DeftEval

Sentence Definition Classification

Data preprocessing

Intro

Subtask 1: Sentence Classification, Given a sentence, classify whether or not it contains a definition.

The training data is 18157 sentences, 12143 of them labeled 0 and the other 6014 labeled 1. The testing data is 853 sentences

Data cleaning

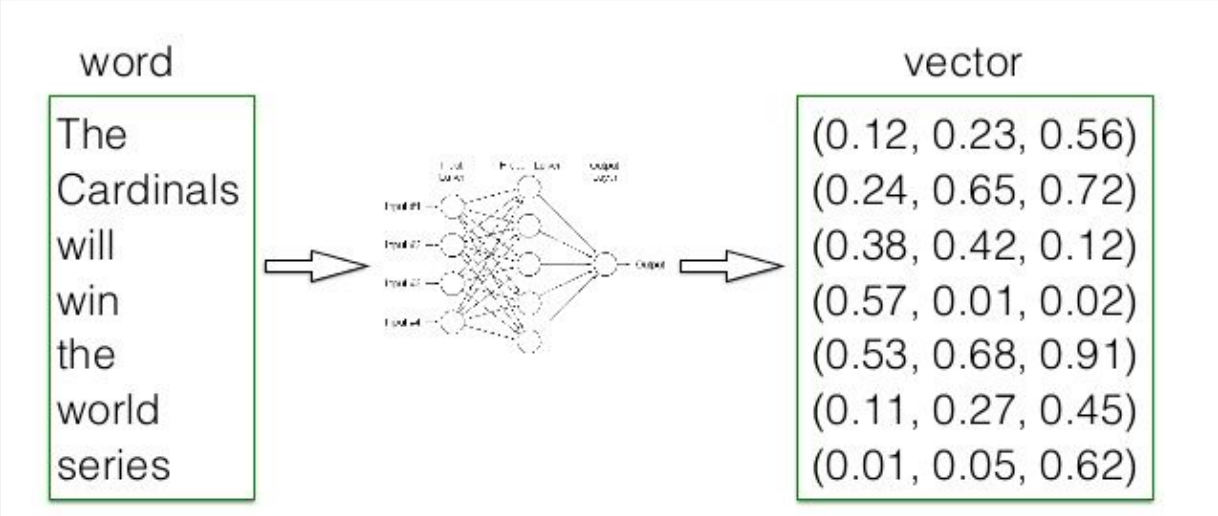
```
" 3247 . [ link ] shows a way of graphing the exchange of a virtual photon between two positive charges ."   "0"
```

Simple_preprocess ()

```
['link', 'shows', 'a', 'way', 'of', 'graphing', 'the', 'exchange', 'of', 'a', 'virtual', 'photon', 'between', 'two', 'positive', 'charges']
```

Word 2 Vec

Word2Vec creates vectors that are distributed numerical representations of word features, features such as the context of individual words.



Sentence Representation

1. **Mean sentence**: by summing the words vector in a sentence and dividing by their count we get a vector that represents the sentence.

2. **Array of vectors with zero pad**: the problem with just encoding each word, is that sentences varies in length, so we decided to zero pad to the length of the longest sentence in the training set.

3. **Keras embedding Layer**: A word embedding layer that sits as the input layer of a neural network model

Classification methods

Nearest Mean

Splitting the training data into positive and negative. After that we generate a sentence mean for the positive data and one for the negative. Each testing example will then be compared to both means and the closest determines the label.

```
[ ] from sklearn.metrics import f1_score,classification_report
    from scipy import spatial
    def nearestMean(trainDocs,trainLabels,testDocs,testLabels):
      posData,negData=splitLabels(trainLabels,trainDocs)
      posMean=generatePositveMean(posData)
      negMean=generateNegitiveMean(negData)

      pred=[]
      for tsen in tdata:
        pos=1 - spatial.distance.cosine(tsen, posVec)
        neg=1 - spatial.distance.cosine(tsen, negVec)
        if(pos>neg):
          pred.append(1)
        else:
          pred.append(0)
      target=['no','yes']
      print(classification_report(testlabels,pred,target_names=target))
```

	precision	recall	f1-score	support
no	0.77	0.69	0.73	573
yes	0.47	0.58	0.52	280
accuracy			0.65	853
macro avg	0.62	0.63	0.62	853
weighted avg	0.67	0.65	0.66	853

Naive Bayes Implementation

This is the implementation of the Naive Bayes that was initially to be used, it uses no libraries for the sklearn, for the creation of the model itself.

Incomplete however, it was halted and another method of implementing the code was used.

```
[ ]
import sklearn
c,v=getCorpus()
print('ooga')
print(len(v))
print(len(c))
print(len(c[0]))
#print(len(c))
#print(c)
uc=distinct(c)
#print(uc)
bigboi = [ [0] * len(uc) for _ in range(len(c))]
print(len(c))
print(len(uc))
#print(bigboi)
print(len(bigboi))
print(len(bigboi[0]))
distinctWords = list(uc)
for x in range(len(uc)):
  for i in range(len(c)):
    for j in range(len(c[i])):
      if distinctWords[x] in c[i][j]:
        bigboi[i][x] = bigboi[i][x]+1
#print(bigboi)
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(bigboi, v)
GaussianNB()
#print(clf.predict([[-0.8, -1]]))
```

Naive Bayes Implementation

Here is the implementation of Naive Bayes that was used, it utilized the libraries from sklearn such as **CountVector** and **MultinomialNB**.

CountVector is an object that can learn the vocabulary of the corpus and transform the data into a matrix, here it returns a sparse matrix.

MultinomialNB is the type of model chosen for this project, as it is used for discrete counts.

```
def classifyUsingIndependantFeatures():

  from sklearn.feature_extraction.text import CountVectorizer
  from sklearn.naive_bayes import MultinomialNB

  dataString, labels = getDataAsString('drive/My Drive/deft_train/')
  testData, testLabels = getDataAsString('drive/My Drive/deft_test/')

  vectorizer = CountVectorizer()
  vector = vectorizer.fit_transform(dataString)
  print(vectorizer.vocabulary_)
  print(vector.shape)
  # print(type(vector))
  # print(vector)
  # print(vector.toarray())

  testDataVectorized = vectorizer.transform(testData)

  NB = MultinomialNB().fit(vector, labels)
  prediction = NB.score(testDataVectorized, testLabels)
  labelsPredicted = NB.predict(testDataVectorized)
  f1Score = f1_score(testLabels, labelsPredicted)

  print("Naive Bayes")
  print("Accuracy: ", prediction)
  print(classification_report(testLabels, labelsPredicted, target_names=targ

  tree = DecisionTreeClassifier(criterion = "entropy", splitter = "random")
  predictModel(tree, vector, labels, testDataVectorized, testLabels)
```

Naive Bayes Implementation

The result is as follows:

The first array shown is the frequency of each word in the vocab.

A classification report is called that prints out the table shown right under the accuracy, which is not included in the report and is calculated it before.

```
{'3247': 2358, 'link': 17347, 'shows': 23582, 'way': 26752, 'of': 19418, 'graphing': 14681,
(18157, 27234)
Naive Bayes
Accuracy:  0.753810082063306
              precision    recall  f1-score   support

          no       0.81      0.83      0.82       573
         yes       0.63      0.60      0.61       280

    accuracy                           0.75       853
   macro avg       0.72      0.71      0.72       853
weighted avg       0.75      0.75      0.75       853

Model DecisionTreeClassifier
Accuracy:  0.7526377491207503
              precision    recall  f1-score   support

          no       0.80      0.84      0.82       573
         yes       0.64      0.56      0.60       280

    accuracy                           0.75       853
   macro avg       0.72      0.70      0.71       853
weighted avg       0.75      0.75      0.75       853
```

Sentence Vector Means method

- Get the data in the form of Mean sentence vectors as explained earlier to be in shape: (18157, 100)
- Our Test data is of shape (853, 100)
- Create Decision Trees model
- Create KNN model
- Create Logistic Regression model
- Predict the labels and compare them with the ground truth to get the accuracy and f1_measure
- The f1_measure is more important as the positive and negative classes are not the same size in training data
- Almost 12000 negative data and 6000 positive

Decision Trees Classifier

```
Model DecisionTreeClassifier
Accuracy:  0.7186400937866354
              precision    recall  f1-score   support

          no       0.79      0.79      0.79       573
         yes       0.57      0.56      0.57       280

    accuracy                           0.72       853
   macro avg       0.68      0.68      0.68       853
weighted avg       0.72      0.72      0.72       853
```

KNN Classifier - K=5

```
Model KNeighborsClassifier
Accuracy:  0.671746776084408
              precision    recall  f1-score   support

          no       0.75      0.77      0.76       573
         yes       0.50      0.47      0.48       280

    accuracy                           0.67       853
   macro avg       0.62      0.62      0.62       853
weighted avg       0.67      0.67      0.67       853
```

Logistic Regression Classifier

```
Model LogisticRegression
Accuracy:  0.6998827667057445
              precision    recall  f1-score   support

          no       0.70      0.96      0.81       573
         yes       0.68      0.16      0.26       280

    accuracy                           0.70       853
   macro avg       0.69      0.56      0.54       853
weighted avg       0.69      0.70      0.63       853
```

Zero Padding Method

After zero padding our training data is of shape (18157, 9000) because the largest sentence is 90 words and each word has 100 vec representation

Now we try the same models used in our previous method

- Decision Trees
- KNN
- Logistic Regression

Decision Trees Classifier

```
Model DecisionTreeClassifier
Accuracy:  0.6975381008206331
              precision    recall  f1-score   support

          no       0.77      0.78      0.78       573
         yes       0.54      0.53      0.53       280

    accuracy                           0.70       853
   macro avg       0.66      0.65      0.65       853
weighted avg       0.70      0.70      0.70       853
```

Logistic Regression Classifier

```
Model LogisticRegression
Accuracy:  0.74091441969519 34
              precision    recall  f1-score   support

          no       0.77      0.88      0.82       573
         yes       0.65      0.45      0.53       280

    accuracy                           0.74       853
   macro avg       0.71      0.67      0.68       853
weighted avg       0.73      0.74      0.73       853
```

SubWord Text Encoder

TensorFlow offers a word encoder that assigns each word a unique value and also encodes sub words and characters, in order to be able to encode unknown words

```
import tensorflow_datasets as tfds
encoder = tfds.features.text.SubwordTextEncoder.build_from_corpus(corpus, target_vocab_size=10000)
```

Recurrent Neural Network

- Take the words order into consideration in a sentence
- We use the SubwordTextEncoder to give each word/subword a unique number
- Then we zero pad each sentence according to the max sentence length which was 113
- Then use the keras.Embedding layer to encode our data into vectors

Recurrent Neural Network - 2

- Embedding Layer with takes our vocab size as input and has 128 output nodes to be passed to the next layer
- LSTM layer with 128 input nodes (Long Short-Term Memory layer)
- Dense layer with 64 nodes and relu activation function
- Dense layer with 32 nodes and relu activation function
- Output which is a sigmoid function [0 and 1]
- We save the best validation f1 measure from 20 epochs

Recurrent Neural Network - 3

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, None, 128)         1275520
_________________________________________________________________
bidirectional_9 (Bidirection (None, 256)               263168
_________________________________________________________________
dense_27 (Dense)             (None, 64)                16448
_________________________________________________________________
dense_28 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_29 (Dense)             (None, 1)                 33
=================================================================
Total params: 1,557,249
Trainable params: 1,557,249
Non-trainable params: 0
_________________________________________________________________
```

Recurrent Neural Network - 4

```
Accuracy:  0.7776413
f1:  0.6841229
```

The RNN model gave the best results when compared to all previously mentioned methods with 68% f1 score for the positive class

Thanks!

Shams Eldin Sherif Hegab	4079
Ahmed Tayel	4219
Ahmed Nawar	4092