Name: Changhyun, Lee
Andrew ID: changhyl

Algorithm for NLP
Homework 4

1. Naïve Bayes

1.1. How did you use the dev data?

I split the dev data into training and testing data with rate of 7:3 because it is widely used portion for splitting data into training and testing data.

1.2. How did you preprocess the data?

First, we used a function below,

```
text = re.sub("[^a-zA-Z]", " ", sentences[i])
```

This line above deletes the character that does not be included from a to z or from A to Z. I intended to delete some kinds of HTML language from our dataset.

Second, I utilized the nltk library to loads a set of stop words like below, then removed the stop words if the data includes it.

```
stops = set(stopwords.words("english"))
words = [w for w in words if not w in stops]
```

However, when I tried to apply this preprocessing step, the accuracy on my test set shows 61.1666%. Removing stop words from my dataset deteriorated the performance of Naïve Bayes classifier. Probably it is because in the set of stop words, there are some meaningful words which affect to estimate positive or negative sentence.

1.3. For each feature you tried,

- What is the feature?

  Unigram, top 10000 words in bag of words.

- Why do you think it makes sense to use this feature?

  I used simply Unigram feature. I can also utilize such kind of bigram, trigram, etc. but the difference will turn out as smoothing effect. I think the difference is also in bias and variance trade-off. (Unigrams : high bias/low variance, Bigrams : low bias/high variance, and so forth)

- Did the feature improve accuracy, either on the heldout or on your own test set?

  About Unigram and other features, I didn't try. However, regarding the number of words in bag of words, I tried the comparison between top 15000 words and 10000 words. Naïve Bayes classifier with top 15000 words showed 63% of test accuracy and the classifier with top 10000 words showed 67.33% of test accuracy.

- Why do you think it did or did not improve accuracy?

  I guess it is because when the number of words in the bag of words increase, the probability of one word exist in the bag of words decrease.

1.4. Final accuracy on my test set

67.3333%

2. Neural Networks

2.1. How did you use the dev data, if different from Task 1?

I divided the dev data into training and testing data with rate of 0.7, same as Task 1.

2.2. How did you preprocess the data, if different from Task 1?

I used default preprocessing step by using tokenizer, when I change our texts into sequence of index.

```
from tensorflow.python.keras.preprocessing.text import
Tokenizer
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
```

2.3. What is the architecture of your model?

2.4. What does the model take as input?

I used word embedding to extract dense representations from words. Using Tokenizer which is in tensorflow library, I made our text data into sequence of index of words. Then, zero padding is applied to fit the features same length. When we use tokenizer.fit_on_texts function, we should know that the function provide removing stop words. Finally the features is fed into the embedding layer of CNN model. I attached related code below.

```
model.add(Embedding(vocab_size, embedding_vector_length,
input_length=MAX_SEQUENCE_LENGTH, trainable=True))
```

2.5. What features did you try?

I changed the number of words in our bag of words from top 15000 to top 10000.

CNN classifier with top 15000 words showed 82.33% of test accuracy, and the classifier with top 10000 words showed 81.33% of test accuracy.

2.6. Any non-trivial extensions you made to the model

```
model = Sequential()
model.add(Embedding(vocab_size, embedding_vector_length,
input_length=MAX_SEQUENCE_LENGTH, trainable=True))
model.add(Conv1D(100, 3, padding='valid', activation='relu',
strides=1))
model.add(Conv1D(50, 3, padding='valid', activation='relu',
strides=1))
model.add(GlobalMaxPooling1D())
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['acc'])
model.summary()
```

First I tried the model above, but the accuracy was 76.00%. However, when I undergo trial and error, 82.33% is achieved by making the model simple. I removed the lines marked as yellow.

2.7. What is the final accuracy on your own test set and how is it compared to your Naïve Bayes classifier?

82.33%, it is much higher than the Naïve Bayes classifier.

2.8. Qualitative analysis on the model outputs and compare it with ones from your Naïve Bayes classifier

Naïve Bayes classifier shows 67.33% accuracy and CNN classifier shows 83.33% accuracy. It is about 16% difference in accuracy.