P2 – Response Type Classification in Discussions

```
def clean_data(df, df_test):
  columns = ['precedent', 'question', 'subsequent', 'response', 'type']
  df = df.iloc[:,5:]
  df_test = df_test.iloc[:,5:]
  df = df.fillna('')
  df = df[columns].replace({'&gt;': '','<&quot;>': ''}, regex=True)
  df_test = df_test.fillna('')
  df_test = df_test[columns].replace({'&gt;': '','<&quot;>': ''}, regex=True)
  return (df, df_test)
```

Function used to filter out columns like threaed_id, question_id etc. I only keep the ‘precedent’, ‘question’, ‘subsequent’, ‘response’ and ‘type’ columns in our train and test data.
I further replace NaN values with an empty string and remove markers ‘>’ and ‘<">’.

Baseline features

```
X_train_vectors = [[] for _ in range(len(X_train.columns))]
for i in range(len(X_train.columns)):
  for sentence in X_train[feature_columns[i]]:
    wordsList = nltk.word_tokenize(sentence)
    text_tagged = nltk.pos_tag(wordsList)
    new_text = []
    for word in text_tagged:
      new_text.append(word[0] + "/" + word[1])
    new_text = ' '.join(new_text)
    emb = embed([new_text])
    sentence_emb = tf.reshape(emb, [-1]).numpy()
    X_train_vectors[i].append(sentence_emb)

X_train_vectors = list(zip(X_train_vectors[0], X_train_vectors[1], X_train_vectors[2], X_train_vectors[3]))
for i in range(len(X_train_vectors)):
  X_train_vectors[i] = [item for sublist in X_train_vectors[i] for item in sublist]
X_train_vectors = np.array(X_train_vectors)
```

For baseline features I use POS tags and sentence embeddings. Each word in a sentence is converted to a new word by appending the POS with the respective words e.g. ‘me’ becomes ‘me/PRP’, ‘example’ becomes ‘example/NN’ etc. The sentence embedding is then found for these new sentences.

The sentence embeddings of ‘precedent’, ‘sentence’, ‘subsequent’ and ‘response’ is concatenated for each data entry.

The resulting train and test vectors are of the shape (# samples x 2048). 512 vector size for each sentence (precedent, question, subsequent, response).

Training models

Two classifier options are available SVC and Neural Nets. My Implementation produces better results with SVC.

```
if classifier == 'SVC':
  #SVC Grid Search
  '''
  param_grid_SVC = {'C': [0.1, 1, 10, 100],
                'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10],
                'kernel': ['rbf', 'linear']}

  clf_SVC = GridSearchCV(SVC(), param_grid_SVC)
  clf_SVC.fit(X_train_vectors, y_train)
  print("Best parameter: {}, Best score: {}".format(clf_SVC.best_params_, clf_SVC.best_score_))
  '''

  model = SVC(C=10, kernel='rbf', gamma=1)
  model.fit(X_train, y_train)
```

The hyper-parameters for the model are selected using grid search. Uncomment to perform grid search on a new dataset. For the given dataset, the given parameters, C=10, kernel=’rbf’, and gamma=1 gives the best results.

```
elif classifier == 'NN':
  model = keras.Sequential()
  model.add(keras.layers.Dense(units=128, input_shape=(X_train.shape[1],),
                               activation='relu'))
  model.add(keras.layers.Dropout(rate=0.5))
  model.add(keras.layers.Dense(units=64, activation='relu'))
  model.add(keras.layers.Dropout(rate=0.5))
  model.add(keras.layers.Dense(4, activation='softmax'))
  model.compile(loss='sparse_categorical_crossentropy', optimizer=keras.optimizers.Adam(0.01),
                metrics=['accuracy'])

  history = model.fit(
  X_train, y_train, epochs=9, batch_size=32,
  verbose=0, validation_split=0.1, shuffle=True)

  if plot_training:
    plt.plot(history.history['loss'], label='training data')
    plt.plot(history.history['val_loss'], label='validation data')
    plt.legend(loc="upper right")
    plt.show()

    plt.plot(history.history['accuracy'], label='training data')
    plt.plot(history.history['val_accuracy'], label='validation data')
    plt.legend(loc="upper left")
    plt.show()
```

The NN classifier implementation is shown above.

Testing Models

```
def test_model(model, X_test, y_test, classifier='SVC'):
  if classifier == 'SVC':
    pred_classes = model.predict(X_test)
  elif classifier == 'NN':
    pred_classes = np.argmax(model.predict(X_test), axis=-1)
  accuracy = accuracy_score(y_test, pred_classes)
  precision = precision_score(y_test, pred_classes, average='weighted')
  recall = recall_score(y_test, pred_classes, average='weighted')
  f1 = f1_score(y_test, pred_classes, average='weighted')
  print(f"Accuracy: {accuracy}")
  print('Precision: %f' %precision)
  print('Recall: %f' %recall)
  print('F1 score: %f' %f1)
```

Accuracy, Precision, Recall, and F1 score are the evaluation metrics calculated.

Baseline feature results

SVC classifier:

```
Accuracy: 0.7731707317073171
Precision: 0.687327
Recall: 0.773171
F1 score: 0.709138
```

Neural Net classifier:

```
Accuracy: 0.7804878048780488
Precision: 0.609161
Recall: 0.780488
F1 score: 0.684263
```

Classification using NN produces a better accuracy and recall however, SVC classifier performs much better in terms of precision and F1 score. The performance gain in SVC is much greater when compared to NN, thus SVC is chosen in the final classifier.

Additional Features

My proposed solution uses Word Senses and sentiment analysis to try and improve the results for classification task.

Feature 1 - Word sense disambiguation

Assigning word senses to each word in a sentence, in addition to the POS tags, helps us gain further information about each word. Extracting more information about each word in the question and response should help up extract more information about a sentence thus, further improving our results.

```
for i in range(len(X_test.columns)):
  for sentence in X_test[feature_columns[i]]:
    sentence_ws = disambiguate(sentence)
    wordsList = nltk.word_tokenize(sentence)
    text_tagged = nltk.pos_tag(wordsList)

    new_text = []
    new_text2 = []
    idx = 0
    for word in text_tagged:
      new_text.append(word[0] + "/" + word[1] + "/" + (sentence_ws[idx][1].name() if (sentence_ws[idx] and sentence_ws[idx][1]) else ''))
      idx += 1

    new_text = ' '.join(new_text)

    emb = embed([new_text])
    sentence_emb = tf.reshape(emb, [-1]).numpy()
    X_test_vectors[i].append(sentence_emb)
```

In addition to the POS we append the word sense associated with each word, e.g. ‘make’ becomes ‘make/VB/make.v.43’, ‘decisions’ become ‘decisions/NNS/decision.n.01’. For words which do not have a word sense. Words which do not have varying word senses have an empty string attached to it, e.g. ‘of’ becomes ‘of/IN/’, ‘these’ becomes ‘these/DT/’.

pyswd python library is used for word sense disambiguation. pyswd uses WordNet.
The sentence embedding for the new sentences is formed.

Feature 2 – Sentiment Analysis of responses

I perform sentiment analysis for the responses and introduce the results as new features which are appended to the existing sentence embeddings. The reasoning is that the sentiment of the response might help us identify its type. An ‘attacked’ response is likely to have a negative sentiment, ‘agreed’ response is likely to have a positive sentiment whereas ‘irrelevant’ and ‘answered’ responses are more likely to be neutral in nature.

vaderSentiment python library is used for sentiment analysis of the responses.

```
def sentiment_analyzer_scores(sentence):
    sentiment_analyzer = SentimentIntensityAnalyzer()
    score = sentiment_analyzer.polarity_scores(sentence)
    return score
```

The score for a sentence in returned as a dictionary.

```
{'neg': 0.0, 'neu': 0.408, 'pos': 0.592, 'compound': 0.4404}
```

The compound score is the overall sentiment score between -1 and 1.
The 4 additional values are introduced as new features.

```
X_train_vectors = list(zip(X_train_vectors[0], X_train_vectors[1], X_train_vectors[2], X_train_vectors[3]))

for i in range(len(X_train_vectors)):
  X_train_vectors[i] = [item for sublist in X_train_vectors[i] for item in sublist]

for i in range(len(X_train_vectors)):
  X_train_vectors[i] += sentiment_score_response[i]

X_train_vectors = np.array(X_train_vectors)
```

The ‘precedent’, ‘question’, ‘subsequent’ and ‘response’ sentence embeddings are concatenated as before.
Sentiment scores of each response is then concatenated to the above concatenation.

The final train and test models are of the shape (# samples x 2052). 2048 (sentence embeddings from 4 sentences) + 4 (sentiment scores for the response).

Additional features results

SVC classifier:

```
Accuracy: 0.7780487804878049
Precision: 0.736904
Recall: 0.778049
F1 score: 0.715986
```

F1 score increases by ~ 1%
Precision increases by ~ 5%
Accuracy increases by ~ 0.5%
Recall increases by ~ 0.5%

Conclusion

The addition of new feature like word sense help us to gain further knowledge about each word present in any given sentence. Word senses help us distinguish between sentiment analysis of the response help us relate the sentiment of a response with the type of the response. These additions features lead to improvement across all evaluation metrics. The gain in accuracy and recall is marginal. However, the precision increases by almost 5%, proving that the new features help improve our classification model.