Midterm Assignment – Natural Language Processing

Sentiment Analysis using Word Embeddings – Report

Bhavesh Waghela – N01639685

1. Import the required Libraries.

   Libraries such as numpy, pandas, from gensim.models – word2vec, matplotlib, seaborn, logging were imported into the ipynb file.

2. Data Preparation.
   a. Read the tsv file provided that is 'labledTrainingData' and 'unlabledTrainingData'. Given Information: There are 25,000 reviews in the training set and 50,000 reviews in the unlabeled dataset.
   b. Check the shape of the dataset provided.
   c. Check the review sample and identify unwanted characters such as HTML tags and special characters and remove them from the entire dataset. (75,000)
      i. Convert each review into sentences using 'punkt' tokenizer from NLTK.
      ii. Remove the HTML tags and replace with space, '- ' replace them with spaces and '.' with '. ' a full stop and a space for sentence separation. Replacing double space with single space.
      iii. Apply special_character function filter on each word of the sentence and remove if any.

3. Word2vec embedding.
   a. Value Definition and Model Creation - Word2Vec function requires various parameters for it to run that are sentence, min word count, window size, sample, workers. Its creates a model using those parameters.
   b. Save the model, Loading the model and check the word count.
   c. Evaluate the model's performance by checking word similarity.
      i. 1 and 2 are near each other. – 0.75
      ii. 'apple' and 'weather' are away from each other. – 0.28
   d. Check for the most similar words to 'character'
      [('role', 0.6837882995605469),
      ('protagonist', 0.6619493961334229),
      ('villain', 0.6557775139808655)…]
   e. Find the missing word – Use of the word2vec model created when given positive and negative input words and find the missing word.
      i. "good: best :: bad: ? " we get ('worst', 0.7713937163352966).
      ii. "Berlin : Germany :: France : ?" we get ('paris', 0.7268273234367371).

   f. Different from Group – function finds which word dosenot belong to the group of words eg. "terrible bad horrible good" – good is picked as it dose not belong to same group.

g. Word vector – Get a vector representation of any word.
h. Word index – Find a word at a specific index location.
i. Index to Word – Get the index location of the word by looking up the vocabulary.

4. Saving the model to array of words and vectors.
    a. The words in the vocabulary and the respective vectors are saved into 2 different arrays for better manipulation.
    b. Any word that is not found in the list will be linked with '-unk' and its respective vector will be assigned a value of 0.
    c. Find the number of words in the word_list and word_vector shape.
        i. Number of words in list: 24802
        ii. Shape of word vector: (24802, 100)
    d. Get the first word from the vocab list and its respective vector value.
    e. Get the last word from the vocab list and its respective vector value.
    f. Check if the saved values are consistent with the original model values - Results of the loaded word vector and the model word vector for the word 'movie' should give us the results as 0 when subtracted from each other.

5. Fast Text embedding algorithm with Gensim.
    a. Training the word embeddings – FastText model requires various parameters as inputs for it to execute and create a model, they are as follows: sentences, vector size, window size, min word count, workers, minimum length of characters in n-grams, maximum length of characters in n-grams.
    b. Check for the most similar words to 'character' we get the following results.

        [('characterisation', 0.8808436989784241),
        ('characterize', 0.8790835738182068),
        ('characteristically', 0.8554302453994751),
        ('characterization', 0.8502060770988464),
        ('characteristic', 0.8450654745101929),..]
    c. Comparing and trying to find the same word 'citi' in word2vec model and in fastText Model – The word did not exist in word2vec model and in did give a vector output for fastText Model.
    d. FastText learns word representation better than `word2vec` because it breaks word into sub-words, making the learning more efficient. Due to which it can find better similar words.
    e. Output for most_similar function for 'citi' in fastText Model gives the following results.

        [('citizens', 0.8885571956634521),
        ('citizen', 0.8578288555145264),
        ('europa', 0.8339913487434387),
        ('america', 0.8214999437332153),
        ('citys', 0.8171120882034302),..]
    f. Find the missing word – Use of the fastText model created when given positive and negative input words and find the missing word.

    i. “good: best :: bad: ? “ we get ('worst', 0.8682386875152588).
  g. It has been noted that, `fastText` model doesn't work well with analogy tasks in some cases whereas the `word2vec` model is able to find better match from the list.
  h. Different from Group – function finds which word dosenot belong to the group of words. eg. “terrible bad horrible good” – good is picked as it dose not belong to same group. *Both of this model has given same results form this function.*

6. Saving the model to array of words and vectors.
  a. The words in the vocabulary and the respective vectors are saved into 2 different arrays for better manipulation.
  b. Any word that is not found in the list will be linked with ‘-unk’ and its respective vector will be assigned a value of 0.
  c. Find the number of words in the word_list and word_vector shape.
    i. Number of words in list: 24802
    ii. Shape of word vector: (24802, 50)

7. Findings and Recommendations:
  a. FastText Models can be used where the problem is more focused on finding the similar words for let’s say a half word line ‘citi’ this is not considered a word in word2vec model whereas in fastText model we could see some good results as it was considering it as a broken word and was able to give complete words what was meaningful.
  b. Word2vec was able to perform extremely well in the analogy task as it would very well identify the capital of the provinces where as extremely bad results were produced by fastText model for the same set of inputs.
  c. Based on the use case defined above we must use word2vec or fastText respectively.