with those from CountVectorizer().

If you're curious - what other preprocessing techniques and features do you think could improve your Logistic Regression performance? Try them out! (note: this is completely optional and not part of your grade)

The data is divided into a **training set**, **development (validation) set**, and **test set**. Recall that the training set is used to compute the statistics for your model. These statistics are then used to classify the documents in the development and test sets. For this assignment, you have access to the training set and the dev set. The test set is hidden, but your submission will be evaluated on it as well.

- The training set is located in **data/train/**
- The dev set is located in **data/dev/**
- Within each of these directories, there is a directory for each class: pos and negative. Documents belonging to class positive are located in the **pos** directory and those belonging to class **neg** are located in the **neg** directory. Every file in these directories is a single review.

Finally, note that the data you are given is already **preprocessed**; all text has been converted to lowercase and stopwords removed. Depending on the specific NLP task, preprocessing can significantly improve performance. You do not need to do any additional preprocessing.

Libraries Required

This assignment uses sklearn and NumPy. Install them using pip:

```
(cs124-env) <sunetid>@rice09:~$ pip install numpy scikit-learn
```

Other Useful Information

CountVectorizer() converts a collection of text documents to a term-document matrix of token count. Rows in the matrix correspond to individual documents and each column to words. The value at Matrix[row][col] is the number of occurrences of the word col in document row.

Example usage from [https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (Links to an external site.)](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html):

```
>>> from sklearn.feature_extraction.text import CountVectorizer

>>> corpus = [

...     'This is the first document.',

...     'This document is the second document.',

...     'And this is the third one.',

...     'Is this the first document?',

... ]

>>> vectorizer = CountVectorizer()

>>> X = vectorizer.fit_transform(corpus)

>>> print(vectorizer.get_feature_names())

['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']

>>> print(X.toarray())

[[0 1 1 1 0 0 1 0 1]

 [0 2 0 1 0 1 1 0 1]

 [1 0 0 1 1 0 1 1 1]

 [0 1 1 1 0 0 1 0 1]]
```

In our implementation:

corpus => list of documents (strings), derived from trainData (should be in the same form as the corpus example above)

vectorizer => self.vect.

vectorizer.fit_transform(corpus) will learn the vocabulary and return the term-document matrix (this is functionally the same as calling fit then transform) vectorizer.transform([document]) will transform a single document to a term-document matrix

Where to Make Your Changes

To ensure that your code works properly not only when running from the command line but also when executed by our autograder, you should limit your changes to train(), sigmoid(), predict(), and classify() methods within LogisticRegression.py. You're free to add other elements further invoked from these functions (you would need to add some data structures for your work -- where to add them will be mentioned in the starter code), but note that main() is **NOT** executed by the autograder so you cannot rely on anything added there.

Changes beyond the functions listed above, if you choose to make any, should be done with caution. The autograder directly manipulates the INCLUDE_LEXICON variable. If the INCLUDE_LEXICON switch is true, your code should build a model that uses lexicon features in addition to unigram features.

Evaluation

Your classifier will be evaluated on the training, development, and held-out/unseen test sets from the **imdb1** data.

Running the code

In your terminal, execute

```
$ python LogisticRegression.py

This requires the LogisticRegression.py file to be located in the same direct
ory as data/. This is the default in the starter code.
```

Adding a flag (*-l*)... l as in lexicon

```
$ python LogisticRegression.py -l
```

Flag -l invokes your model with lexicon features.

Submitting Your Solution

Submit your assignment via **Gradescope** (www.gradescope.com). We expect the following files in your final submission:

- LogisticRegression.py

You will only be able to see your score on the dev set. The test set scores will be released after the deadline.