For the base line solution, I will be using the python library Spacy to help with breaking the words into word vectors and their premade natural language processing functions. It also has a built-in tokenizer, which will help simplify the process. For the base model to generate the vectors, rather than using my own dataset, I am going to use one of Spacy's included datasets that has millions of recognized words and has vectors already made. With Spacy, I ran each sentence through Spacy and generated a word vector based on the sentence. I then concatenated the word vectors in a row together and added that to the list of entries I would run through the scikit-learn library to fit the data. For each of the rows in the data set, a 4 x 150 (75 rows for each sentence) numpy array was created and needed to be formatted to be passed into scikit. One of the issues with this method is that Spacy's tokenizer breaks up the words into tokens, but it also breaks any punctuation up, so each of the parenthesis were made into tokens as well. I ran a similar process for the testing data, but I had to format into a normal 2-D array to run it through the prediction model of scikit. Rather than predicting it all at once, I ended up predicting the label for each row, comparing it the actual label, and then keep tracking of which ones were accurate. I did it this way because we can't edit the CSV files and manually trying to count and organize the data into the appropriate rows would be tedious. This produced about a model that was ~51% accurate with predicting labels with the testing data set.

For the different word embedding method, I would use Google's BERT into order to try to do word embedding and word vector creation. With the baseline method of word embedding, context of the word isn't taken into account, which can play a large part in how it is used and the vector itself. An example would be the word "bank," it can be used in the context of a place where people and deposit and withdraw money, however, it can also be used in the context of like the land next to a body of water like a river. Under the baseline method (Word2Vec), both of the situations would generate a very similar vector, however, BERT would generate different vectors based on the context of the situation. With a large enough dataset such as the one Google has, this can lead to much more accurate word vectors as far as similarity goes, thus it should be able to predict the label for the pair of sentences much more accurately. BERT has a pretrained model, pytorch, by Google that was trained on many hours on Wikipedia and Book Corpus, meaning that its dataset is vastly superior and larger than most others, which gives it a good edge over all the other models.

Another potential word embedding method is using GloVe's learning algorithm that focuses on aggregated word to word co-occurrences from corpuses and linearly connecting words and vectors. This type of model would try to find pairs of words or words that are likely to co-occur often like how water is associated with liquid more so than solid or gas. Using this type of model, given a large dataset, we can find patterns in the sentences, and when converted to word vectors, we can match it to the closest neighbor within the model. This type of algorithm would be useful given a large dataset, and if there happens to be a decent amount of co-occurs and patterns within the sentences. Given our dataset, most are 5-6 words, meaning that there will likely be co-occurrences, and so using GloVe could potentially improve the performance of the labelled over the normal baseline, Word2Vec.

For the proposed solution, I will be using a Doc2Vec library rather than a Word2Vec library like Spacy. I assume this will produce a more accurate model, as a Word2Vec model does not care about wording sequential order and just vectorizes the sentence based on the words alone. This means that sentences that feature the same words may be categorized the same but have completely different meanings. Examples of which may be “I bought an apple and then ate it” versus “I ate an apple and then bought it” would have similar vectors, but the meaning itself is different and the sequential order of the tokens. By using a Doc2Vec sentence embedding technique, I would be able to create a vector that factors in the sequential order of the words in the sentence. Due to this, I believe I should be able to obtain more accurate results for the prediction of the sentence labels, given that I have a sizable amount of data and the model is made correctly.

For making the model, I used the Gensim’s Doc2Vec package to help with training and NLTK’s tokenizer to break up the sentence into individual tokens for the purpose of feeding it into the Doc2Vec package. To generate a data set, I fed the Doc2Vec all the sentences of “P1_Labelled_dataset.csv,” so it had ~14,400 sentences to train on, which is still relatively small compared to other datasets found online or through other libraries. I set the vector size for the model to be 20 and to run for 100 iterations, with an alpha of .025, and made it so I focused on a distributed memory (PV-DM) model rather than a distributed bag of words so that I could preserve the word order. After generating the model (d2v.model), I ran it through the Python file to train and predict accuracy. I ended up getting a lower accuracy than the base line, with an average around ~45% correct labels, however, this could be for many different reasons. The primary reason I believe that it ended up with lower accuracy is due to my dataset being relatively small with only ~14,400 sentences to run off, while Spacy’s model had millions of data points to learn and vectorize from. Even with a little more than 14,000 sentences, it was only less accurate by a 5%, and with that, I have reason to believe that if I had at least as much data that Spacy had, my model would be significantly (>10%) more accurate at predicting labels.