NLP skill testing project: Movie review sentiment analysis

Objective: Download the Large Movie Review dataset (http://ai.stanford.edu/~amaas/data/sentiment/) and train models to predict the sentiment of the review (positive or negative). The dataset contains 25000 labeled reviews for training, and 25000 for testing.

Background: This type of task should be familiar to you. It is a binary classification problem, where the input data is text. Sentiment analysis is a very common NLP application.

How to begin: We've provided you with three modules containing starter code/templates. They are *data_preprocessing.py*, *models.py* and *execute.py.* In each module, you'll find function headers, along with descriptions of what each function should do (in the docstrings beneath). Add functions and modify the function templates as needed.

Step 1: load and explore the raw data

Fill out the **load_dataset()** function in *data_preprocessing.py* to load the movie review data. The function should return the training and testing data as python lists (where each item is a review), as well as their corresponding labels (optionally, encoded in one-hot format). Take a look at some of the reviews to get a sense of the data you're dealing with.

Step 2: classify the data with LSA + LogisiticRegression

Using the template found in *models.py,* define **LSATextClassifier()**. Remember, LSA is just SVD applied to TF-IDF vectors. Using the dense vectors created using LSA as inputs, classify the reviews with a logistic regression model. Most of the heavy lifting can be done with the **sklearn** library.

Once the model is complete, modify *execute.py* accordingly and run it with
$ python3 execute.py LSATextClassifier

You can achieve a surprisingly high accuracy with such a simple model. With little optimization, you can achieve an accuracy above 85%; this turns out to be tough to beat with deep learning approaches.

Once you reach this point, send your code to @yazabi and we'll give you feedback on it!

Step 3: create word embeddings from the data

Fill out **make_embedding_matrix()** to create word embeddings from the movie review data. Use any approach you feel comfortable with, but the python package **gensim** is recommended as it provides a good API for word embeddings. You'll likely need to call **tokenize()** to tokenize the text into words and filter it (e.g. remove punctuation and stop words). The python package **nltk** has a lot of useful functions for text preprocessing.

As this step can be computationally/time expensive, it is a good idea to save your embeddings and use **load_embedding_matrix()** to load it and save time as you tweak your models.

The **embedding_matrix** is essentially a look-up table for converting words to word-vectors. Fill out **to_word_vectors()** to convert text sample(s) into sequence(s) of word-vectors. Limit the sequence length to a fixed value and pad shorter reviews with zeros and truncate longer reviews.

Finally, complete *data_preprocessing.py* by filling out **batch_generator()** which will be called to create batches of data/labels when training/testing your deep learning models.

Step 4: classify the data using a CNN

Using the template found in *models.py,* define **CNNTextClassifier().** The model should take a sequence of word-vectors as its input. Use any architecture you want; for an idea of where to start, check out this paper. The authors apply several 1d convolutions (essentially word-level filters) with global max-pooling which feed into a dense layer with softmax activation.

Code this in pure TensorFlow or use Keras. the links in ***CNNS for NLP*** should help you get started on either.

Note: 1d convolution is just a 2d convolution with one dimension set to unity. (TensorFlow has a conv1d function, which is simply a convenience function wrapped around conv2d.)

Once the model is complete, modify *execute.py* accordingly and run it with
$ python3 execute.py CNNTextClassifier

Once you reach this point, send your code to @yazabi and we'll give you feedback on it!

Step 5: classify the data using a LSTM network

Using the template found in *models.py,* define **LSTMTextClassifier().** The model should take a sequence of word-vectors as its input. You can code this in TensorFlow, but Keras can save loads of time. See keras' sequential model (hint: check out the examples).

Once the model is complete, modify *execute.py* accordingly and run it with
$ python3 execute.py RNNTextClassifier

Once you reach this point, send your code to **@yazabi** and we'll give you feedback on it!