P1- Sentiment Analysis Report

Base Models:

Some text pre-processing is done for before applying the base models. The punctuations and all digits are removed. Also, the text is converted to lower case. Both base models use logistic regression for classification.

1. TF-IDF: For the first base model, word embeddings are created using TF-IDF i.e. term frequency- inverse document frequency.

2. Glove Word2Vec pre-trained: The second base model uses glove's pre-trained word embedding model. Glove's 300-d wiki gigaword embedding is used.

Proposed Solution:

Universal Sentence Encoder: The proposed solution uses universal sentence encoder embedding. USE computes sentence level meaning similarity instead of just considering word meaning.

Universal sentence encoder large is used for this solution.

Also, in addition to using logistic regression, a simple neural network is proposed for sentiment classification.

Classification techniques:

1. Logistic Regression: For logistic regression models, we first perform a grid-search to find the best hyper-parameter C. The model is then trained using this value. Accuracy, precision, recall and f1 scores are calculated to compare the performances of different models.

2. Neural Network: A simple neural network is trained for classification. The two layers have 64 and 32 units respectively with a dropout rate of 0.5 each. Sparse categorical cross-entropy loss and Adam optimizer with a learning rate of 0.001 is used. The network is trained for 8 epochs. Again, accuracy, precision, recall, and f1 scores are calculated.

Model Performances:

1. TF-IDF

```
Model: TF-IDF
Accuracy: 0.5563689604685212
Precision: 0.570978
Recall: 0.556369
F1 score: 0.563128
```

2. Glove w2v

```
Model: GLOVE WORD2VEC
Accuracy: 0.5592972181551976
Precision: 0.634280
Recall: 0.559297
F1 score: 0.585576
```

3. Universal Sentence Encoder with Logistic Regression:

```
Model: UNIVERSAL SENTENCE ENCODER
Accuracy: 0.6076134699853587
Precision: 0.660863
Recall: 0.607613
F1 score: 0.625554
```

4. Universal Sentence Encoder with Neural Network:

```
Model: UNIVERSAL SENTENCE ENCODER
Accuracy: 0.6456808199121523
Precision: 0.625972
Recall: 0.645681
F1 score: 0.614776
```

The two baseline models have almost the same accuracy and recall however glove word2vec pre-trained embedding perform better in terms of precision and f1-score.
The proposed solution which utilizes universal sentence encoder outperforms both of the baseline models in all performance metrics. Comparing two different classification models for the proposed solution, logistic regression has a better precision and f1-score however, the neural network performs better in terms of recall and overall accuracy.

Code Snippets:

1. TF-IDF:

Tokenize and Vectorize: TfidfVectorizer is used to tokenize and create word vectors for all the words in the corpus. It is a part of tensorflow. Fitted and transformed on the training dataset.

```
tfidf_vectorizer = TfidfVectorizer()
tfidf_vectorizer.fit(X_train)
X_train_tfidf = tfidf_vectorizer.transform(X_train)
```

Logistic Regression Classification: GridSearchCV is used to perform hyper-parameter tuning for the logistic regression model using grid search and cross validation. Hyper-parameter C is varied and the C with the highest score is used to train the final model.

```
param_grid_LR = {'C':[0.01, 0.05, 0.25, 0.5, 0.1]}
clf_LR = GridSearchCV(LogisticRegression(max_iter = 500, class_weight='balanced'), param_grid=param_grid_LR)
clf_LR.fit(X_train_tfidf, y_train)

model = LogisticRegression(C=clf_LR.best_params_['C'], max_iter=500, class_weight='balanced')
model.fit(X_train_tfidf, y_train)
```

2. Glove w2v:

Tokenize: The training dataframe is converted to a list and a tokenized list is created using simple preprocess method provided by genism module.

```
sentences = X_train.values.tolist()
tokenized_list = [simple_preprocess(sentence) for sentence in sentences]
```

Vectorize: The sentence is represented by the average of the word embedding vectors of the words that compose the sentence. If the word does not exist in the vocabulary of the w2v mode its vectors are set to 0.

```
def compute_avg_vector(w2v_dict, sentence):
  list_of_word_vectors = [w2v_dict[w] for w in sentence if w in w2v_dict.vocab.keys()]
  if len(list_of_word_vectors) == 0:
    result = [0.0]*300
  else:
    result = np.sum(list_of_word_vectors, axis=0)/len(list_of_word_vectors)

    return result
```

Logistic Regression Classification: Similar to logistic regression model used for tfidf

```
param_grid_LR = {'C':[0.01, 0.05, 0.25, 0.5, 0.1]}
clf_LR = GridSearchCV(LogisticRegression(max_iter = 500, class_weight='balanced'), param_grid=param_grid_LR)
clf_LR.fit(X_train_w2v, y_train)

model = LogisticRegression(C=clf_LR.best_params_['C'], max_iter=500, class_weight='balanced')
model.fit(X_train_w2v, y_train)
```

3. Universal Sentence Encoder

Tokenize and Vectorize: The sentences are tokenized and vectorized using the embeddings loaded from the pre-trained USE large 5 module. The sentence is embedded using embed which is the var in which the module is loaded.

```
for r in X_train:
  emb = embed([r])
  sentence_emb = tf.reshape(emb, [-1]).numpy()
  X_train_vectors.append(sentence_emb)
```

Logistic Regression Classification: Similar to previous logistic regression

```
param_grid_LR = {'C':[0.01, 0.05, 0.25, 0.5, 0.1]}
clf_LR = GridSearchCV(LogisticRegression(max_iter = 500, class_weight='balanced'), param_grid=param_grid_LR)
clf_LR.fit(X_train_vectors, y_train)

model = LogisticRegression(C=clf_LR.best_params_['C'], max_iter=500, class_weight='balanced')
model.fit(X_train_vectors, y_train)
```

Neural Network Classification: A simple neural net is trained for 8 epochs with 64, 32 and 3 units in first, second, and third dense layers respectively. Categorical cross entropy and Adam optimizer are used. If plot_training is True, the training accuracy and loss curves can be observed.

```
model = keras.Sequential()
model.add(keras.layers.Dense(units=64, input_shape=(X_train_vectors.shape[1],),
                             activation='relu'))
model.add(keras.layers.Dropout(rate=0.5))
model.add(keras.layers.Dense(units=32, activation='relu'))
model.add(keras.layers.Dropout(rate=0.5))
model.add(keras.layers.Dense(3, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer=keras.optimizers.Adam(0.001),
              metrics=['accuracy'])

history = model.fit(
X_train_vectors, y_train, epochs=7, 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()
```