Report on Opinion Mining In Long Sentences

Unity id: sdinaka

Baseline Report and Classification

The baselines that I've chosen are

- the TFIDF for vector generation

  And
- Spacy model for vector generation

And Support vector Classifier for classification of sentences into { negative, neutral and positive}

TFIDF:

Each document is converted to a row of the TF-IDF matrix and each word is stored in a column vector. With no preprocessing even the stopwords and punctuations are taken as a column.

The TFIDF gives weights for each cell with the frequency of their occurrence giving higher weights to rare words across given documents. The whole corpus needs to be fitted with the vectorisor on formation of matrix since each word(even in test) should be available for prediction in the corpus.

```
def tf_idf_baseline():
    vectoriser = TfidfVectorizer()
    doc_term_matrix = vectoriser.fit_transform(corpus)

    # Converting the labels from strings to binary
    # tfidf_model = LinearSVC()
    # tfidf_model.fit(doc_term_matrix[0: train_cutoff], label)
    # pickle.dump(tfidf_model, open("G:/NLP/hw1/opinion_mining/tfidf_baseline", 'wb'))

    tfidf_model = pickle.load(open("G:/NLP/hw1/opinion_mining/tfidf_baseline", 'rb'))

    tfidf_prediction = tfidf_model.predict(doc_term_matrix[train_cutoff: len(corpus)])  # pass matrix
    f1_tfidf = f1_score(tfidf_prediction, test_label, average='macro')
    print("tfidf F1 score : "+f1_tfidf)
    acc_tfidf = accuracy_score(tfidf_prediction, test_label)
    print("tfidf accuracy score : "+acc_tfidf)
```

Spacy:

Has built in models that come with inbuilt word vectors.

HEre the model is loaded in nlp and the vector for that particular sentence is obtained by doc.vector.

This vector takes each word and gets the vector of it from using the model as space and averages the vector for each word, into a sentence.

```
# Load the spacy model that you have installed
def spacy_baseline():
    nlp = spacy.load('en_core_web_md')
    spacy_features = []
    for line in corpus:
        doc = nlp(line)
        spacy_features.append(doc.vector)
    # spacy_word_embed = LinearSVC()
    # spacy_word_embed.fit(spacy_features[0: train_cutoff], label)
    #pickle.dump(spacy_word_embed, open("G:/NLP/hw1/opinion_mining/spacy_baseline", 'wb'))

    spacy_word_embed = pickle.load(open("G:/NLP/hw1/opinion_mining/spacy_baseline", 'rb'))

    spacy_prediction = spacy_word_embed.predict(spacy_features[train_cutoff: len(corpus)])  # pass matrix
    f1_spacy = f1_score(spacy_prediction, test_label, average='macro')
    print("spacy F1 score : ",f1_spacy)
    acc_spacy = accuracy_score(spacy_prediction, test_label)
    print("spacy accuracy score : ",acc_spacy)
```

Comparison:

1. TFIDF: since TFIDF calculates the vector with respect to each word in the sentence for all given sentences. The words that appear rarely dominate with its sentiment. Hence the classification for the sentiment of the whole sentence is biased.
2. Spacy: The spacy also gives equal weightage to all words but calculates the vectors considering each word separately in the sentence. Hence the accurate classification is affected by each word rather than the whole sentence.

Proposed Solution Report and classification

Rank based:

One proposed solution is to increase the window of sentiment classification with n-gram models.

Eg: bigram and trigram models.

We have 3 models trained with unigram , bigram and trigram. And each of them predict the mood of the sentence with some confidence.

Using this confidence score as a matrix, we can predict the mood of sentence by

1. Getting the mode of the prediction
2. Getting highest confidence score among the 3 models'

```
unigram()
bigram()
trigram()
final_rank_prediction = []
final_confidence_prediction = []
for i in range(0, len(spacy_prediction_1)):
    pred1 = np.where(spacy_prediction_1[i] == np.max(spacy_prediction_1[i]))
    pred2 = np.where(spacy_prediction_2[i] == np.max(spacy_prediction_2[i]))
    pred3 = np.where(spacy_prediction_3[i] == np.max(spacy_prediction_3[i]))
    print([pred1[0][0], pred2[0][0], pred3[0][0]])
    pred_i = statistics.mode([pred1[0][0], pred2[0][0], pred3[0][0]])
    final_rank_prediction.append(pred_i)
    all_spacy = spacy_prediction_1[i] + spacy_prediction_2[i] + spacy_prediction_3[i]
    max_index = np.where(all_spacy == np.max(all_spacy))
    max_index = max_index[0][0] % 3
    final_confidence_prediction.append(max_index)

f1_proposed = f1_score(final_rank_prediction, label[train_cutoff + 1: len(text)], average='macro')
print("f1_final_rank_prediction", f1_proposed)

acc_spacy = accuracy_score(final_rank_prediction, label[train_cutoff + 1: len(text)])
print("accuracy_final_rank_prediction", acc_spacy)

f1_proposed = f1_score(final_confidence_prediction, label[train_cutoff + 1: len(text)], average='macro')
print("final_confidence_prediction", f1_proposed)

acc_spacy = accuracy_score(final_confidence_prediction, label[train_cutoff + 1: len(text)])
print("final_confidence_prediction", acc_spacy)
```

Both these methods perform inferior to the baselines

Hence used Universal encoder that instead of each word classifies based on the whole sentence. By creating word vectors with respect to sentences rather than words.

```
with tf.Session() as session:
  session.run([tf.global_variables_initializer(), tf.tables_initializer()])
  message_embeddings = session.run(embed(text))
  test_embeddings = session.run(embed(test_text))

  #model = LinearSVC()
  #model.fit(message_embeddings, label)

  #pickle.dump(model, open("G:/NLP/hw1/opinion_mining/universal", 'wb'))
  model = pickle.load(open("G:/NLP/hw1/opinion_mining/universal", 'rb'))

  prediction = model.predict(test_embeddings)  # pass matrix
  f1_proposed = f1_score(prediction, test_label, average='macro')
  print("universal F1 score : ",f1_proposed)
  acc_proposed = accuracy_score(prediction, test_label)
  print("universal accuracy score : ",acc_proposed)
```

Which performs slightly better than the baselines.

Metrics for Performance

Have calculated the metrics with F1 score and accuracy.
Baselines:

```
G:\NLP\hw1\opinion_mining>python baseline.py
tfidf F1 score :  0.4337027176826842
tfidf accuracy score :  0.5592972181551976
C:\Users\ASUS\AppData\Local\Programs\Python\Python37\lib\site-packa
core_web_md' (2.2.0) requires spaCy v2.2 and is incompatible with
expected results or runtime errors. To resolve this, download a ne
 the current spaCy version. For more details and available updates
  warnings.warn(warn_msg)
spacy F1 score :  0.465927880369056
spacy accuracy score :  0.5885797950219619
```

Proposed solution(Universal):

```
G:\NLP\hw1\opinion_mining>python proposed_solution_3.py
2020-09-06 11:08:02.524728: W tensorflow/stream_executor/platform/default
2020-09-06 11:08:02.538819: I tensorflow/stream_executor/cuda/cudart_stub
WARNING:tensorflow:From C:\Users\ASUS\AppData\Roaming\Python\Python37\sit
and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
2020-09-06 11:08:13.949017: I tensorflow/stream_executor/platform/default
2020-09-06 11:08:14.683596: I tensorflow/core/common_runtime/gpu/gpu_devi
pciBusID: 0000:02:00.0 name: GeForce MX250 computeCapability: 6.1
coreClock: 1.582GHz coreCount: 3 deviceMemorySize: 2.00GiB deviceMemoryBa
2020-09-06 11:08:14.702018: W tensorflow/stream_executor/platform/default
2020-09-06 11:08:14.717083: W tensorflow/stream_executor/platform/default
2020-09-06 11:08:14.763300: I tensorflow/stream_executor/platform/default
2020-09-06 11:08:14.776921: I tensorflow/stream_executor/platform/default
2020-09-06 11:08:14.826663: I tensorflow/stream_executor/platform/default
2020-09-06 11:08:14.863168: W tensorflow/stream_executor/platform/default
2020-09-06 11:08:14.888644: W tensorflow/stream_executor/platform/default
2020-09-06 11:08:14.900045: W tensorflow/core/common_runtime/gpu/gpu_devi
ike to use GPU. Follow the guide at https://www.tensorflow.org/install/gp
Skipping registering GPU devices...
2020-09-06 11:08:14.922507: I tensorflow/core/platform/cpu_feature_guard.
rmance-critical operations:  AVX2
To enable them in other operations, rebuild TensorFlow with the appropria
2020-09-06 11:08:14.975235: I tensorflow/compiler/xla/service/service.cc:
2020-09-06 11:08:14.989570: I tensorflow/compiler/xla/service/service.cc:
2020-09-06 11:08:15.000633: I tensorflow/core/common_runtime/gpu/gpu_devi
2020-09-06 11:08:15.012250: I tensorflow/core/common_runtime/gpu/gpu_devi
universal F1 score :  0.4920042817886911
universal accuracy score :  0.6178623718887262
```

F1: 0.49
Accuracy: 0.61

Code snippets

Tokenisation:
In proposed solution:

```
for line in corpus:
    sentence_vector = 0
    words = line.split()
    for i in range(0, len(words)):
        word_vec = nlp(words[i])
        sentence_vector += word_vec.vector
    print(sentence_vector)
    sentence_vector /= len(words)
    spacy_features.append(sentence_vector)
```

Vector calculation:

```
message_embeddings = session.run(embed(text))
```

Classification model generation:

```
svc = LinearSVC()
spacy_word_embed = CalibratedClassifierCV(svc)
spacy_word_embed.fit(spacy_features[0: train_cutoff], label)
```

Train test split:

```
file_name = "G:\\NLP\\hw1\\training_pasted.xlsx"
data = pd.read_excel(r"G:\NLP\hw1\opinion_mining\P1_training.xlsx", encoding='unicode_escape',
header=0, names=["text", "opinion"],
                     error_bad_lines=False, lineterminator='\n')
test_data = pd.read_excel(r"G:\NLP\hw1\opinion_mining\P1_testing.xlsx", encoding='unicode_escape',
header=0, names=["test_text", "test_opinion"],
                     error_bad_lines=False, lineterminator='\n');
text = data['text']
label = data['opinion']

test_text = test_data['test_text']
test_label = test_data['test_opinion']


train_cutoff = int(len(text))
corpus = text
corpus = corpus.append(test_text)
```