CS918-Assignment 2: Report

Loading

All the necessary libraries are imported at the beginning. The datasets are uploaded into the 'data' dictionary. For each of the datasets, the tweetids, tweets and tweet sentiments are uploaded into the 'tweets', 'tweetids', 'tweetgts'. In the 'tweetgts_num' dictionary, positive and neutral tweets are encoded as 0, and the negative tweets are encoded as 1.

Preprocessing

The tweets in all of the sets are cleaned in the following order:

a. '@username' handles are not useful in sentimental analysis. So these are removed using regex.
b. Any URLs are removed.
c. Any non alpha characters, except hashtags are removed.
d. Any numbers are removed.
e. Stopwords with three or less characters are removed, as they may not be meaningful to our analysis.

Removing any characters that are not relevant to the analysis increases the performance of the classifier. We will use this preprocessed data to create tfidf vectors. We do not use stemming on the tokenized data as it creates meaningless tokens or tokens with different meanings than the context.

Tweets preliminary analysis

The given datasets are imbalanced. There are very few negative tweets compared to positive and negative tweets.

```
In [39]: # check for class imbalance
         train_df['sentiments'].value_counts()

Out[39]: neutral     20789
         positive    15986
         negative     8326
         Name: sentiments, dtype: int64
```

Fig 1: Checking for class imbalance

But it is more important to classify the negative tweets correctly as hateful tweets often depletes the harmony in a social media platform. Hence, this analysis labels both 'positive' and 'negative' tweets as 0, and negative tweets as 1. Accuracy may not be a good estimator of performance as the dataset is imbalance. Hence, we use the macro averaged F1 score as the performance indicator.

F1 score

The F1 score is a good performance indicator for imbalanced datasets. It is calculated as the harmonic mean of a system's precision and recall values. F1 scores can take values between 0 and 1. Values close to 0 indicate a poor performance. The F1 scoring metric that we use is macro averaged, which means the estimator considers all the classes equally. Since the data is imbalanced, with few negative tweets in each of the sets, any score above 0.5 indicates that it's not a bad classifier. A random classifier that generates random classes for the tweets in our datasets has an F1 score around 0.35.

```
In [49]: import random
         testset = 'twitter-dev-data.txt'
         prediction_list = list(np.zeros(len(tweetids[testset])))
         for i in range(len(prediction_list)):
             prediction_list[i]=random.randint(0,1)

In [51]: for i in range(len(prediction_list)):
             if prediction_list[i] == 0:
                 prediction_list[i] ='positive'
             else:
                 prediction_list[i] ='negative'

In [53]: id_preds = dict(zip(tweetids[testset], prediction_list))
         evaluate(id_preds, testset, 'Random Classifier')

         twitter-dev-data.txt (Random Classifier): 0.376
```

Fig 2: Performance of a random classifier on the development set

So any classifier that has a higher F1 score is good. Values close to 1 indicate excellent performance. The higher the F1 score the better the classifier.

Features

We use the TF-IDF features in the traditional models, and word embeddings in the LSTM based model.

TF-IDF

The TfidfVectorizer from sklearn is used to create a TF-IDF feature matrix on a corpus containing the 'twitter-training-data' and the 'twitter-dev-data'. We separate the train and test vectors from the tfidf matrix by splicing. These are then used to train the traditional machine learning models.

Glove embedding

Encoding the datasets

The torchtext tokeniser is used to build a vocabulary, which is the unique list of words we are going to use, from the training and validation datasets. 'encode_sentence' is a function to create a dictionary of unique words and the numbers that are going to represent them. We

set a default index for ‘<UNK>’. The encoded tweets are stored in the dictionary: ‘tweets_encoded’.

Create Pytorch dataset

Since the data are not uploaded as tensors, we need to convert them into a form that can be processed using PyTorch. The class ‘ReviewsDataset’ is designed for this purpose. The training and validation data are converted and loaded into the data loaders: train_dl and val_dl, which enables mini-batch gradient descent. This reduces the computation time and memory usage significantly.

Word Embedding Matrix

The glove vectors for 400000 words are loaded using the load_glove_vectors function into a dictionary, and the embedding matrix is obtained using the get_emb_matrix function. A zero vector is added at the beginning for padding. And a random vector is assigned for unknown words. The remaining vectors in the matrix are obtained using the words in our vocabulary, by matching with the vectors from the glove dictionary.

Traditional Classifiers

The following traditional classifiers are implemented

1. MaxEnt(Logistic Regression)
2. Naive Bayes
3. Decision tree

Deep learning based classifiers

The neural networks implemented are designed as follows:

LSTM base classifier

The first neural network model is a base classifier built on an embedding layer, an LSTM layer and a linear layer.

Bi_LSTM classifier

The base model is modified to include a bidirectional LSTM layer, three linear layers with batch normalisation and ReLU activation, and a final linear layer with a sigmoid activation function.

Bi_LSTM_GRU classifier

The Bi_LSTM classifier is modified to include a GRU layer between the first two linear layers.

We use Adam as the optimiser, and Cross Entropy as the loss function. A learning rate of 1e-3 is used in each of the cases. The models are trained on the train sets and optimised using the development set. The performances are assessed on each of the test sets.

Performance measures and confusion matrix

We use the F1 macro averaged score as the performance estimator. Visualising the confusion matrix on each testset, we can see that most of the positive and neutral tweets are classified as positive, and that a majority of the negative tweets are classified as negative.

```
In [36]: print(classifier)
         print(testset)
         confusion(id_preds, testset, classifier)
```

```
Bi_LSTM_GRU
twitter-test3.txt
             positive  negative  neutral
positive     0.464     0.113     0.423
negative     0.227     0.427     0.347
neutral      0.000     0.000     0.000
```

Fig 3: Demonstration of the performance of Bi_LSTM_GRU on each of the class labels using a Confusion matrix.

Comparison of models

Classifier	Random Classifier (a base case)	MaxEnt(Logistic Regression)	NaiveBayes	Decision Tree	LSTM base classifier	Bi_LSTM_GRU classifier	Bi_LSTM classifier
Validation set	0.376				0.530	0.520	0.507
Testset1		0.467	0.487	0.261	0.492	0.543	0.544
Testset2		0.495	0.509	0.275	0.544	0.551	0.566
Testset3		0.456	0.484	0.255	0.490	0.503	0.522

```
semeval-tweets/twitter-test2.txt (bow-Logistic Regression): 0.495
/Users/aryamathew/opt/anaconda3/envs/tensorflow/lib/python3.9/site-pack
nvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
  n_iter_i = _check_optimize_result(
semeval-tweets/twitter-test3.txt (bow-Logistic Regression): 0.456
Training Decision Tree
semeval-tweets/twitter-test1.txt (bow-Decision Tree): 0.261
semeval-tweets/twitter-test2.txt (bow-Decision Tree): 0.275
semeval-tweets/twitter-test3.txt (bow-Decision Tree): 0.255
semeval-tweets/twitter-test1.txt (bow-Naive Bayes): 0.487
semeval-tweets/twitter-test2.txt (bow-Naive Bayes): 0.509
semeval-tweets/twitter-test3.txt (bow-Naive Bayes): 0.484
Training LSTM
semeval-tweets/twitter-test1.txt (bow-LSTM): 0.492
semeval-tweets/twitter-test2.txt (bow-LSTM): 0.544
semeval-tweets/twitter-test3.txt (bow-LSTM): 0.490
Training Bi_LSTM
semeval-tweets/twitter-test1.txt (bow-Bi_LSTM): 0.544
semeval-tweets/twitter-test2.txt (bow-Bi_LSTM): 0.566
semeval-tweets/twitter-test3.txt (bow-Bi_LSTM): 0.522
Training Bi_LSTM_GRU
semeval-tweets/twitter-test1.txt (bow-Bi_LSTM_GRU): 0.543
semeval-tweets/twitter-test2.txt (bow-Bi_LSTM_GRU): 0.551
semeval-tweets/twitter-test3.txt (bow-Bi_LSTM_GRU): 0.503

Training Logistic Regression
/Users/aryamathew/opt/anaconda3/envs/tensorflow/lib/python3.9/site-pa
nvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as sho
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options
    https://scikit-learn.org/stable/modules/linear_model.html#logisti
  n_iter_i = _check_optimize_result(
semeval-tweets/twitter-test1.txt (bow-Logistic Regression): 0.467
```

Fig 4: F1 scores obtained on each of the testsets

A random classifier that generates random labels for the development set gives an F1 score of 0.356. This can be considered as a base classifier to compare the remaining classifiers. The Decision tree classifier has a poor performance compared to the random classifier. The classifiers based on deep learning(LSTM, Bi_LSTM, Bi_LSTM_GRU) perform better on both the development and the test sets. In all of the LSTM based models, the performance on the test sets are better than on the development set. Among the traditional classifiers the Gaussian Naive Bayes classifier performs the best, although its F1 scores are worse than the least performing LSTM based model.