README.md

NLP ASSIGNMENT #2


```
Naveen Singh Pundir
17111026
CS671A: Introduction to Natural Language Processing
IIT KANPUR
```


Sentiment Analysis on IMDB movie review dataset

Data

The movie review dataset can be downloaded from Large Movie Review Dataset. We will also need to use GloVe and Word2Vec pre-trained vectors for word representation. They can easily be downloaded from GloVe and Word2Vec. However, we will also be using our own word vectors trained using Gensim on the review dataset.

Note : The above files should be decompressed and put in the main folder.

Dependencies

- Python 3.6
- Scikit-learn
- Tensorflow
- Keras
- Gensim
- NLTK

Most of the dependencies can be simply installed using `pip install <pack-name>`.

Run

Before running the code, make sure that you have downloaded the dataset as well as the pre-trained vectors and extracted them in the main directory. Now simply run:

```
$ ./run.sh
```

This file contains

```
#!/bin/bash
python preprocess.py
python word2vec.py
python emb2dict.py
```

`preprocess.py`: Convert all train, test and unsupervised files into CSV files and learns dictionary from them, and stores them in *preprocessed_data* directory

`word2vec.py` : Learns the word embeddings from the dataset

`emb2dict.py` : Creates the subset from *GloVe*, *Word2Vec* and learned embeddings using the learning vocabulary and stores them in *preprocessed_data* directory (for faster access).

Now, all the necessary requirements are done and we can run the code.

The main file is `train.py` in the main directory which will help you invoke different parameters to be used.

Simply run `$ python train.py --help` to know about the parameters to use. For eg:

```
$ python train.py -r tfidf -a ffnn --epochs 5
```

Description

Representation:

- **Binary bag of words :** `CountVectorizer(binary=True)`
- **Normalized Term frequency (tf) representation :** `TfidfVectorizer(use_idf=False)`
- **Tfidf representation :** `TfidfVectorizer(ngram_range=(1,2))`
- **Average of Word2Vec and GloVe vectors**
- **Averaged sentence vectors for sentences in the document:** Trained a Doc2Vec model on the set of all sentences (after breaking each review into sentences). After the training is complete, the sentences belonging to a particular review is averaged to get the vector for that review. From this, you can get the vector for each review in the training corpus. At the test time, you can again break the test review into sentences, and use the model to infer vector for each sentence and again average the sentence vectors to get the vector for each test review in the test set.
- **Paragraph vector:** Same approach as above using Doc2Vec, the only difference is that, in this, each review is considered as a single document and same during test time. The paragraph vector is trained on 75,000 (25,000 labelled and 50,000 unlabeled instances) training documents. This approach is same as done in Original Doc2Vec paper. Although original paper claims astounding `7.42%` error rate, we didn't achieve this in our code. For that, we would have to tune hyperparameters properly to train our model.

Classification Algorithm:

- **Naive Bayes:** Naive Bayes algorithm is simply invoked using `MultinomialNB().fit(self.rep, self.y)`. The Naive Bayes algorithm requires us to have non-negative values for feature weights, therefore, Word Vector averaging, Sentence Vector (Doc2Vec), and Paragraph Vector (Doc2Vec) can't be used with this algorithm.
- **Logistic Regression :** `LogisticRegression().fit(self.rep, self.y)` . The logistic regression algorithm works very fast and also give decent performance compared to other classification algorithms.
- **Support Vector Machine :** `SVC().fit(self.rep, self.y)`. This algorithm works fine but is very slow.

Our *feed forward neural network* as well as *recurrent neural network* is implemented in `Keras` using `Tensorflow` as backend. Implementation details are as follows:

- **Feed Forward Neural Network:**

```
# Creating a sequential model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.1))
# Last layer neuron ranging value from 0 to 1
model.add(Dense(1, activation='sigmoid'))
```

- **Recurrent Neural Network:**

```
# Prepare tokenizer
tokenizer = Tokenizer(num_words=gc.VOCAB_SIZE)
tokenizer.fit_on_texts(X_train)

# Encode the training and test data
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

# Pad documents to a max length of MAX_SEQUENCE_LENGTH
X_train = pad_sequences(X_train, maxlen=MAX_SEQUENCE_LENGTH)
X_test = pad_sequences(X_test, maxlen=MAX_SEQUENCE_LENGTH)
```

From the above code, we convert our dataset into the required format for LSTM or GRU. After this, we can simply add LSTM or GRU layer in our model for training.

```
if self.seq_model == 'lstm':
    self.model.add(LSTM(self.hidden_units, dropout=self.dropout, recurrent_dropout=self.dropout))
else:
    self.model.add(GRU(self.hidden_units, dropout=self.dropout, recurrent_dropout=self.dropout))
```

Score

SNo	Representation	Classification	Accuracy
1	Binary bag of words	Naive Bayes	84.716%
2	Binary bag of words	Logistic Regression	85.284%
3	Binary bag of words	Support Vector Machine (SVM)	84.172%
4	Binary bag of words	Feed Forward Neural Network	87.388%
5	Normalized Term frequency (tf)	Naive Bayes	84.864%
6	Normalized Term frequency (tf)	Logistic Regression	87.216%
7	Normalized Term frequency (tf)	Support Vector Machine (SVM)	60.504%
8	Normalized Term frequency (tf)	Feed Forward Neural Network	87.308%
9	Tfidf representation	Naive Bayes	86.424%
10	Tfidf representation	Logistic Regression	88.948%
11	Tfidf representation	Support Vector Machine (SVM)	67.972%
12	Tfidf representation	Feed Forward Neural Network	87.904%
13	Average Word2Vec (without tfidf)	Logistic Regression	79.892%
14	Average Word2Vec (without tfidf)	Support Vector Machine (SVM)	68.604%
15	Average Word2Vec (without tfidf)	Feed Forward Neural Network	79.772%
16	Average Word2Vec (with tfidf)	Logistic Regression	79.268%
17	Average Word2Vec (with tfidf)	Support Vector Machine (SVM)	71.636%
18	Average Word2Vec (with tfidf)	Feed Forward Neural Network	79.26%
19	Average GloVe (without tfidf)	Logistic Regression	80.38%
20	Average GloVe (without tfidf)	Support Vector Machine (SVM)	78.46%
21	Average GloVe (without tfidf)	Feed Forward Neural Network	78.056%
22	Average GloVe (with tfidf)	Logistic Regression	78.728%
23	Average GloVe (with tfidf)	Support Vector Machine (SVM)	77.292%
24	Average GloVe (with tfidf)	Feed Forward Neural Network	78.132%
25	Average Gensim Word2Vec (without tfidf)	Logistic Regression	87.888%
26	Average Gensim Word2Vec (without tfidf)	Support Vector Machine (SVM)	87.672%
27	Average Gensim Word2Vec (without tfidf)	Feed Forward Neural Network	87.344%
28	Average Gensim Word2Vec (with tfidf)	Logistic Regression	87.152%
29	Average Gensim Word2Vec (with tfidf)	Support Vector Machine (SVM)	87.052%
30	Average Gensim Word2Vec (with tfidf)	Feed Forward Neural Network	86.948%
31	Averaged Sentence Vectors	Logistic Regression	69.264%
32	Averaged Sentence Vectors	Support Vector Machine (SVM)	56.472%
33	Averaged Sentence Vectors	Feed Forward Neural Network	70.348%

SNo	Representation	Classification	Accuracy
34	Paragraph Vector	Logistic Regression	84.88%
35	Paragraph Vector	Support Vector Machine (SVM)	85.32%
36	Paragraph Vector	Feed Forward Neural Network	84.736%
37	GloVe	LSTM	84.454%
38	GloVe	GRU	84.228%
39	Word2Vec	LSTM	86.476%
40	Word2Vec	GRU	85.516%
41	Gensim Word2Vec	LSTM	84.736%
42	Gensim Word2Vec	GRU	84.728%
43	None(Embedding)	LSTM	83.216%
44	None(Embedding)	GRU	86.404%

Note: The above table just gives the slight idea of how different representation work with different classification algorithms. It is no way state the best performance for that representation since **NO** parameter tuning was done to improve accuracy.

Few points observed from above table:

- *TF-IDF* representation of documents, although being a bag of words approach, gives the best accuracy compared to other approaches.
- In case of word vector averaging representation, our learned gensim word vectors outperform both pre-trained Gensim and Word2Vec vectors. The reason for this is that the pre-trained word vectors are trained on the different corpus and therefore are not fine-tuned for sentiment related corpus.
- Sentence Vector averaging model implemented using Doc2Vec algorithm did not perform quite well compared to Paragraph Vector model.
- LSTM and GRU both gave almost the same accuracy