Advanced Natural Language Processing - Assignment 1

Submitted by: Kalyani Prashant Kawale

Student ID: 21237189

The systems generated for classifying the relation between adjectives are described in the following sections:

Task 1 Solution

Splitting Data Set into Training and Development Sets:

To generate the multi-class classifier systems, the data-set provided in the file *train.csv* was divided into training and development sets as follows:

- The *train.csv* was loaded using **read_csv** method of python's **pandas** library using the following code. The **fillna** method [1] was used to handle the **NAN** values present in some columns of the data-set, by replacing them with the string 'none'.

```
synonyms_corpus = pd.read_csv('train.csv') # Reading the train file
synonyms_corpus = synonyms_corpus.fillna('none') # Replacing NAN values with string none
```

- 1/3rd portion of the data-set was used as the development set, while the remaining 2/3rd portion was used as the train set.
- The **synonyms_corpus** data-frame was first pre-processed as a whole for each task 2, 3 and 4 depending on the pre-processing required for the models and then split into train and develop sets.
- As the size of the data-set was relatively small (1190 train cases) and the number of labels for classification were relatively more (6 classes), 10 random train and develop divisions were generated to provide different images of the data to the classifiers for training as follows, where **corpus_pooled** contains pre-processed data-frame and **splits** saves 10 dictionaries of train and develop sets which are further used for training and evaluation,

```
splits = [] # Initialising splits list
corpus_len = len(corpus_pooled) # Getting length of entire data-set
develop_size = int(corpus_len / 3) # Getting 1/3rd length of data-set
for i in range(10):
    copy = np.copy(corpus_pooled)
    np.random.shuffle(copy)  # Shuffling the data-set
    # Saving 2/3rd portion as train set and 1/3rd portion as develop set
    splits.append({'train': copy[:(corpus_len - develop_size), :],
                   'develop': copy[(corpus_len - develop_size):, :]})
```

- The labels were encoded using **sklearn LabelEncoder**. The classes [**'equivalent', 'less intense', 'less specific', 'more intense', 'more specific', 'related'**] were encoded into [**0, 1, 2, 3, 4, 5**] respectively, using following code,

```
from sklearn.preprocessing import LabelEncoder
# Initialising LabelEncoder
label_encoder = LabelEncoder()
y = synonyms_corpus['LABEL'].values
# Fitting label_encoder with label values
label_encoder.fit(y)
# Encoding classes into integers
y_encoded = label_encoder.transform(y)
```

Evaluation Method Selected:

- The evaluation method used to evaluate the classifier systems is **Accuracy.**
- While **Accuracy** measures may not always provide correct evaluation of classifiers due to dominance of one majority class, it is one of the most useful measures in classification tasks.
- Based on an approach provided by [2] to decide the evaluation metric for imbalanced classification, the percentage of majority class (**'more specific'**) was determined as follows,

```
label, count = np.unique(synonyms_corpus['LABEL'].values, return_counts=True)
print("Percentage of majority class": max(count)/sum(count))
```

- The above code returned a value of **47%** for the class label with maximum counts.
- [2] suggests different metrics based on the outputs and nature of the task. Since this task involved classifying labels and no specific class label held more importance than any other, and the percentage of majority class was below 80% to 90% in the data-set, **Accuracy** was chosen as the evaluation metric.
- The individual and mean train and test accuracy for the 10 splits have been reported for each classifier in the following sections.

Task 2 Solution

The simple classifier using common words features was developed as follows:

Feature Extraction:

The features used for task 2 were,

1. Number of common words in DEFINITION 1 and DEFINITION 2 for a sample
2. Number of common words in EXAMPLE 1 and EXAMPLE 2 for a sample

The strings in definition and example columns were pre-processed using following **preprocess_sent** method to generate tokens, remove punctuation, and remove extra quotes around the strings,

```
from nltk import word_tokenize
from nltk.corpus import stopwords
import string
# Items to be removed from strings
exclusions = [list(string.punctuation), '``', "''"]
# Method to pre-process sentences and return clean tokens
def preprocess_sent(sentence):
    words = []
    for word in word_tokenize(sentence):
        word = word.lower()
        if word not in exclusions and word not in words:
            words.append(word)
    return words
# Method to return number of common words in two lists
def common_occurences(words1, words2):
    return len(set(words1) & set(words2))
# Method to extract features after preprocessing
def extract_features(X):
    X_features = []
    for i in range(X.shape[0]):
        features = []
        definition1 = preprocess_sent(X['DEFINITION 1'][i])
        definition2 = preprocess_sent(X['DEFINITION 2'][i])
        # Appending feature 1
        features.append(common_occurences(definition1, definition2))
        example1 = preprocess_sent(X['EXAMPLE 1'][i])
        example2 = preprocess_sent(X['EXAMPLE 2'][i])
        # Appending feature 2
        features.append(common_occurences(example1, example2))
        X_features.append(features)
    return np.array(X_features)
```

Model Settings:

- Random Forest classifier was used to train and predict classes using the two features extracted in the above section. Random Forest is an inherently multi-class classifier [3] and was found to be performing better than other available classifiers.
- The **RandomForestClassifier** object from **scikit-learn** was used to generate the model with following parameters, fixed after fine-tuning the model for the given data-set,

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=20, max_depth=8, criterion='entropy')
```

Results:

After training and evaluating the 10 random divisions generated for data-set with the two features extracted for task 2, following results were obtained,

Training and Development Accuracy for 10 Divisions:

	Split	TRAIN ACCURACY	DEVELOP ACCURACY
0	1	0.506297	0.429293
1	2	0.491184	0.469697
2	3	0.460957	0.507576
3	4	0.502519	0.419192
4	5	0.498741	0.439394
5	6	0.496222	0.452020
6	7	0.486146	0.469697
7	8	0.483627	0.469697
8	9	0.481108	0.479798
9	10	0.493703	0.441919

MEAN TRAIN ACCURACY: 49%
MEAN DEVELOP ACCURACY: 46%

Figure 1: Task 2: Train & Develop Accuracy for 10 Splits and Mean Accuracy for All Runs

As reported in Figure 1. the average accuracy provided by task 2 model for development set was **46%**, while the highest accuracy achieved on the development set was approximately **51%**.

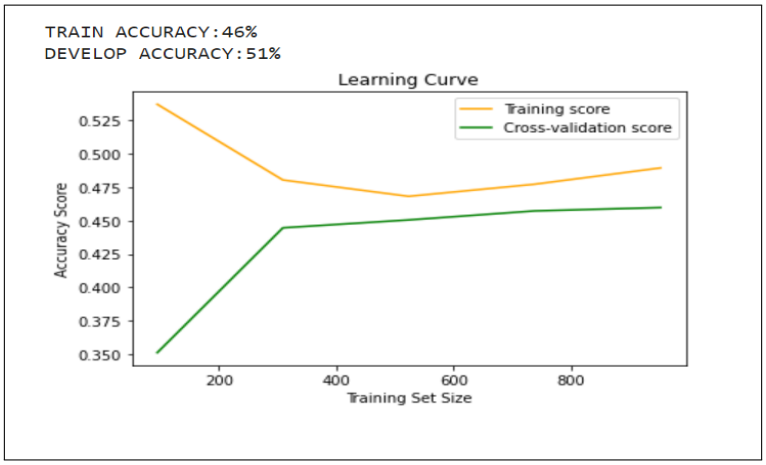

Figure 2: Task 2: Learning curve obtained for the division with highest development set accuracy generated using LearningCurve [4] method provided by scikit-learn

Despite the low accuracy, task 2 model was able to capture all the classes in predictions for different runs. The classes mostly captured in all runs were, **0 ('equivalent'), 3 ('more intense'), 4 ('more specific'), and 5 ('related')**.

```
Classes Predicted for Development Set and their Counts:

Split 1: (array([0, 3, 4, 5]), array([ 17,  17, 360,   2]))
Split 2: (array([0, 1, 3, 4, 5]), array([ 16,   1,   4, 372,   3]))
Split 3: (array([0, 1, 3, 4, 5]), array([  8,   2,  14, 367,   5]))
Split 4: (array([0, 2, 3, 4, 5]), array([ 24,   1,   1, 361,   9]))
Split 5: (array([0, 3, 4, 5]), array([ 19,   2, 368,   7]))
Split 6: (array([0, 3, 4, 5]), array([ 24,   6, 357,   9]))
Split 7: (array([0, 3, 4, 5]), array([ 16,   1, 378,   1]))
Split 8: (array([0, 3, 4, 5]), array([ 34,   8, 352,   2]))
Split 9: (array([0, 3, 4, 5]), array([ 14,   6, 370,   6]))
Split 10: (array([0, 3, 4, 5]), array([ 11,  11, 371,   3]))
```

Figure 3: Task 2: Classes Predicted for Development Set in 10 runs

Task 3 Solution

The classifier using BERT model embedding, based on [6] was developed as follows:

Feature Extraction:

- The feature used for task 3 was a single feature generated after concatenating the pooled_output for DEFINITION 1 and DEFINITION 2 obtained from the BERT model embedding.
- **bert_en_uncased_preprocess** model was used to pre-process the definition columns as follows,

```
bert_preprocess_model = hub.KerasLayer(tfhub_handle_preprocess)
def1_preprocessed = bert_preprocess_model(synonyms_corpus['DEFINITION 1'])
def2_preprocessed = bert_preprocess_model(synonyms_corpus['DEFINITION 2'])
```

- The BERT model used was Small BERT available at https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-128_A-2/1, due to RAM limitations and the small size of the data-set. Following code was used to obtain the embeddings and concatenate them into one feature array,

```
bert_model = hub.KerasLayer(tfhub_handle_encoder)
def1 = bert_model(def1_preprocessed)
def2 = bert_model(def2_preprocessed)
corpus_pooled = np.concatenate((
                        def1['pooled_output'],
                        def2['pooled_output']
                        ), axis=1)
```

- Further, the labels encoded into integers were converted into one-hot encoded format using **np_utils.to_categorical** method provided by **keras.utils** for further processing.

Model Settings:

- The model for task 3 was developed as a neural network with single output layer with 6 units for 6 classes using **Keras** with following configurations,

```
from keras.models import Sequential
from keras.layers import Dense
def build_model(input_dim, output_dim):
    model = Sequential()
    model.add(Dense(output_dim, activation='softmax', input_dim=input_dim))
    model.compile(optimizer='adam', loss='CategoricalCrossentropy', metrics=['accuracy'])
    return model
```

- The above model was also trained and evaluated on the 10 random divisions generated in Task 1.

Results:

After training and evaluating the 10 random divisions generated for concatenated pooled outputs for definitions extracted for task 3, following results were obtained,

Training and Development Accuracy for 10 Divisions:

	Split	TRAIN ACCURACY	DEVELOP ACCURACY
0	1	0.459698	0.457071
1	2	0.459698	0.477273
2	3	0.455919	0.452020
3	4	0.467254	0.477273
4	5	0.477330	0.421717
5	6	0.476071	0.459596
6	7	0.464736	0.477273
7	8	0.478589	0.454545
8	9	0.460957	0.474747
9	10	0.478589	0.452020

MEAN TRAIN ACCURACY: 47%
MEAN DEVELOP ACCURACY: 46%

Figure 4: Task 3: Train & Develop Accuracy for 10 Splits and Mean Accuracy for All Runs

The average accuracy provided by task 3 classifier for development set was **46%**, while the highest accuracy achieved on the development set was approximately **48%**. However, it was observed that this model did not provide stable outcomes compared to task 2 model, as shown in Figure 4.

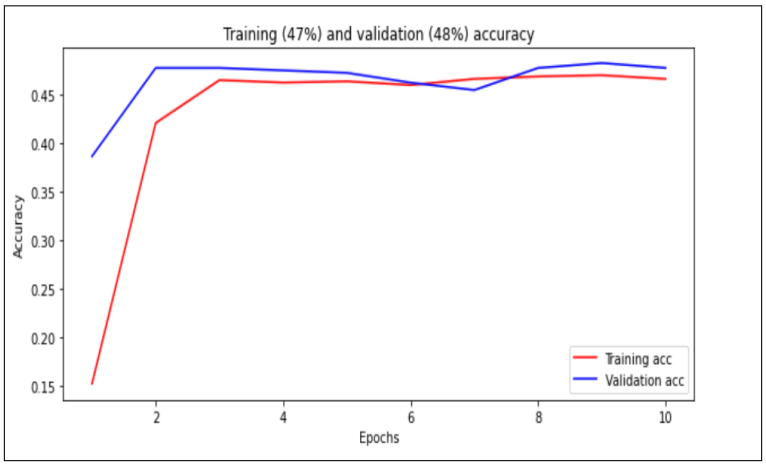

Figure 5: Task 3: Learning curve obtained for the division with highest development set accuracy, using history information obtained from Keras Classifier [6]

Task 3 model was able to capture only the **0 ('equivalent'), 3 ('more intense'), 4 ('more specific'), and 5 ('related')** classes in predictions as follows,

```
Classes Predicted for Development Set and their Counts:

Split 1: (array([0, 4, 5]), array([  6, 384,   6]))
Split 2: (array([0, 4, 5]), array([  8, 386,   2]))
Split 3: (array([0, 4, 5]), array([ 16, 371,   9]))
Split 4: (array([0, 4, 5]), array([  2, 393,   1]))
Split 5: (array([0, 4]), array([ 32, 364]))
Split 6: (array([3, 4, 5]), array([  4, 390,   2]))
Split 7: (array([0, 4, 5]), array([  5, 390,   1]))
Split 8: (array([4]), array([396]))
Split 9: (array([0, 3, 4, 5]), array([  3,   4, 388,   1]))
Split 10: (array([0, 4, 5]), array([  5, 390,   1]))
```

Figure 6: Task 3: Classes Predicted for Development Set in 10 runs

Task 4 Solution

The classifier to provide an improvement to the models developed in previous sections was developed as follows:

Feature Extraction:

- Features extracted in task 2 and 3 were both used in this model, as BERT model provided contextual embedding and count of common words provided relation between the related columns in the data-set.
- Further, the word mover's distance was calculated between the definitions and examples, with the intuition that WMD followed a pattern based on the class of the sample. For example, following output of WMD similarities between definitions was obtained for the instances in the data-set,

```
LABEL for 0th instance: 'more specific' / wmd: 1.9572955857955332
LABEL for 1st instance: 'equivalent' / wmd: 1.6035504836060375
LABEL for 5th instance: 'less specific' / wmd: 2.359159918856343
```

Figure 7: Task 4: wmd similarities between definitions for samples with different class labels

From Figure 7. it can be seen that **'equivalent'** lemmas have a shorter distance compared to lemmas with **'more specific'** relation, which is shorter than lemmas with **'less specific'** relation.

- Thus, the features used for task 4 model were, pooled_output of BERT embedding for Definitions, the count of common words between definitions and examples and word mover's distance similarities between definitions.

Model Settings:

- Using different combinations of the features described in previous sections on models from task 2 and 3, it was found that **RandomForestClassifier** model provided more stable results.
- Due to increased features however, **RandomForestClassifier** model initially resulted in overfitting, to overcome this issue the model in task 2 was fine-tuned by reducing the max depth to perform tree pruning as follows,

```
classifier = RandomForestClassifier(n_estimators=10, max_depth=4, criterion='entropy')
```

Results:

After training and evaluating the 10 random divisions on enhanced task 2 model, following results were obtained,

```
Training and Development Accuracy for 10 Divisions:

   Split  TRAIN ACCURACY  DEVELOP ACCURACY
0      1        0.508816          0.487374
1      2        0.507557          0.467172
2      3        0.508816          0.482323
3      4        0.496222          0.517677
4      5        0.505038          0.457071
5      6        0.498741          0.467172
6      7        0.493703          0.467172
7      8        0.510076          0.452020
8      9        0.508816          0.444444
9     10        0.530227          0.459596


MEAN TRAIN ACCURACY: 51%
MEAN DEVELOP ACCURACY: 47%
```

Figure 8: Task 4: Train and Develop Accuracy

The average accuracy provided by task 4 classifier for development set was **47%**, while the highest accuracy achieved on the development set was approximately **52%**, as shown in Figure 8.

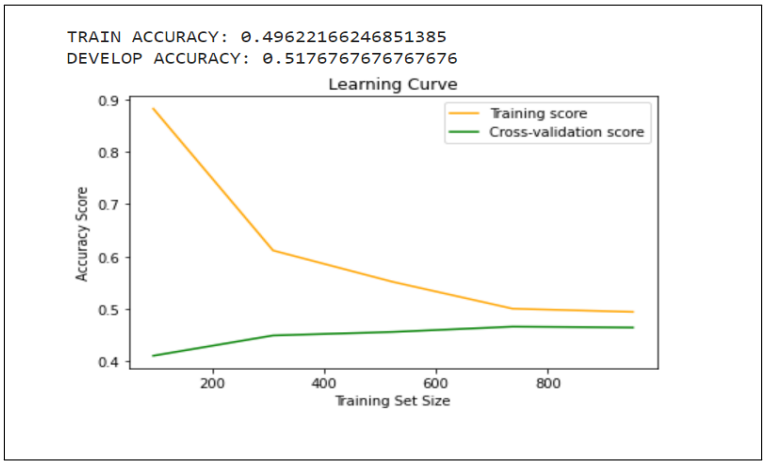

Figure 9: Task 4: Learning curve obtained for the division with highest development set accuracy generated using LearningCurve [4] method provided by scikit-learn

Task 4 model was mostly able to capture the **0 ('equivalent'), 3 ('more intense'), 4 ('more specific'), and 5 ('related')** classes in predictions for reported run. For different runs, the model was also able to capture **1 ('less intense'), 2 ('less specific')** classes.

```
Classes Predicted for Development Set and their Counts:

Split 1: (array([0., 3., 4., 5.]), array([  3,   2, 388,   3]))
Split 2: (array([0., 3., 4., 5.]), array([ 12,   3, 377,   4]))
Split 3: (array([0., 4., 5.]), array([ 10, 381,   5]))
Split 4: (array([0., 3., 4., 5.]), array([  5,   4, 386,   1]))
Split 5: (array([0., 3., 4., 5.]), array([  2,   3, 390,   1]))
Split 6: (array([0., 3., 4., 5.]), array([  1,   2, 388,   5]))
Split 7: (array([0., 4., 5.]), array([  1, 394,   1]))
Split 8: (array([4.]), array([396]))
Split 9: (array([0., 4., 5.]), array([  6, 388,   2]))
Split 10: (array([0., 3., 4., 5.]), array([  3,   6, 386,   1]))
```

Figure 10: Task 4: Classes Predicted for Development Set in 10 runs

The task 4 model was then ran on *eval.csv* to generate the predictions for test samples.

References

[1] API Reference - pandas.DataFrame.fillna. Available at: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html [Accessed on: 28/02/2022]

[2] Jason Brownlee. (2022) Tour of Evaluation Metrics for Imbalanced Classification. Available at: https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/ [Accessed on: 28/02/2022]

[3] Multiclass and multioutput algorithms. Available at: https://scikit-learn.org/stable/modules/multiclass.html [Accessed on: 28/02/2022]

[4] Plotting Learning Curves. Available at: https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html [Accessed on: 02/03/2022]

[5] Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework. (2019). Available at: https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/ [Accessed on: 28/02/2022]

[6] Classify text with BERT. Available at: https://www.tensorflow.org/text/tutorials/classify_text_with_bert [Accessed on: 01/03/2022]

[7] Training and evaluation with the built-in methods. Available at: https://www.tensorflow.org/guide/keras/train_and_evaluate [Accessed on: 01/03/2022]

[8] CT5120 Lab Sheet: Exercise Sheet 3 - Sentiment Analysis

[9] CT5121 Lab Sheet: ANLP Lab 4

[10] CT5121 Lab Sheet: ANLP Lab 5: Textual Similarity

[11] CT5121 Lab Sheet: ANLP Lab 7 - Under-resourced language processing