COMP4332 Group 4 Project 1 Sentiment Analysis Report

Introduction

This report presents the findings and observations from the sentiment analysis project conducted by Group 4. The goal of the project was to develop models for sentiment analysis on movie reviews. The project involved data preprocessing, model development, and performance evaluation. This report summarizes the key steps taken and the results obtained

Data Preprocessing

The project began with an analysis of the class label distribution in both the training and testing datasets. It was observed that the label distribution exhibited some bias, with a higher number of relative positive and neutral ratings compared to relative negative ratings. This bias in the dataset can potentially make the models more sensitive to positive and neutral reviews. If the testing dataset has a more even distribution of class labels, the test time performance will be affected and result in poor testing accuracy and F1 score.

The data preprocessing steps involved tokenization and stemming of the text data. However, stopword filtering using the nltk library was not applied. The stopwords list included negation words ("no," "not") and extended words ("so," "some," "too," "very").

Here is a possible example to illustrate the effect of applying stopword filtering:

possible ratings	possible words in reviews		after stopword filtering
1	very not good		good
2	not good	stopword filtering	good
3	not so good	—>	good
4	good		good
5	very good		good

It was noted that applying stopword filtering using nltk library may not be suitable for sentiment analysis as these words could be important features affecting the ratings.

Classification Models

CNN Model (Selected Model):

The primary classification model used in the project was a Convolutional Neural Network (CNN) architecture. The CNN model was chosen due to its effectiveness in handling text inputs.

The CNN model employed a kernel size of 1, 2, and 3 to extract unigram, bigram, and trigram features from the text inputs. It was determined that further N-grams might not be useful for analysis as expressions can often be captured using fewer words.

The softmax activation function was used for the output layer, and the Adam optimizer was selected as it is more suitable for multi-label classification tasks.

The model layout is as follows:

```
Model: "model_9"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_10 (InputLayer)       [(None, 100)]             0         []

 embedding_9 (Embedding)     (None, 100, 100)          454600    ['input_10[0][0]']

 conv1d_27 (Conv1D)          (None, 100, 100)          10100     ['embedding_9[0][0]']

 conv1d_28 (Conv1D)          (None, 99, 100)           20100     ['embedding_9[0][0]']

 conv1d_29 (Conv1D)          (None, 98, 100)           30100     ['embedding_9[0][0]']

 activation_37 (Activation)  (None, 100, 100)          0         ['conv1d_27[0][0]']

 activation_38 (Activation)  (None, 99, 100)           0         ['conv1d_28[0][0]']

 activation_39 (Activation)  (None, 98, 100)           0         ['conv1d_29[0][0]']

 max_pooling1d_27 (MaxPooli  (None, 1, 100)            0         ['activation_37[0][0]']
 ng1D)

 max_pooling1d_28 (MaxPooli  (None, 1, 100)            0         ['activation_38[0][0]']
 ng1D)

 max_pooling1d_29 (MaxPooli  (None, 1, 100)            0         ['activation_39[0][0]']
 ng1D)

 flatten_27 (Flatten)        (None, 100)               0         ['max_pooling1d_27[0][0]']

 flatten_28 (Flatten)        (None, 100)               0         ['max_pooling1d_28[0][0]']

 flatten_29 (Flatten)        (None, 100)               0         ['max_pooling1d_29[0][0]']

 concatenate_9 (Concatenate  (None, 300)               0         ['flatten_27[0][0]',
 )                                                               'flatten_28[0][0]',
                                                                 'flatten_29[0][0]']

 dropout_9 (Dropout)         (None, 300)               0         ['concatenate_9[0][0]']

 dense_19 (Dense)            (None, 100)               30100     ['dropout_9[0][0]']

 activation_40 (Activation)  (None, 100)               0         ['dense_19[0][0]']

 dense_20 (Dense)            (None, 5)                 505       ['activation_40[0][0]']

==================================================================================================
Total params: 545505 (2.08 MB)
Trainable params: 545505 (2.08 MB)
Non-trainable params: 0 (0.00 Byte)
```

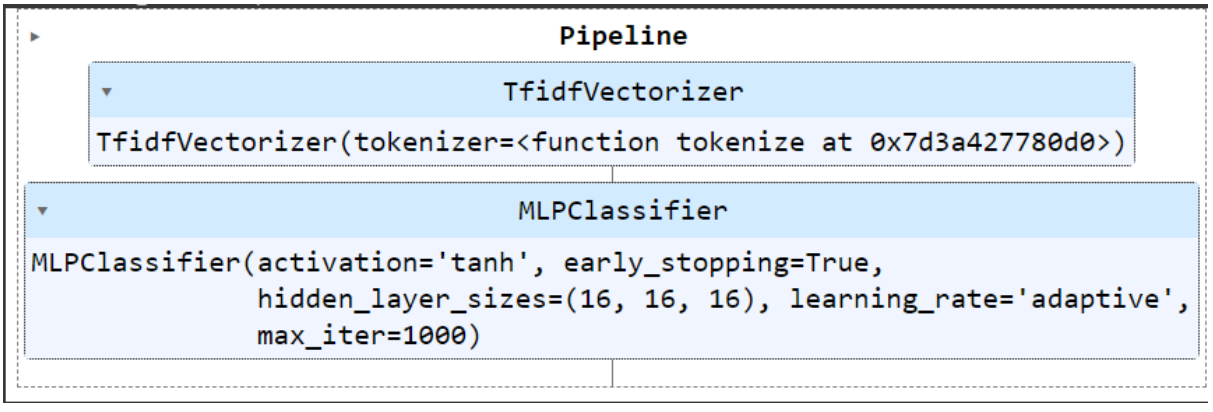
CNN Model Validation Performance
The CNN model's validation performance was evaluated using different configurations. The following are the validation performance achieved for various settings:

CNN kernel size (input condition)	Accuracy	Precision	Recall	F1
[1, 2, 3] (input without stopword filtering)	0.543	0.524	0.514	0.514
[1, 2, 3] (input with stopword filtering)	0.524	0.503	0.491	0.493
[1, 2, 3, 4] (input without stopword filtering)	0.536	0.519	0.506	0.511

Additionally, two alternative models were developed by utilizing multi-layer perceptrons for classification. The key observations from these models were as follows:

MLP Model A:

- Data preprocessing: TF-IDF
- Architecture:



MLP Model B:

- Data preprocessing:

```
# extract features

# get the word tokens
train_tokens = [tokenize(text) for text in x_train]
validation_tokens = [tokenize(text) for text in x_valid]

# lowercase
train_tokens = [lower(t) for t in train_tokens]
validation_tokens = [lower(t) for t in validation_tokens]

# stemming
train_stemmed = [stem(tokens) for tokens in train_tokens]
validation_stemmed = [stem(tokens) for tokens in validation_tokens]

# filter the stopwords
train_stemmed = [filter_stopword(tokens) for tokens in train_stemmed]
validation_stemmed = [filter_stopword(tokens) for tokens in validation_stemmed]

# get the bigram list
train_2_gram = [n_gram(tokens, 2) for tokens in train_stemmed]
validation_2_gram = [n_gram(tokens, 2) for tokens in validation_stemmed]

# get the trigram list
train_3_gram = [n_gram(tokens, 3) for tokens in train_stemmed]
validation_3_gram = [n_gram(tokens, 3) for tokens in validation_stemmed]


# build the feature list
train_feats = list()
for i in range(len(x_train)):
       train_feats.append(
              train_stemmed[i] + train_2_gram[i] + train_3_gram[i])

validation_feats = list()
for i in range(len(x_valid)):
       validation_feats.append(
              validation_stemmed[i] + validation_2_gram[i] + validation_3_gram[i])

# build a mapping from features to indices
feats_dict = get_feats_dict(chain.from_iterable(train_feats), min_freq = 6)

# build the feats_matrix
# convert each example to a ont-hot vector, and then stack vectors as a matrix
train_feats_matrix = np.vstack([get_onehot_vector(f, feats_dict) for f in train_feats])
validation_feats_matrix = np.vstack([get_onehot_vector(f, feats_dict) for f in validation_feats])

# convert labels to label_matrix
###num_classes = max(y_train)
num_classes = y_train.nunique()
print("number of classes:", num_classes)

# convert each label to a ont-hot vector, and then stack vectors as a matrix
train_label_matrix = keras.utils.to_categorical(y_train - 1, num_classes = num_classes)
validation_label_matrix = keras.utils.to_categorical(y_valid - 1, num_classes = num_classes)
```

- Architecture:

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Trainable
=================================================================
 dense1 (Dense)              (None, 64)                822784    Y

 dense2 (Dense)              (None, 32)                2080      Y

 dropout1 (Dropout)          (None, 32)                0         Y

 batch_norm1 (BatchNormaliz  (None, 32)                128       Y
 ation)

 dense3 (Dense)              (None, 20)                660       Y

 dropout2 (Dropout)          (None, 20)                0         Y

 dense4 (Dense)              (None, 16)                336       Y

 dense5 (Dense)              (None, 5)                 85        Y

=================================================================
Total params: 826073 (3.15 MB)
Trainable params: 826009 (3.15 MB)
Non-trainable params: 64 (256.00 Byte)
_________________________________________________________________
```

MLP Model Validation Performance

MLP Model	Accuracy	Precision	Recall	F1
Model A	0.535	0.43	0.48	0.45
Model B	0.469	0.37	0.43	0.39

The observations from Model A and Model B indicated that the TfidfVectorizer() and the simplicity of the model played crucial roles in achieving high validation accuracy. The TfidfVectorizer() was found to significantly boost the performance, as observed from the hard baseline model. On the other hand, the complexity of the model did not necessarily lead to improved performance. Model B, which employed more layers, more hidden units, and advanced techniques, yielded lower validation accuracy compared to Model A. This suggested that using fewer layers, fewer hidden units, and simpler models could result in better performance for this sentiment analysis problem.

Conclusion

In conclusion, the sentiment analysis project involved data preprocessing, model development, and performance evaluation. The CNN model with different kernel sizes demonstrated competitive validation accuracy. Additionally, alternative models utilizing multi-layer perceptrons were explored, highlighting the importance of TfidfVectorizer() and simplicity in achieving high validation accuracy. The findings from this project can serve as a basis for further research and improvement in sentiment analysis tasks.