## Baseline solution

My baseline solution consists of the following steps:

1. Read data from file.
2. Tokenize sentences if required.
3. Vectorize sentences using either word2vec or TF-IDF.
4. Classify sentence vectors into labels of 0,1 or 2.

For the two vectorization methods I chose to use word2vec and TF-IDF.

### Word2vec:

For my word2vec baseline model, I used the word2vec method from Spacy. As outlined in the steps above, I vectorized the sentences using the word2vec model downloaded from Spacy and then used a random forest classifier to classify the sentences.

1. Tokenization: As the Spacy library's word2vec method performs tokenization on its own, I did not tokenize the sentences separately beforehand.
2. Word embeddings: I used a downloaded model from spacy to implement word2vec and get word embeddings. The model obtains word embeddings and then averages them to get the sentence embeddings (from my understanding). The output is a vector for each sentence.

```
#vectorize text using word2vec
def word2vec_vectorize(sentences):
    nlp = spacy.load("en_core_web_sm")
    #process sentences using model
    vectors = []
    for sent in sentences:
        temp = nlp(sent)
        vectors.append(temp.vector)
    return vectors
```

3. Classification: I used a random forest classifier (scikit-learn) to label the sentences as 0, 1 or 2.

```
#Random forest classifier
def randomforest(X_train, y_train, X_test, y_test):
    print(X_test)
    model = RandomForestClassifier(n_estimators=200, random_state=0)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    f1score = f1_score(y_test, y_pred, average='micro')
    print('Accuracy: '+str(round(accuracy,3)))
    print('F1-score: '+ str(round(f1score, 3)))
    print(classification_report(y_test, y_pred))
    return accuracy, y_pred
```

Results:

```
(base) devikrishnan@Devis-MBP P1_Final % python3 word2vec.py
Accuracy: 0.507
F1-score: 0.507
/Users/devikrishnan/anaconda3/lib/python3.8/site-packages/sklearn
icted samples. Use `zero_division` parameter to control this beha
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        82
           1       0.50      0.65      0.57       303
           2       0.52      0.50      0.51       298

    accuracy                           0.51       683
   macro avg       0.34      0.38      0.36       683
weighted avg       0.45      0.51      0.47       683
```

The word2vec baseline model performed with an accuracy of 0.507. In terms of other metrics, the recall for the label value of 1 was better than that of the value 2 and 0, but similar to 2 in terms of precision. Some reasons why the accuracy is so low could be:

- Lack of inter-sentence context: Since the word2vec vectorization model is computing word embeddings and then averaging them, the context between sentences may be lost when the word embeddings are averaged to get sentence embeddings. For example, “I am lost” may be computed to be the same as “Lost I am”.
- Model size: the small model size used may not have some of the words in the vocabulary created from the dataset.

TF-IDF:

I implemented TF-IDF for vectorization using the scikit-learn module in Python. I performed tokenization separately although the package is capable of performing tokenization on its own as well. The details are shown below:

1. Tokenization: I tokenized the sentences using NLT-K’s word_tokenize() tool. I did not perform any other text preprocessing before tokenizing.

```
#tokenize text
sentences = data['sentence'].tolist()
tokenized_sent = [word_tokenize(sentence) for sentence in sentences]
```

2. Vectorization: I implemented TF-IDF from scikit-learn's library. I used a "dummy" tokenizer since I had already tokenized the sentences. In addition, I specified smooth_idf as False. If a word was not seen before in the vocabulary and is seen in the test data, setting smooth_idf allows it to be processed without a zero division error. Initially I set it to True but setting it to False gave me a slightly higher accuracy.

```
#vectorize text using TF-IDF
def tfidf_vectorize(tokenized_sent):
    #dummy tokenizer
    def tokenizer(text):
        return text

    vectorizer = TfidfVectorizer(tokenizer=tokenizer, lowercase=False, norm=False, smooth_idf=False)
    vectors_list = vectorizer.fit_transform(tokenized_sent)
    vectors = list(vectors_list.toarray())
    return vectors
```

3. Classification: I used the same random forest classifier used in my word2vec model here.

```
#Random forest classifier
def randomforest(X_train, y_train, X_test, y_test):
    model1 = RandomForestClassifier(n_estimators=200, random_state=0)
    model1.fit(X_train, y_train)
    y_pred = model1.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    f1score = f1_score(y_test, y_pred, average='micro')
    print('Accuracy: '+str(round(accuracy,3)))
    print('F1-score: '+ str(round(f1score, 3)))
    print(classification_report(y_test, y_pred))
    return accuracy, f1score, y_pred
```

Results:

```
(base) devikrishnan@Devis-MBP P1_Final % python3 tfidf.py
Accuracy: 0.559
F1-score: 0.559
/Users/devikrishnan/anaconda3/lib/python3.8/site-packages/sklear
icted samples. Use `zero_division` parameter to control this beh
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        82
           1       0.53      0.80      0.64       303
           2       0.62      0.47      0.54       298

    accuracy                           0.56       683
   macro avg       0.38      0.42      0.39       683
weighted avg       0.51      0.56      0.52       683
```

The TF-IDF baseline model performs with a higher accuracy than the word2vec model, with an accuracy of 0.56, or 56%.

## Proposed Model

I have used the BERT model before for a text summarization task and I thought it would be interesting to apply it to sentiment analysis and see how it performs. My proposed model makes use of BERT for extracting word embeddings from the text. I used a version called sentence-BERT since it is easy to get sentence vectors using this method. The BERT model I chose to use was bert-base-nli-stsb-mean-tokens, which was pretrained on NLI+STSb datasets and then had mean pooling performed on the output. I followed similar steps to my baseline model in other respects.

1. Tokenization: BERT performs its own tokenization and other preprocessing steps, so I did not perform these steps separately.
2. Word Embeddings: I used BERT from the sentence-transformers library in python, which implements BERT specifically for getting sentence embeddings. The BERT model itself is available online, so i did not need to download the model.

```python
#vectorizer
def bert_vectorize(sentences):
    #load pretrained BERT model
    model = SentenceTransformer('bert-large-nli-stsb-mean-tokens')
    #encode sentences
    vectors = model.encode(sentences)
    return list(vectors)
```

3. Classification: To classify the sentences I again used random forest from scikit-learn since decision tree classifiers usually work well for multiclass classification problems.

```
#Random forest classifier
def randomforest(X_train, y_train, X_test, y_test):
    print(X_test)
    model = RandomForestClassifier(n_estimators=200, random_state=0)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    f1score = f1_score(y_test, y_pred, average='micro')
    print('Accuracy: '+str(round(accuracy,3)))
    print('F1-score: '+ str(round(f1score, 3)))
    print(classification_report(y_test, y_pred))
    return accuracy, y_pred
```

Results:

```
(base) devikrishnan@Devis-MacBook-Pro P1_Final % python bert.py
Running bert_vectorize...
Accuracy: 0.654
F1-score: 0.654
              precision    recall  f1-score   support

           0       0.60      0.04      0.07        82
           1       0.61      0.81      0.69       303
           2       0.73      0.66      0.69       298

    accuracy                           0.65       683
   macro avg       0.64      0.50      0.49       683
weighted avg       0.66      0.65      0.62       683

(base) devikrishnan@Devis-MacBook-Pro P1_Final %
```

The BERT model showed a significant improvement in accuracy over the two baseline models. This could be attributed to the following:

- Preprocessing: BERT performs its own preprocessing, including tokenization, stop word filtering, stemming/lemmatization, etc.
- Pretraining: BERT is pretrained on a huge corpus of unlabelled Wikipedia articles plus the Book Corpus. As a result the model has a deep understanding of the language and can perform on many NLP tasks with a good accuracy.

These factors give the BERT model a boost in accuracy, its accuracy being 14% more than that of the word2vec model and 9% more than that of the tf-idf model.