CSC791 NLP P2: Response Classification in discussions

Shreyas Chikkballapur Muralidhara - schikkb

Report Requirement:

1. Include the following features as your baseline model – word/sentence embedding, POS tagging.
2. You should experiment with multiple classification models (such as SVM or neural-network-based classifiers) and choose the best to report results for. You are required to report the results for the following feature sets:
   - set 1: only baseline features
   - set 2: baseline features + **two** additional features (that you proposed)
3. Evaluation: accuracy, precision, recall and F1 score
4. Based on the performance of the model on the two feature sets, analyze why your proposed additional features contribute or not to type classification.

Solution:

For the Response classification in discussions, I have used the 2 suggested features as baseline model 1, 2 additional features added to baseline model 1 as model 2.

set 1–

- **POS tagging using nltk.pos_tag**
- **sentence embedding using Doc2Vec**

Set 2 **–**

- **POS tagging using nltk.pos_tag**
- **Named Entity Recognition (NER) using spacy**
- **Sentimental analysis using Textblob**
- **Sentence vector embedding using Doc2Vec.**

Source Code link from google colab -
https://colab.research.google.com/drive/1jTz4xSSkSuNZnsCPHSAPnBcjOzc8NWaH?usp=sharing

PERFORMANCE and METRICS –

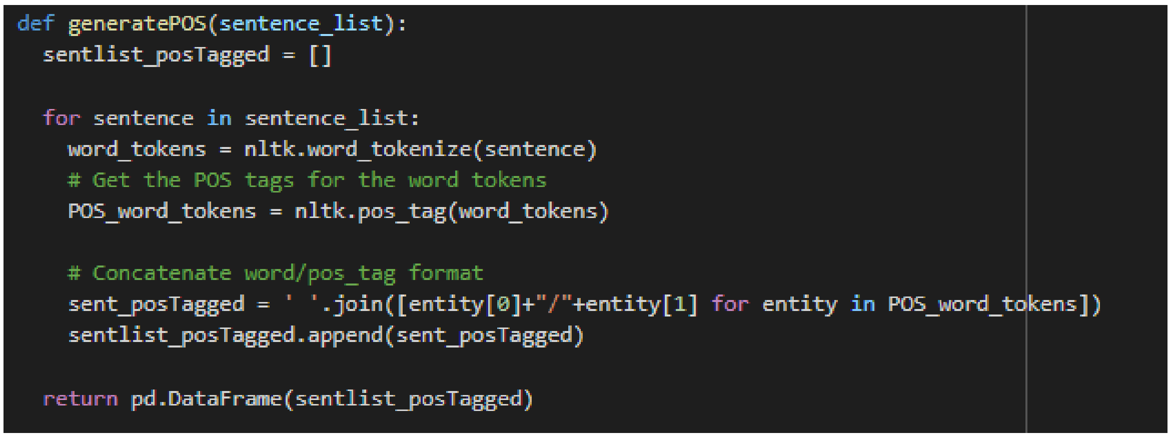
Baseline model – set1

Feature 1 - POS Tagging: It was implemented using nltk.pos_tag. Each sentence of question list and response list were updated to new version of each word replaced by **word/POS tag.** Associating word with the POS tags provides better sentence vectorization hence this feature is expected to have better accuracy.

Example: *Should all the blacks in the state move out?*
POS tagged sentence: *Should/MD all/PDT the /DT blacks/NNS in/IN the/DT state/NN move/NN out/IN ?/.*

Generating the vector embeddings for **POS tagged sentences** provided better accuracy than directly generating the vector embedding from the **sentences**. The validation accuracy increased from 53.65% to 56.1% accuracy. Even th precision for the attacked and irrelevant increased as well.

Implementation:

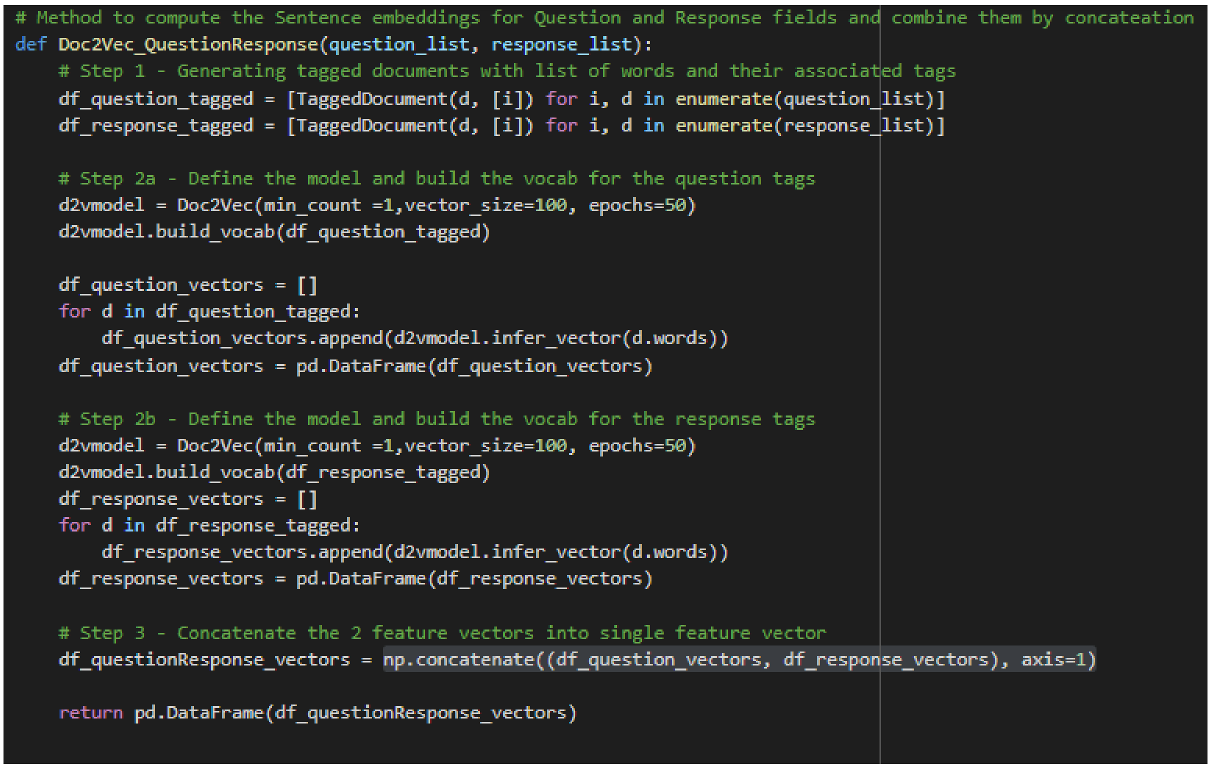


```python
def generatePOS(sentence_list):
  sentlist_posTagged = []

  for sentence in sentence_list:
    word_tokens = nltk.word_tokenize(sentence)
    # Get the POS tags for the word tokens
    POS_word_tokens = nltk.pos_tag(word_tokens)

    # Concatenate word/pos_tag format
    sent_posTagged = ' '.join([entity[0]+"/"+entity[1] for entity in POS_word_tokens])
    sentlist_posTagged.append(sent_posTagged)

  return pd.DataFrame(sentlist_posTagged)
```

Feature 2 – sentence embedding using Doc2vec: Doc2Vec which is inspired from word2Vec, is one of the most popular models for sentence vectorization. It generates the sentence vectors from the tagged document. This was implemented using **Doc2vec, TaggedDocument** from the **genism.models.doc2vec** module. The hyper parameters vector size was 100 dimensions and min count was considered as 1 occurrence in the document, with epoch as 50.

Sentence vectors were generated for the **questions** and the **responses** separately of size (1640, 100) each. Later they were concatenated to generate a single vector embedding of size (1640,200).

Implementation:



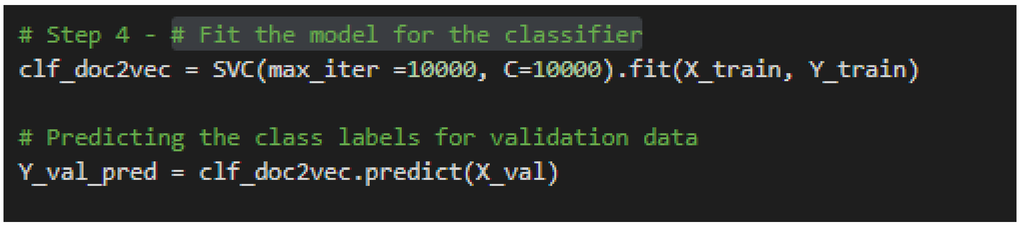
```python
# Method to compute the Sentence embeddings for Question and Response fields and combine them by concateation
def Doc2Vec_QuestionResponse(question_list, response_list):
    # Step 1 - Generating tagged documents with list of words and their associated tags
    df_question_tagged = [TaggedDocument(d, [i]) for i, d in enumerate(question_list)]
    df_response_tagged = [TaggedDocument(d, [i]) for i, d in enumerate(response_list)]

    # Step 2a - Define the model and build the vocab for the question tags
    d2vmodel = Doc2Vec(min_count =1,vector_size=100, epochs=50)
    d2vmodel.build_vocab(df_question_tagged)

    df_question_vectors = []
    for d in df_question_tagged:
        df_question_vectors.append(d2vmodel.infer_vector(d.words))
    df_question_vectors = pd.DataFrame(df_question_vectors)

    # Step 2b - Define the model and build the vocab for the response tags
    d2vmodel = Doc2Vec(min_count =1,vector_size=100, epochs=50)
    d2vmodel.build_vocab(df_response_tagged)
    df_response_vectors = []
    for d in df_response_tagged:
        df_response_vectors.append(d2vmodel.infer_vector(d.words))
    df_response_vectors = pd.DataFrame(df_response_vectors)

    # Step 3 - Concatenate the 2 feature vectors into single feature vector
    df_questionResponse_vectors = np.concatenate((df_question_vectors, df_response_vectors), axis=1)

    return pd.DataFrame(df_questionResponse_vectors)
```

CSC791 NLP P2: Response Classification in discussions

Shreyas Chikkballapur Muralidhara - schikkb

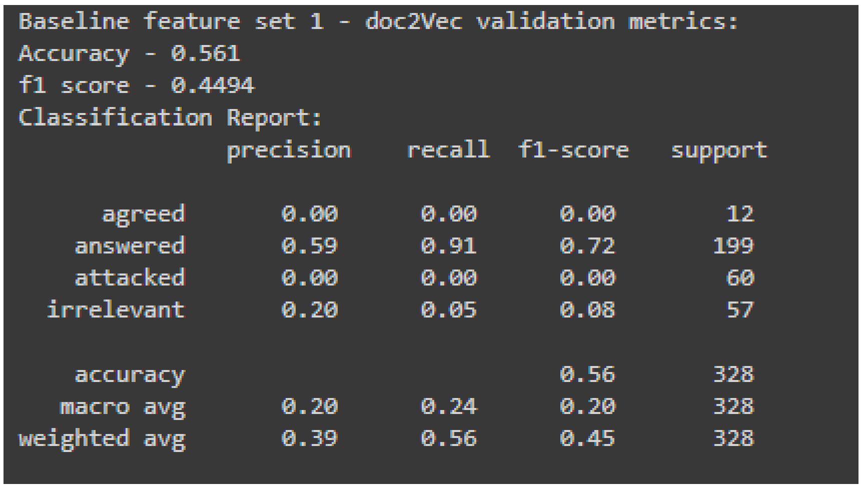
Classifier – Support Vector Machine - SVC(radial bias kernel): The vector embeddings [(1640,200) – (1312,200) training and (328,200) validation] generated by Doc2Vec was used to train the SVC model with Radial Bias kernel. The hyperparameters max iter = 10000 and C = 10000.

Implementation:



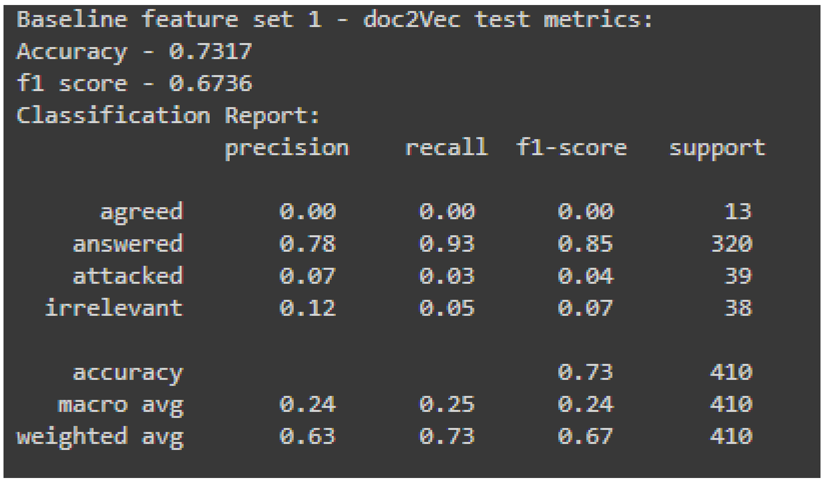
```
# Step 4 - # Fit the model for the classifier
clf_doc2vec = SVC(max_iter =10000, C=10000).fit(X_train, Y_train)

# Predicting the class labels for validation data
Y_val_pred = clf_doc2vec.predict(X_val)
```

Set 1 Validation Metrics – Baseline model of set1 reported the validation accuracy of **56.1%** and F1 score of **44.94%**. Below are the details of accuracy, Precision, recall and F1 score for the baseline model – set 1:



```
Baseline feature set 1 - doc2Vec validation metrics:
Accuracy - 0.561
f1 score - 0.4494
Classification Report:
              precision    recall  f1-score   support

     agreed       0.00      0.00      0.00        12
   answered       0.59      0.91      0.72       199
   attacked       0.00      0.00      0.00        60
 irrelevant       0.20      0.05      0.08        57

   accuracy                           0.56       328
  macro avg       0.20      0.24      0.20       328
weighted avg      0.39      0.56      0.45       328
```

Set 1 Test Metrics – Baseline model of set1 reported the test accuracy of **73.17%** and F1 score **67.36%**. Below are the details of accuracy, Precision, recall, F1 score for the baseline model–set 1:



```
Baseline feature set 1 - doc2Vec test metrics:
Accuracy - 0.7317
f1 score - 0.6736
Classification Report:
              precision    recall  f1-score   support

     agreed       0.00      0.00      0.00        13
   answered       0.78      0.93      0.85       320
   attacked       0.07      0.03      0.04        39
 irrelevant       0.12      0.05      0.07        38

   accuracy                           0.73       410
  macro avg       0.24      0.25      0.24       410
weighted avg      0.63      0.73      0.67       410
```

CSC791 NLP P2: Response Classification in discussions

Shreyas Chikkballapur Muralidhara - schikkb

Set 2: baseline features + **two** additional features

Baseline model – set1

Feature 1 - POS Tagging: It was implemented using nltk.pos_tag. Each sentence of question list and response list were updated to new version of each word replaced by **word/POS tag.** Associating word with the POS tags provides better sentence vectorization hence this feature is expected to have better accuracy.

Example: *Should all the blacks in the state move out?*
POS tagged sentence: *Should/MD all/PDT the /DT blacks/NNS in/IN the/DT state/NN move/NN out/IN ?/.*

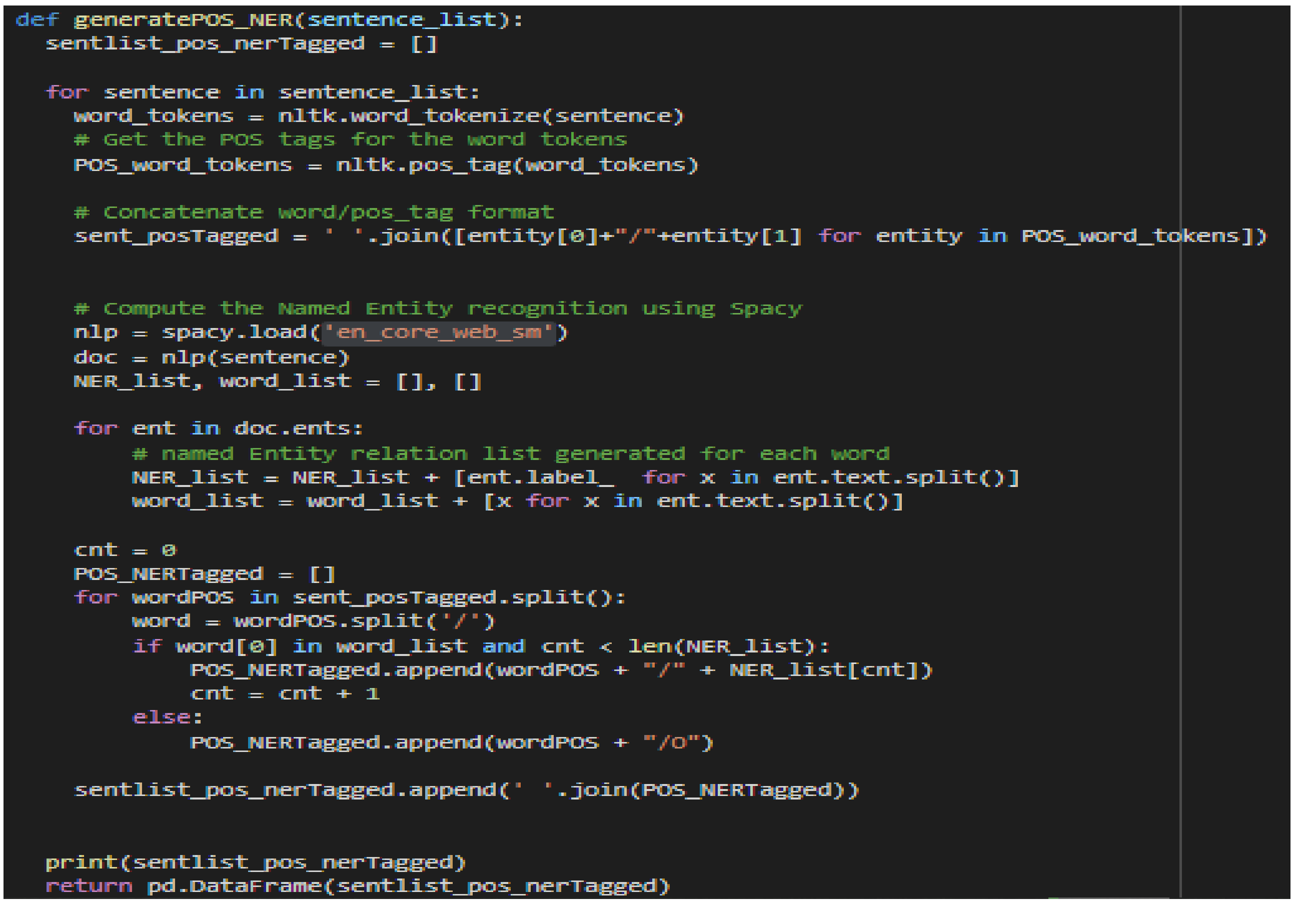
Generating the vector embeddings for **POS tagged sentences** provided better accuracy than directly generating the vector embedding from the **sentences**. The validation accuracy increased from 53.65% to 56.1% accuracy. Even the precision for the attacked and irrelevant increased as well.

Feature 2 – Named Entity Recognition NER: It was implemented using *spacy.load('en_core_web_sm')*. For each sentence, Named entity recognition by spacy provided entity label for specific word group. This was modified to assign the entity label to all the elements in the group. For those elements which didn't get the NER tag '/O' was concatenated.

Example: *And men commit more crimes than women, ages 20 to 60 commit more crimes than those above 60, the poor more likely to commit crimes than the rich, etc - so why not export one of these groups?*
NER and POS tagged sentence: *'And/CC/O men/NNS/O commit/VBP/O more/JJR/O crimes/NNS/O than/IN/O women/NNS/O ,/,/O ages/VBZ/O 20/CD/CARDINAL to/TO/CARDINAL 60/CD/CARDINAL commit/NN/O more/JJR/O crimes/NNS/O than/IN/O those/DT/O above/IN/O 60/CD/CARDINAL ,/,/O the/DT/O poor/JJ/O more/RBR/O likely/JJ/O to/TO/O commit/VB/O crimes/NNS/O than/IN/O the/DT/O rich/JJ/O ,/,/O etc/FW/O -/:/O so/RB/O why/WRB/O not/RB/O export/VB/O one/CD/O of/IN/O these/DT/O groups/NNS/O ?/./O'*

This tagging improved the scores as the words could be associated with specific NER in one sentences and different in another. Hence it gives better clarity in word with tags generating different vectors than compared to set1. The validation accuracy improved from 56.1% to 58.3%. But since not all words could have the NER tags, the impact was quite not as good as expected.

Implementation of Feature 1 and Feature 2:

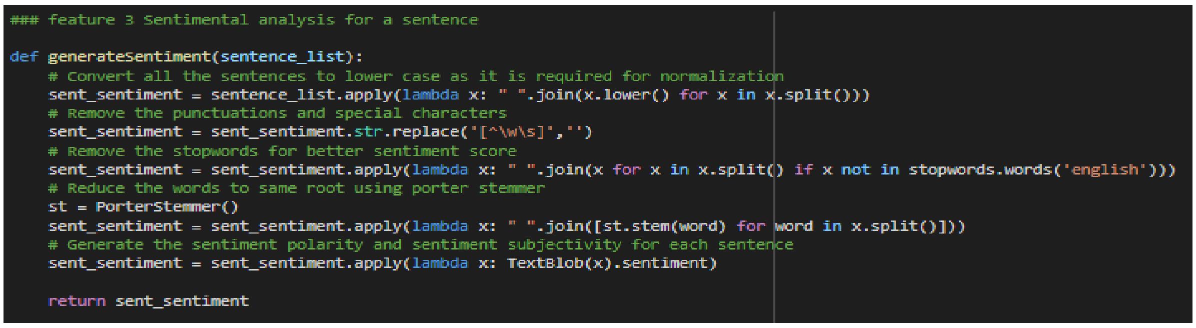


```python
def generatePOS_NER(sentence_list):
  sentlist_pos_nerTagged = []

  for sentence in sentence_list:
    word_tokens = nltk.word_tokenize(sentence)
    # Get the POS tags for the word tokens
    POS_word_tokens = nltk.pos_tag(word_tokens)

    # Concatenate word/pos_tag format
    sent_posTagged = ' '.join([entity[0]+"/"+entity[1] for entity in POS_word_tokens])


    # Compute the Named Entity recognition using Spacy
    nlp = spacy.load('en_core_web_sm')
    doc = nlp(sentence)
    NER_list, word_list = [], []

    for ent in doc.ents:
        # named Entity relation list generated for each word
        NER_list = NER_list + [ent.label_  for x in ent.text.split()]
        word_list = word_list + [x for x in ent.text.split()]

    cnt = 0
    POS_NERTagged = []
    for wordPOS in sent_posTagged.split():
        word = wordPOS.split('/')
        if word[0] in word_list and cnt < len(NER_list):
            POS_NERTagged.append(wordPOS + "/" + NER_list[cnt])
            cnt = cnt + 1
        else:
            POS_NERTagged.append(wordPOS + "/O")

    sentlist_pos_nerTagged.append(' '.join(POS_NERTagged))


  print(sentlist_pos_nerTagged)
  return pd.DataFrame(sentlist_pos_nerTagged)
```

Feature 3 – Sentimental Analysis for the question and response inputs: sentiment scores both polarity and subjectivity were generated using the TextBlob().sentiment. Since the sentences had noise and we need better accuracy score, there were a couple of pre-processing steps that was performed. Convert the sentences to lower case, removal of punctuations and special characters. Removal od frequently occurring stopwords. Reduce the word forms to single root using the porter stemmer. Finally get the sentiments by polarity and subjectivity.

This process was repeated for Question and responses, later these scores were concatenated with the sentence vectors to generate embedding with sentiment scores. Since not all sentences had sentiments while few did have, it didn't make significant difference. The validation accuracy of the model increased from 58.3% to 59.15% after implementing this feature.

Implementation:



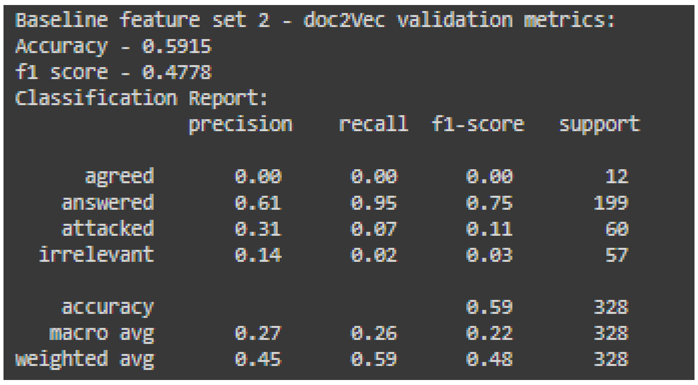
```python
### feature 3 Sentimental analysis for a sentence

def generateSentiment(sentence_list):
    # Convert all the sentences to lower case as it is required for normalization
    sent_sentiment = sentence_list.apply(lambda x: " ".join(x.lower() for x in x.split()))
    # Remove the punctuations and special characters
    sent_sentiment = sent_sentiment.str.replace('[^\w\s]','')
    # Remove the stopwords for better sentiment score
    sent_sentiment = sent_sentiment.apply(lambda x: " ".join(x for x in x.split() if x not in stopwords.words('english')))
    # Reduce the words to same root using porter stemmer
    st = PorterStemmer()
    sent_sentiment = sent_sentiment.apply(lambda x: " ".join([st.stem(word) for word in x.split()]))
    # Generate the sentiment polarity and sentiment subjectivity for each sentence
    sent_sentiment = sent_sentiment.apply(lambda x: TextBlob(x).sentiment)

    return sent_sentiment
```

Feature 4 – sentence embedding using Doc2vec: Doc2Vec which is inspired from word2Vec, is one of the most popular models for sentence vectorization. It generates the sentence vectors from the tagged document. This was implemented using **Doc2vec, TaggedDocument** from the **genism.models.doc2vec** module. The hyper parameters vector size was 100 dimensions and min count was considered as 1 occurrence in the document, with epoch as 50.

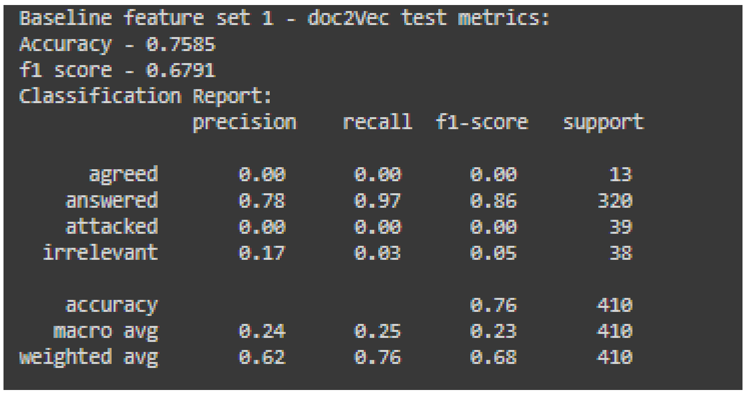
Sentence vectors were generated for the **questions** and the **responses** separately of size (1640, 100) each. Later they were concatenated to generate a single vector embedding of size (1640,200).

Implementation: Exact same implementation as set 1-Feature 2.

Set 2 Validation Metrics – Baseline model of set2 with 2 additional features reported the validation accuracy of **59.15%** and F1 score of **47.78%**. Below are the details of accuracy, Precision, recall and F1 score for the baseline model – set 2 with 2 additional features:


```
Baseline feature set 2 - doc2Vec validation metrics:
Accuracy - 0.5915
f1 score - 0.4778
Classification Report:
              precision    recall  f1-score   support

      agreed       0.00      0.00      0.00        12
    answered       0.61      0.95      0.75       199
    attacked       0.31      0.07      0.11        60
  irrelevant       0.14      0.02      0.03        57

    accuracy                           0.59       328
   macro avg       0.27      0.26      0.22       328
weighted avg       0.45      0.59      0.48       328
```

Set 2 Test Metrics – Baseline model of set2 reported the test accuracy of **75.85% raising from 73.17%** and F1 score **67.91% raising from 67.31%**. Below are the details of accuracy, Precision, recall and F1 score for the baseline model – set 2 with 2 additional features:


```
Baseline feature set 1 - doc2Vec test metrics:
Accuracy - 0.7585
f1 score - 0.6791
Classification Report:
              precision    recall  f1-score   support

      agreed       0.00      0.00      0.00        13
    answered       0.78      0.97      0.86       320
    attacked       0.00      0.00      0.00        39
  irrelevant       0.17      0.03      0.05        38

    accuracy                           0.76       410
   macro avg       0.24      0.25      0.23       410
weighted avg       0.62      0.76      0.68       410
```