Programming Assignment - 2

Report

Karan Dave (kdave)

Task: Classification of responses into 4 major categories: Irrelevant, Agreed, Answered and Attacked

Dataset:

- Train Dataset [1640 datapoints] - [994 answered, 299 attacked, 286 irrelevant, 61 agreed]
- Test Dataset [410 datapoints] - [320 answered, 39 attacked, 38 irrelevant, 13 agreed]

The dataset is quite unbalanced and makes it very difficult for the classifier to classify minor classes.

Mapping of classes – {0: answered, 1: attacked, 2: irrelevant, 3: agreed}

Baseline Features:

- Sentence Embeddings – The sentence embeddings can be determined by using various pretrained models such as glove and spacy's encore_web_lg. Spacy's pretrained model works better as it contains a very large training corpus as compared to others. The sentence embeddings are calculated for each question and response. The pretrained models have 300-dimensional vector representation. Concatenating the two vectors would lead to higher dimensional vector space. To reduce the dimensional space, we can apply dimensionality reduction using PCA such that it captures maximum variance of the data.

  Code Snippets:

```
def vector_representation(sentence):
    doc=nlp(sentence)
    vector=doc.vector
    return vector

def get_vectors(df,col):
    df[col+'_vector_rep']=df[col].apply(lambda x: vector_representation(x))
    return df

## Get vector represenation of question and response
vectors_df=get_vectors(df,'question')
vectors_df=get_vectors(df,'response')

## Concatenate the question and response vectors
vectors=[]
question_vectors=df['question_vector_rep'].tolist()
response_vectors=df['response_vector_rep'].tolist()

for i in range(len(question_vectors)):
  vector=np.concatenate((question_vectors[i],response_vectors[i]),axis=None)
  vectors.append(vector)


vectors_df['vector']=vectors
```

- POS Tagging - The part of speech tagging is done to get the syntactic structure of the sentence. The part of speech tags is identified for both the question and response. We then create a mapping for the pos tags and then create a count vectorizer for questions and responses. Since the pos tags for the sentences cannot be directly concatenated, we need to represent them as encoding vectors. The count vectorizer representation is helpful for classification as it gives importance to the pos tags.

  Code Snippets:

```
## POS Tagging Features
## Generate pos tags for both questions and responses
'''
question_pos_rep - Pos tag representations of questions
response_pos_rep - Pos tag representations of responses
pos-tags - Concatenation of both vectors
'''

def pos_tags_representation(sentence):
  if len(sentence)!=0:

    tagged_tokens=nltk.pos_tag(sentence.split())
    words,tags=zip(*tagged_tokens)
    return list(tags)
  else:
    return []

def get_pos_tags(df,col):
  df[col+'_postag_rep']=df[col].apply(lambda x: pos_tags_representation(x))
  return df
```

```
# Concatenate the pos tags of both question and respone sentences
def concatenate_tags():
  tags_df=get_pos_tags(vectors_df,'question')
  tags_df=get_pos_tags(vectors_df,'response')
  tags_df['pos-tags']=tags_df['question_postag_rep']+tags_df['response_postag_rep']
  return tags_df

tags_df=concatenate_tags()
tags_df.head()
```

```
## Create a mapping for the pos_tags so that number of diffent type of pos tags are represented by count vecto
all_tags=[]
for i in range(len(tags_df)):
  all_tags+=(tags_df['pos-tags'].iloc[i])
tags_list=list(set(all_tags))
mapping={}
for i in range(len(tags_list)):
  mapping[tags_list[i]]=i

def pos_vector_mapping(pos_list):
  one_hot=np.zeros(len(mapping))
  for i in pos_list:
    one_hot[mapping[i]]+=1

  return one_hot

tags_df['pos_tag_vector']=df['pos-tags'].apply(lambda x: pos_vector_mapping(x))
tags_df.head()
```

Baseline Model:

- The baseline model consists of both the sentence embeddings and the pos tagging. The features can be concatenated, and weights can be given to each of the features.

```
SVC Report
[[282  15  14   9]
 [ 18  14   3   4]
 [ 20   9   7   2]
 [  7   2   0   4]]
              precision    recall  f1-score   support

           0       0.86      0.88      0.87       320
           1       0.35      0.36      0.35        39
           2       0.29      0.18      0.23        38
           3       0.21      0.31      0.25        13

    accuracy                           0.75       410
   macro avg       0.43      0.43      0.43       410
weighted avg       0.74      0.75      0.74       410
```

```
Logistic Regression
[[296  13  10   1]
 [ 26   9   4   0]
 [ 22   9   7   0]
 [ 10   0   1   2]]
              precision    recall  f1-score   support

           0       0.84      0.93      0.88       320
           1       0.29      0.23      0.26        39
           2       0.32      0.18      0.23        38
           3       0.67      0.15      0.25        13

    accuracy                           0.77       410
   macro avg       0.53      0.37      0.40       410
weighted avg       0.73      0.77      0.74       410
```

New Features:

- Sentiment Polarity – The sentiment polarity of the question and response can indicate whether the response is related to question or not by considering pair of both question and response polarity. The pairing of those polarity can help in indicating the response of the question.

```
## Generate sentiment values for the questions and response

def sentiment_vector(sentence):
  analyser = SentimentIntensityAnalyzer()
  score_dict=analyser.polarity_scores(sentence)
  return score_dict['compound']


tags_df['question_sentiment']=tags_df['question'].apply(lambda x: sentiment_vector(x))
tags_df['response_sentiment']=tags_df['response'].apply(lambda x: sentiment_vector(x))
```

- Cosine Similarity - Cosine Similarity is the measure of how close the two sentences are. The cosine similarity can help in indicating how related the response is to the question. It can help in determining the answered and irrelevant class. Irrelevant class will show a very less value for cosine similarity since the embeddings between the two would not be much similar. The opposite goes for the answered as the response would have many similar word representations as the question.

```
def get_cosine_similarity(feature_vec_1, feature_vec_2):
    return cosine_similarity(feature_vec_1.reshape(1, -1), feature_vec_2.reshape(1, -1))[0][0]

cosines=[]
for i in range(len(tags_df)):
  cosines.append(get_cosine_similarity(tags_df['question_vector_rep'].iloc[i],tags_df['response_vector_rep'].i
tags_df['cosine_similarity']=cosines
```

Other features such as sentence-based and word-based features can be used. However, they do not seem to improve the minor classes much. Being imbalanced dataset, new features should be added in order to improve the importance of minority classes.

Baseline Models with new Features:

```
SVC Report
[[313   4   2   1]
 [ 32   7   0   0]
 [ 25   9   4   0]
 [ 10   0   1   2]]
              precision    recall  f1-score   support

           0       0.82      0.98      0.89       320
           1       0.35      0.18      0.24        39
           2       0.57      0.11      0.18        38
           3       0.67      0.15      0.25        13

    accuracy                           0.80       410
   macro avg       0.60      0.35      0.39       410
weighted avg       0.75      0.80      0.74       410
```

```
Logistic Regression
[[297  13   9   1]
 [ 26  11   2   0]
 [ 22   9   7   0]
 [  9   0   2   2]]
              precision    recall  f1-score   support

           0       0.84      0.93      0.88       320
           1       0.33      0.28      0.31        39
           2       0.35      0.18      0.24        38
           3       0.67      0.15      0.25        13

    accuracy                           0.77       410
   macro avg       0.55      0.39      0.42       410
weighted avg       0.74      0.77      0.75       410
```

Model Comparison:

Evaluation Metric:

- Accuracy:

	KNN	Linear SVC	Logistic Regression
Baseline Model	0.72	0.71	0.77
Baseline + New features	0.72	0.79	0.78

Based on terms of accuracy, we see that the model with new features integrated performs better then the baseline model. Also, Linear SVC seems to be performing the best among the three classifiers. However, accuracy cannot be considered as the only evaluation metric because of imbalanced distribution of class.

- Precision (Weighted average):

	KNN	Linear SVC	Logistic Regression
Baseline Model	0.67	0.72	0.74
Baseline + New features	0.67	0.74	0.75

- Recall (Weighted average):

	KNN	Linear SVC	Logistic Regression
Baseline Model	0.72	0.71	0.77
Baseline + New features	0.72	0.79	0.78

- F1 Score (Weighted average):

	KNN	Linear SVC	Logistic Regression
Baseline Model	0.69	0.71	0.74
Baseline + New features	0.69	0.74	0.76

Based on precision, recall and F1 score model with new features works better then the baseline model. However, the new model showed a very little difference in the above evaluation metrics as the new features were not able to vary well classify the minor classes.

Conclusion:

- Sentence embeddings seems to be the major feature in classifying different classes.
- Sentence embeddings represented the responses into different classes very well.
- Pos Tags can help in identifying the syntactic structure of the response and can help in classifying the responses.
- Features such as sentiment score and cosine similarity help in identifying the answered and irrelevant classes. However, the new features did not seem to work very well for identifying the minor classes.
- Classification of imbalanced dataset was a major issue for the classifiers. With a very less training data for the minor classes, the features were not able to create a distinct difference among the classes.