P1 REPORT

Archit Kwatra - akwatra

1st Baseline: Word2Vec

Method: Word2Vec embeddings fed to a RandomForest Classifier

Steps Involved:

1) Filter the given sentences by removing stopwords and punctuations.
2) Convert the sentences to lowercase
3) Tokenize the given sentences

```
def filterWords(review):
    text = re.sub("[^a-zA-Z]"," ",review)
    words = text.lower().split()
    stops = set(stopwords.words("english"))
    words = [word for word in words if not word in stops]
    return(words)

def tokenize(review, tokenizer):
    sentences = tokenizer.tokenize(review.strip())
    allSentences = []
    for sentence in sentences:
        if len(sentence)>0:
            allSentences.append(filterWords(sentence))
    return allSentences
```

4) Train with the given data using word2vec and convert the given sentences into embeddings
5) Normalize the data and remove the words which do not appear in the model's vocabulary

```
model = word2vec.Word2Vec(sentences, min_count=min_word_count, sample=downsampling)
```

```
def getFeatureVecs(words, model, num_features):
    featureVec = np.zeros(num_features, dtype="float32")
    totalWords = 0
    index2word_set = set(model.wv.index2word)
    for word in words:
        if word in index2word_set:
            totalWords += 1
            featureVec = np.add(featureVec, model[word])
    featureVec = np.divide(featureVec, totalWords)
    return featureVec

def getAvgFeatures(train_sentences, model, num_features):
    l = len(train_sentences)
    reviewFeatureVecs = np.zeros((l, num_features), dtype="float32")
    for i, review in enumerate(train_sentences):
        reviewFeatureVecs[i] = getFeatureVecs(review, model, num_features)
    return reviewFeatureVecs
```

6) Classify the training data with their corresponding labels using RandomForestClassifier
7) Predict using the test set sentences

```
forest = RandomForestClassifier(n_estimators = 1400)
forest = forest.fit(trainDataVecs, train["label"])
result = forest.predict(testDataVecs)
```

Results:

Accuracy – 53.3%

F1 Score – 37.6

2nd Baseline: TF_IDF

Method: TF_IDF fed to a RandomForest Classifier

Steps Involved:

1) Tokenize the given sentences

```
trainTokens = [sentence for sentence in train['sentence']]
testTokens = [sentence for sentence in test['sentence']]
```

2) Calculate the TF-IDF score for all the tokens using fit_transform

```
forest = RandomForestClassifier(n_estimators = 1400)
forest = forest.fit(trainFeatures, train["label"])
results = forest.predict(testFeatures)
```

3) Predict the label for the test sentences

Results:

Accuracy – 58.8%

F1 Score – 41.7

Proposed Solution: BERT

Method: Bert Tokenization with pretrained BERT Model + Deep Learning Classifier

Steps Involved:

1) Convert the given sentences to tokens using BERT
2) Add data padding and mask to the above tokens

```
def getTokens(data):
  temp = []
  for row in data['sentence']:
      temp.append( ["[CLS]"] + tokenizer.tokenize(str(row)) + ["[SEP]"] )
  tokens = list(map(tokenizer.convert_tokens_to_ids, temp))
  tokens = map(lambda tids: tids + [0] * (sentenceLength - len(tids)), tokens)
  tokens = [tf.convert_to_tensor(xi) for xi in list(tokens)]
  return tokens

x_train = tf.convert_to_tensor(getTokens(train))
x_test = tf.convert_to_tensor(getTokens(test))
```

```
bert_params = params_from_pretrained_ckpt(basePath)
bert_layer = BertModelLayer.from_params(bert_params, name="bert")
bert_layer.apply_adapter_freeze()

def create_model(sentenceLength, classes):
    inputShape = Input(shape=(sentenceLength,), dtype='int32', name='input')
    bertLayer = bert_layer(inputShape)
    cls_out = Lambda(lambda seq: seq[:, 0, :])(bertLayer)
    dropout = Dropout(0.1)(cls_out)
    fc_1 = Dense(64, activation=tf.nn.relu)(dropout)
    dr_2 = Dropout(0.2)(fc_1)
    finalOutputShape = Dense(classes, activation='softmax')(dr_2)
    model = Model(inputShape, finalOutputShape)
    return model
```

3) Initialize a deep learning model with different Dense layers and dropout layers

```python
model = create_model(sentenceLength, totalClasses)
model.build(input_shape=(None, sentenceLength))
load_stock_weights(bert_layer, basePath+"bert_model.ckpt")
def flatten_layers(bert_layer):
    if isinstance(bert_layer, keras.layers.Layer):
        yield bert_layer
    for layer in bert_layer._layers:
        for sub_layer in flatten_layers(layer):
            yield sub_layer

def getLayerInfo(name):
  if layer.name in ["LayerNorm", "adapter-down", "adapter-up"]:
    return True
  return False

for layer in flatten_layers(bert_layer):
  if getLayerInfo(layer.name): layer.trainable = True
  else: layer.trainable = False

bert_layer.embeddings_layer.trainable = False
model.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(lr=0.00001), metrics=['accuracy'])
print(model.summary())
```

```
Done loading 196 BERT weights from: /content/drive/My Drive/NLP/input/bert-pretrained-models/multi_cased_L-12_H-768_A-12/multi_cased_L-12_H-768_A-12/bert_model.c
Unused weights from checkpoint:
	bert/embeddings/token_type_embeddings
	bert/pooler/dense/bias
	bert/pooler/dense/kernel
	cls/predictions/output_bias
	cls/predictions/transform/LayerNorm/beta
	cls/predictions/transform/LayerNorm/gamma
	cls/predictions/transform/dense/bias
	cls/predictions/transform/dense/kernel
	cls/seq_relationship/output_bias
	cls/seq_relationship/output_weights
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           [(None, 256)]             0
_________________________________________________________________
bert (BertModelLayer)        (None, 256, 768)          177261312
_________________________________________________________________
lambda (Lambda)              (None, 768)               0
_________________________________________________________________
dropout (Dropout)            (None, 768)               0
_________________________________________________________________
dense (Dense)                (None, 64)                49216
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 195
=================================================================
Total params: 177,310,723
Trainable params: 49,411
Non-trainable params: 177,261,312
```

4) Classify using the deep learning model
5) Predict the test sentences using the above model

Results:

Accuracy – 46.55%

F1 Score – 28.9

Justification for the Proposed Model:

BERT (Bidirectional Encoder Representations from Transformers) is a language model which creates bidirectional embeddings of the hidden word, which means it predicts the hidden word by looking at words before and after the hidden word. BERT model has been used on complex language tasks by using the power of transfer learning.

I would like to propose BERT as the proposed model, despite of its low accuracy and F1 score, because BERT is known to perform very well on NLP tasks with medium to large data sets. Also, the basic intuition behind BERT is better than the other embedding's models since it predicts the hidden word by looking at its predecessor and successor words which is intuitively better as well.

The BERTS low performance could also be attributed to the below 2 reasons: -

1) The given dataset is very small, and BERT is not a very good choice if the dataset is very small
2) The annotations were not done by professionals and this could induce some amount of inaccuracy resulting in a lower F1 and accuracy score.