NLP P2 Report

Unity id: sdinaka

The problem to be solved is that given an interaction of the form Question : Answer, we need to classify the interaction into one of the following { irrelevant, agreed, answered, attacked }.

Since the task is classification and the problem is non-trivial. We need to do feature engineering and add defining features that can enable the classifier through the task of finding the sentiment of interaction.

Baseline Features

The baseline features that are considered for the problem are

1. sentence embedding
2. Parts- of-speech tags

Baseline Outline

The baseline is built on the basis of the given method outline

1. Get sentence embedding for the question as a float vector of length n in spacy.
2. Get sentence embedding for the answer as a float vector of length n .

```
question_emb=[]
question_emb_test=[]
for line in Q:
   #question_emb.append([sbert_model.encode(line[0])])
  doc = nlp(line[0])
  question_emb.append(doc.vector)

for line in Q_test:
  doc = nlp(line[0])
  question_emb_test.append(doc.vector)
```

3. Then concatenate the sentence embeddings together and get a feature vector of length 2n.

```
train_features=[]

for q_emb,a_emb in zip(question_emb, answer_emb):
  train_features.append(q_emb+a_emb)
```

4. Get the POS tags form question and answer.

```
train_tagged_q=[]
train_tagged_a=[]
for line in Q:
    wordsList = nltk.word_tokenize(line[0])
    train_tagged_q.append(nltk.pos_tag(wordsList) )
for line in A:
    wordsList = nltk.word_tokenize(line[0])
    train_tagged_a.append(nltk.pos_tag(wordsList) )
```

```
[['MD', 'PDT', 'DT', 'NNS', 'IN', 'DT', 'NN', 'NN', 'IN', '.', 'PRP', 'MD', 'RB', 'VB', 'NNS', 'IN', 'JJ', 'NNS', '.', 'CC', 'PRP
[['RB', 'IN', 'NNP', 'VBD', 'JJ', '(', 'VB', 'POS', 'VB', 'PRP', 'CC', 'NNP', 'VBP', 'DT', 'IN', 'JJ', 'NN', ')', 'CC', 'NNP', 'V
```

5. Train a label binarizer to for each part of speech tag, give a vector with 1 in the place of the tag,

```
from sklearn.preprocessing import LabelBinarizer

encoder = LabelBinarizer()
transfomed_label = encoder.fit_transform(["","CC","CD","DT","EX","FW","IN","JJ","JJR","JJS","LS
labels = encoder.classes_
mappings = {}
for index, label in zip(range(len(labels)), labels):
  mappings[label]=index
pos_feature=[]
```

6. Add all vectors of the tags together for a sentence to get a unique feature vector for each question and answer,

```
for sent in range(len(feature_train_vector)):
  res_temp=np.zeros(len(mappings))
  for j in range(len(feature_train_vector[sent])):
    if len(feature_train_vector[sent])>1 and feature_train_vector[sent][j] in mappings.keys():
      res_temp+=(transfomed_label[mappings[feature_train_vector[sent][j]]])
    res_temp+=transfomed_label[mappings[""]]
    res_temp=np.asarray(res_temp)
  pos_feature.append(res_temp)
```

7. The answer and question vectors are concatenated with each other

8. The above 2 feature vectors are concatenated together to a combined feature vector for adn interaction that is trained using different classifiers.

```
combined_train=[]
for emb,pos in zip(train_features,pos_feature):
  combined_train.append(np.concatenate((emb,pos), axis=None))
combined_train=np.vstack(combined_train)
combined_train.shape
```

9. The classifiers used here are

10. SVM classifier:

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')

svclassifier.fit(combined_train, Label)
y_pred = svclassifier.predict(combined_test)
```

11. KNN classifier:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5)
Label=np.asarray(Label)
classifier.fit(combined_train, Label)
y_pred = classifier.predict(combined_test)
```

12. Use the test features on the above 2 classifiers and record performance

Baseline Performance: Evaluation Metrics

1. KNN :

The KNN algorithm works on the intuition that after plotting the points on 2-d(linear) we segregate based on the given labels in the training set. The classification boundary is set based on the neighboring points here set to 5.
The KNN doesn't need to be trained unlike SVM, hence it performs based on the distribution of the testing set.

```
accuracy_score :  0.6804878048780488
precision_score :  0.62119327068868
recall_score :  0.6804878048780488
f1_score :  0.6487228471543905
```

.

2. SVM :

The Support Vector Machine finds the optimum boundary for better classification

```
accuracy_score :  0.7682926829268293
precision_score :  0.6680325508034836
recall_score :  0.7682926829268293
f1_score :  0.7034976846392168
```

Proposed Features:

1. Sentiment Score
2. Cosine similarity between question and answer for a specific interaction

Reason for choice:

1. Sentiment score:

The choice of sentiment score is so that based on the sentiment the classifier can classify the interaction as
We take the sentiment of question and answer in a range of 0 to 1.
This can help with classification of the interaction.
Eg: if the sentiment of the answer is positive and the word embeddings are close then it can be classified to be agreed interaction

- If in the above case, the sentiment of the answer is neutral then it can be classified to be answered.

Implementation:

```
for line in Q:
    result = TextBlob(line[0]).sentiment.polarity
    train_sentiment_q.append([result])

for line in A:
    result = TextBlob(line[0]).sentiment.polarity
    train_sentiment_a.append([result])
```

The sentiment analysis can be done in an in-built package, called TextBlob, whose polarity returns a float value between 0 and 1.

2. Cosine Similarity:

Once we summarise the question/answer to a single sentence, we can calculate the cosine similarities between the summarised question and answer, this feature can capture how close the question and answer can be in term of semantic meaning.

This can also be used as a feature since for example, if the interaction is irrelevant, then the cosine similarity high,
Implementation:

```
for line in Q:
  q = word_tokenize(line[0])
  Q_set.append([w for w in q if not w in sw])
```

After preprocessing

```
  for q_set,a_set in zip(Q_set,A_set):
    all_set = q_set + a_set
    for w in all_set:
      if w in q_set: l1.append(1) # create a vector
      else: l1.append(0)
      if w in a_set: l2.append(1)
      else: l2.append(0)
    c = 0
```

Create a vector

```
  for i in range(len(all_set)):
        c+= l1[i]*l2[i]
  cosine = c / float((sum(l1)*sum(l2))**0.5)
  train_cosine_feature.append([cosine])
train_cosine_feature=np.vstack(train_cosine_feature)
print(train_cosine_feature.shape)
```

Get the cosine vector

And concatenate all the feature vectors created so far. And pass the same to the classifier for training.

Proposed model : Evaluation metrics

1. KNN

```
accuracy_score :  0.6804878048780488
precision_score :  0.62119327068868
recall_score :  0.6804878048780488
f1_score :  0.6487228471543905
```

2. SVM:

```
accuracy_score :  0.7658536585365854
precision_score :  0.6651335163846221
recall_score :  0.7658536585365854
f1_score :  0.7026023169580139
```

3. Random Forest :

```
accuracy_score :  0.7804878048780488
precision_score :  0.6091612135633552
recall_score :  0.7804878048780488
f1_score :  0.6842632809889744
```

Combining vectors:

```
combined_train = np.vstack(combined_train)
```

Comparison

Baseline SVM:

```
              precision    recall  f1-score   support

    agreed       0.20      0.15      0.17        13
  answered       0.79      0.96      0.87       320
  attacked       0.00      0.00      0.00        39
irrelevant       0.45      0.13      0.20        38
```

Proposed SVM:

```
              precision    recall  f1-score   support

    agreed       0.18      0.15      0.17        13
  answered       0.80      0.96      0.87       320
  attacked       0.00      0.00      0.00        39
irrelevant       0.42      0.13      0.20        38
```

The proposed SVM, is better in identifying “answered” but less precise with agreed and irrelevant.

This is due to adding 3 more features corresponding to .

- Sentiment of question.
- Sentiment of answer.
- Cosine similarity between question and answer.

These 3 features mis-identifies irrelevant and agreed since,

- When the interaction is irrelevant, the sentiment of question and answer doesn’t have a meaning, when we try to classify based on this feature, we get the wrong answer.

- When the interaction is agreed, cosine sometimes, well, that's right can be neutral, resulting in misclassification.

Conclusion

The Random Forest classifier predicted the interaction type with highest accuracy among KNN, SVM.

The proposed features of sentiment score and text summarisation and cosine distance didn't improve the performance significantly beyond the usage of sentence embedding and pos tags.

The following possible implementations can be added to make the solution predict with better precision.

- Using ensemble models to train each feature and then boosting.
- The lexicon feature can be used for each interaction type.
- The wordshape can be used, since agreed type almost always follows. Xxxx. word shape, where the word is followed by a fullstop.