Sentiment Analysis

Learn about sentiment analysis and how it relates to NLP.

Chapter Goals:

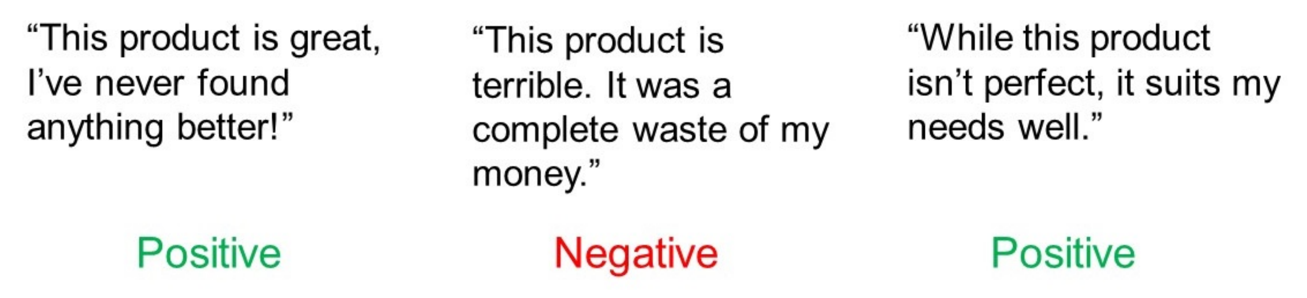
- Learn about sentiment analysis and how it's related to NLP
- Create sentiment analysis training pairs from a text corpus

A. Classifying sentiment

Any time you read an opinionated text passage (e.g. social media post, editorial, etc.), the text contains **sentiment**. Sentiment refers to the writer's attitude in the text, so **sentiment analysis** is the task of analyzing or deducing what the writer's attitude is based on the text.

Oftentimes, the writer's attitude can be classified into one of several categories. For example, online user product reviews are normally given on a 1-5 star scale. In this case, we could separate the reviews based on how many stars they have.

An even simpler classification would be to separate the reviews into two categories: positive and negative. This is just regular binary classification, while the stars example is multiclass classification.


"This product is great, I've never found anything better!"
"This product is terrible. It was a complete waste of my money."
"While this product isn't perfect, it suits my needs well."
Positive
Negative
Positive

Classifying product reviews as either "Positive" or "Negative".

B. Training pairs

Similar to regular classification, text classification involves training pairs consisting of input data and labels. In this case, the input data will be tokenized text sequences and each text sequence will be labeled with a class (category). For simplicity, the class labels are just integers in the range $[0, n-1]$, where n is the total number of classes.

$$([1, 5, 6, 8, 2], 0)$$

$$([3, 5, 2, 9, 8], 1)$$

$$([2, 4, 7, 6, 6], 0)$$

Example batch of 3 training pairs. The maximum sequence length is 5 and there are two classes (binary classification).

Time to Code!

In this section of the course you'll be creating a general text classification model, by completing the `ClassificationModel` object.

Specifically, in this chapter you'll be completing the `make_training_pairs` function, which creates training pairs for the classification model.

The function has already been filled with code that processes raw text into tokenized, truncated sequences. Your job is to combine the training sequences and labels into training pairs.

Set `training_pairs` equal to `zip(sequences, labels)`, cast as a `list`. Then return `training_pairs`.

```
import tensorflow as tf
tf_fc = tf.contrib.feature_column

# Text classification model
class ClassificationModel(object):
    # Model initialization
    def __init__(self, vocab_size, max_length, num_lstm_units):
        self.vocab_size = vocab_size
        self.max_length = max_length
        self.num_lstm_units = num_lstm_units
        self.tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=self.vocab_size)

    def tokenize_text_corpus(self, texts):
        self.tokenizer.fit_on_texts(texts)
        sequences = self.tokenizer.texts_to_sequences(texts)
        return sequences

    # Create training pairs for text classification
    def make_training_pairs(self, texts, labels):
        sequences = self.tokenize_text_corpus(texts)
        sequences = self.tokenize_text_corpus(texts)
        for i in range(len(sequences)):
            sequence = sequences[i]
            if len(sequence) > self.max_length:
                sequences[i] = sequence[:self.max_length]
        # CODE HERE
```